



Introduzione al corso

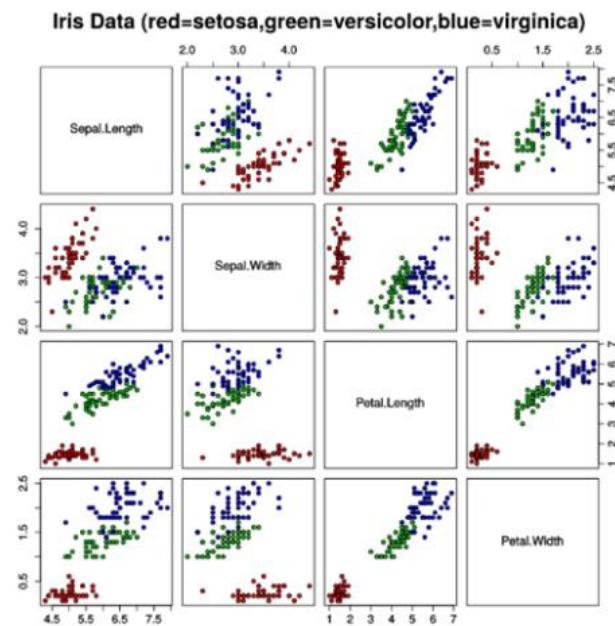
Machine Learning

Daniele Gamba

2022/2023

Benvenuti

Benvenuti al corso di Machine Learning!



From classification
to Gandalf in space



Chi sono

Daniele Gamba

Ingegnere Informatico Meccatronico @ Unibg

Founder e CEO di AI Sent Srl dal 2018

Per domande, ricevimenti, ecc.

daniele@aisent.io

meglio questa rispetto a quella @unibg almeno le vedo

AISent

AISent si occupa di ricerca, sviluppo e produzione di sistemi intelligenti per l'industria.

Creiamo nuovi algoritmi e sistemi industriali, dalla robotica al controllo qualità
dalla manutenzione predittiva alla ricerca delle cause.



Il corso

1) Machine learning

- Use of scikit-learn in Python
- Classification problems and methods
- Regression problems and methods
- Clustering problems and methods
- Dimensionality reduction
- Model selection
- Pre-processing

2) Deep learning

- Use of Tensorflow in Python
- Introduction to artificial neural networks
- Optimization algorithms
- Convolutional neural networks
- Recurrent neural networks

3) Other algorithms (Reinforcement learning, Generative models, Recommender systems, Causal Inference, etc.)

Libri

Libro di testo

Hands-on Machine Learning with Scikit-Learn,
Keras & Tensorflow, Aurélien Géron,

Third edition

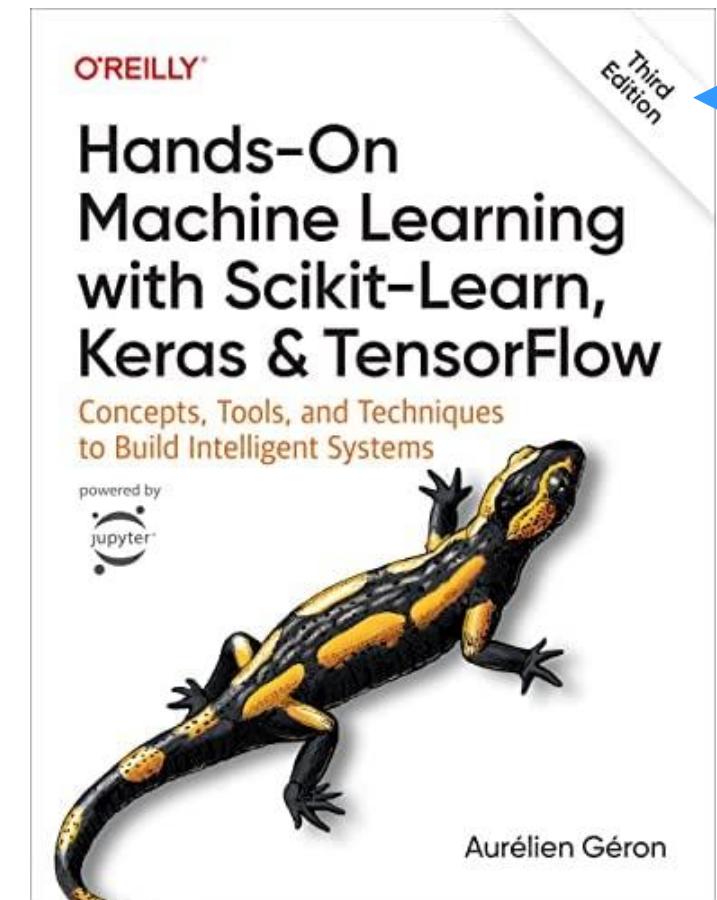
Raccomandati

Pattern Recognition and Machine Learning,

Cristopher M. Bishop

Probabilistic Machine Learning: An Introduction,

Kevin P. Murphy



Lezioni ed esame

Ci saranno 32 ore di lezione e 16 ore di esercitazione in aula con i vostri pc.

Useremo Google Colab, non avete bisogno di installare nulla, serve solo il vostro account unibg.

L'esame sarà scritto con 3 domande aperte.

Possibilità di fare un progettino che verrà valutato in alternativa a una domanda.

- Estiva - I Scritto: *27/6/2023, 16:30-18:30;*
- Estiva - II Scritto: *21/7/2023, 09:30-11:30;*
- Autunnale - I Scritto: *11/9/2023, 09:30-11:30;*



Prerequisiti

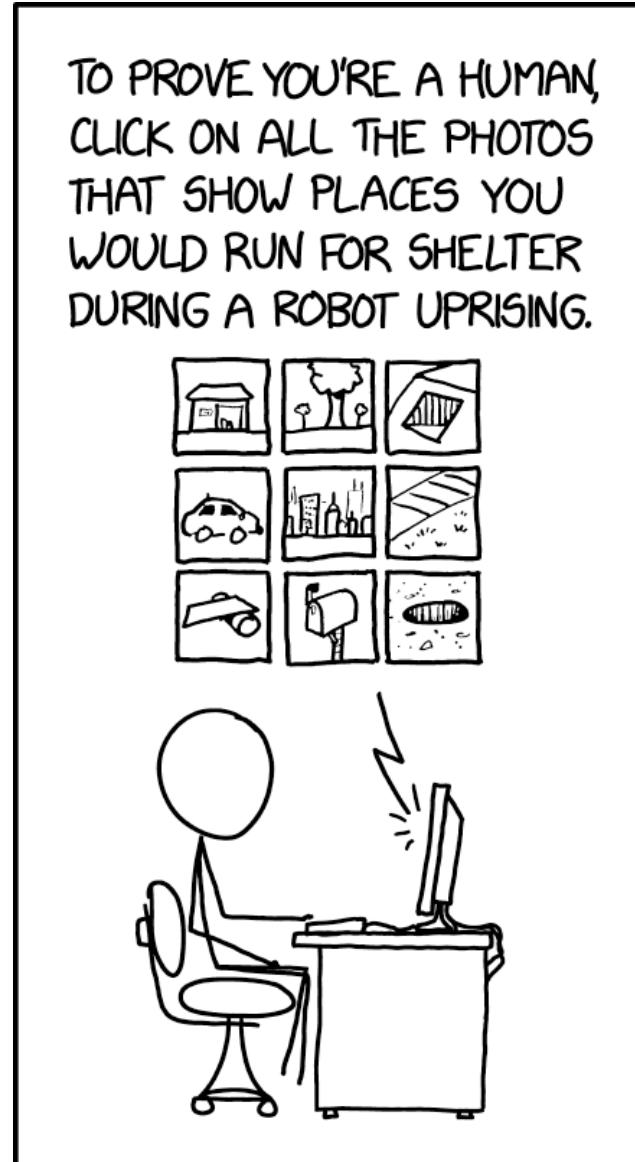
- Un pc
- I corsi che avete sicuramente dato di statistica ed analisi
- Minimo di conoscenza di Python

Se non conoscete Python, faremo una breve introduzione.

Tutorial base

<https://www.learnpython.org/>

Partiamo!





01 - Machine Learning

Daniele Gamba

2022/2023

Cos'è il Machine Learning

Iniziamo da qualche definizione

[Machine learning is the] field of study that gives computers the **ability to learn** without being explicitly programmed.

Arthur Samuel, 1959

A computer program is said to learn from **experience** E with respect to some **task** T and some performance **measure** P, if its performance on T, as measured by P, improves with experience E.

Tom Mitchell, 1997

Experience - E

Per poter imparare un algoritmo deve in qualche modo accumulare esperienza.

Nella maggior parte dei casi l'esperienza è rappresentata da un set di dati chiamato **dataset**.

Normalmente, rappresentiamo come X gli ingressi del nostro algoritmo, come Y le uscite.

Il dataset è l'insieme di entrambe, ad ogni ingresso associa una uscita (se presente).

Dataset - Esempi

	A	B	C	D	E	F
1	sepal_length	sepal_width	petal_length	petal_width	species	
2	5.1	3.5	1.4	0.2	setosa	
3	4.9	3	1.4	0.2	setosa	
4	4.7	3.2	1.3	0.2	setosa	
5	4.6	3.1	1.5	0.2	setosa	
6	5	3.6	1.4	0.2	setosa	
7	5.4	3.9	1.7	0.4	setosa	
8	4.6	3.4	1.4	0.3	setosa	
9	5	3.4	1.5	0.2	setosa	
10	4.4	2.9	1.4	0.2	setosa	
11	4.9	3.1	1.5	0.1	setosa	
12	5.4	3.7	1.5	0.2	setosa	
13	4.8	3.4	1.6	0.2	setosa	
14	4.8	3	1.4	0.1	setosa	
15	4.3	3	1.1	0.1	setosa	
16	5.8	4	1.2	0.2	setosa	
17	5.7	4.4	1.5	0.4	setosa	
18	5.4	3.9	1.3	0.4	setosa	
19	5.1	3.5	1.4	0.3	setosa	
20	5.7	3.8	1.7	0.3	setosa	
21	5.1	3.8	1.5	0.3	setosa	

Iris Dataset

X: lunghezza e larghezza di steli e petali)
Y: specie di Iris



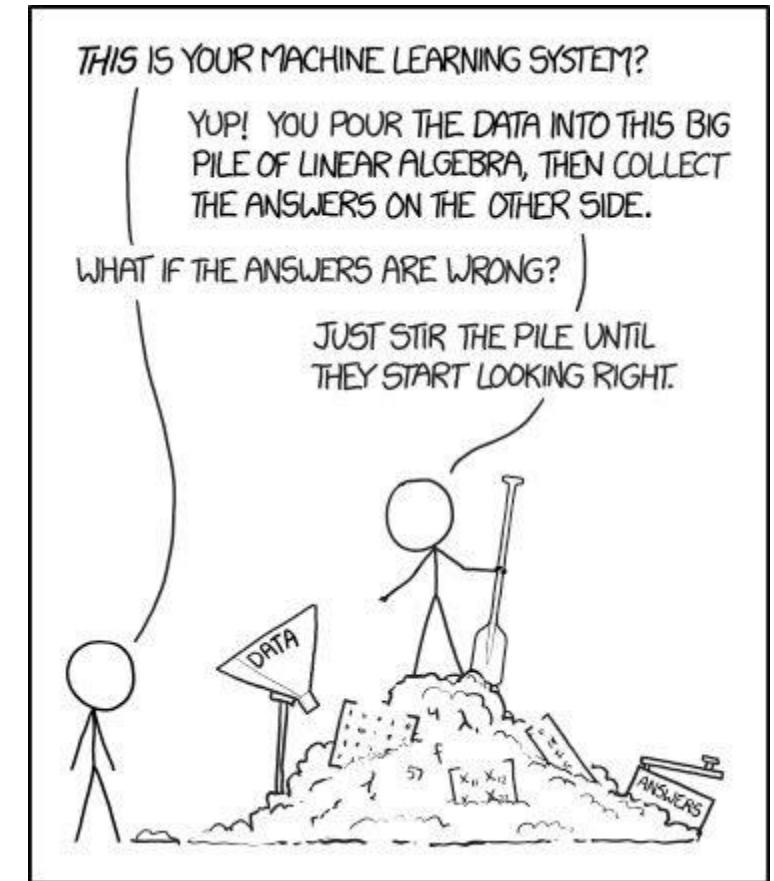
COCO Dataset

X: 200.000 immagini
Y: segmentazioni di 80 classi diverse

Dataset

La qualità del dataset è fondamentale.

È altrettanto importante avere abbastanza dati e che questi siano espressi in un modo «comodo» per far imparare gli algoritmi.



Task - T

Agli algoritmi possiamo chiedere di fare tante cose diverse
guidare una macchina, controllare la qualità di un pezzo, ottimizzare i consumi, cambiare lo sfondo ad un video, ecc.

Questi obiettivi sono la composizione di uno o più Task che diamo ad un algoritmo di machine learning.

Di task ne esistono molteplici e sempre di nuovi

Task - T

- **Regressione**: trovare i parametri che descrivono dei dati
- **Classificazione**: discriminare tra gruppi diversi
- **Forecasting**: prevedere dati futuri
- **Clustering**: trovare pattern diversi all'interno dei dati
- **Object detection**: trovare dove all'interno di un'immagine si trova un oggetto
- **Object segmentation**: trovare quali pixel appartengono all'oggetto
- **Anomaly detection**: trovare ciò che si scosta dalla normalità
- Ecc.

Measure - P

Per consentire ad un algoritmo di imparare dobbiamo dargli una metrica, una misura, che gli consenta di capire in che direzione è l'ottimo che cerchiamo.

Per questo si definiscono delle misure di performance che consentono all'algoritmo di imparare.

Anche qui ne esistono di diversi tipi

- Mean Absolute Error
- Mean Squared Error
- Binary Crossentropy
- Mean Average Precision
- ...

Esempio – Spam Filter

Un filtro anti-spam è un buon esempio di un algoritmo di machine learning.

Abbiamo una raccolta di mail classificate come buone e altre classificate come spam.

Il nostro **dataset** sarà composto da tutti i testi delle mail (X) e dalla loro classificazione buona/spam (Y).

L'algoritmo dovrà **classificare** un nuovo esempio e metterlo in uno dei due gruppi.

La misura della performance è l'**accuratezza** del modello, ovvero quante predizioni esegue correttamente sul totale.

Esempio – Spam Filter

Ora che abbiamo tutti i requisiti del problema (dataset, task e misura) non ci resta che risolverlo.

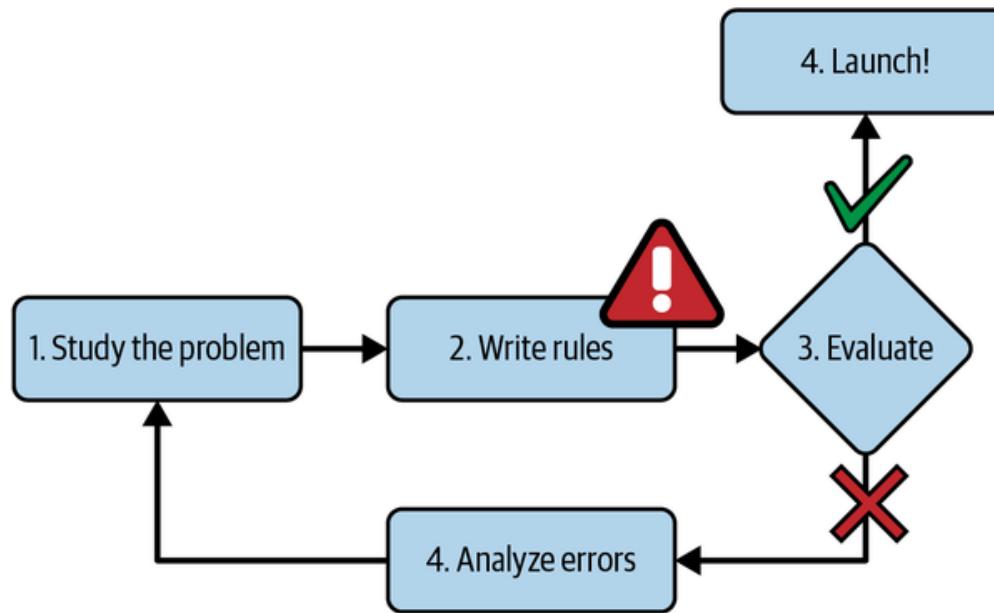
Se approcciassimo il problema in modo '**tradizionale**' dovremmo cercare qualche regola che riusciamo a codificare per discriminare tra le mail.

Un esempio potrebbe essere che se ci sono delle parole chiave tipo «hai vinto», «eredità del principe», «allungare appendici», allora mettiamo la mail nella cartella spam, altrimenti è buona.

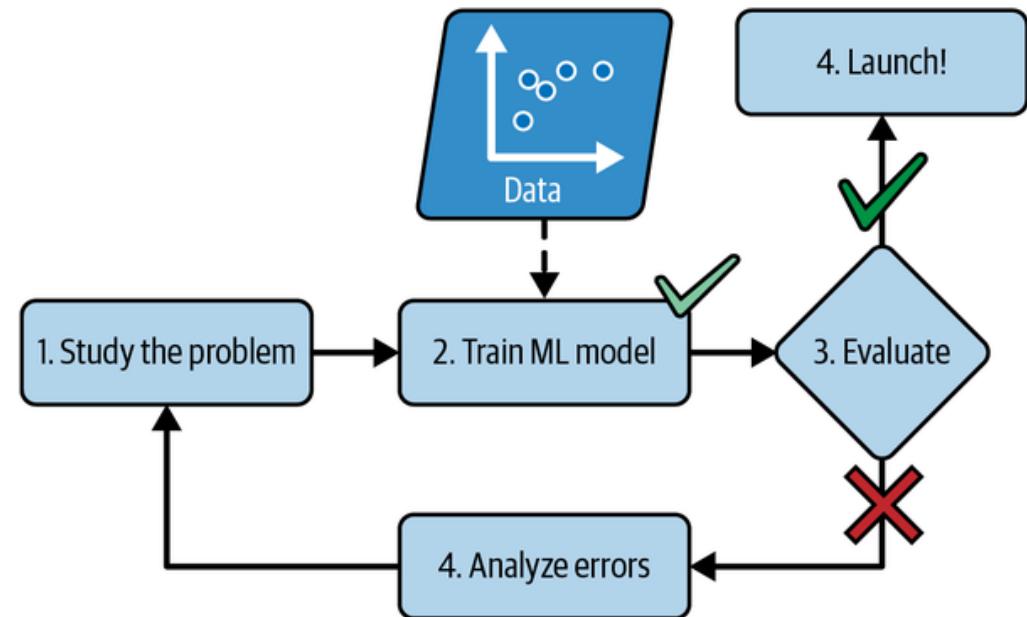
Questo approccio com'è facile immaginare è estremamente oneroso e poco flessibile, sarebbe molto facile per gli spammer aggirare con dei sinonimi le regole definite staticamente.

Programmazione vs Machine Learning

Approccio tradizionale



Approccio ML



Modello

Come avete visto il Machine Learning ci consente di sfruttare i dati a disposizione affinché un **modello** impari a risolvere il task al posto nostro.

Il modello è in generale una rappresentazione semplificata del mondo.

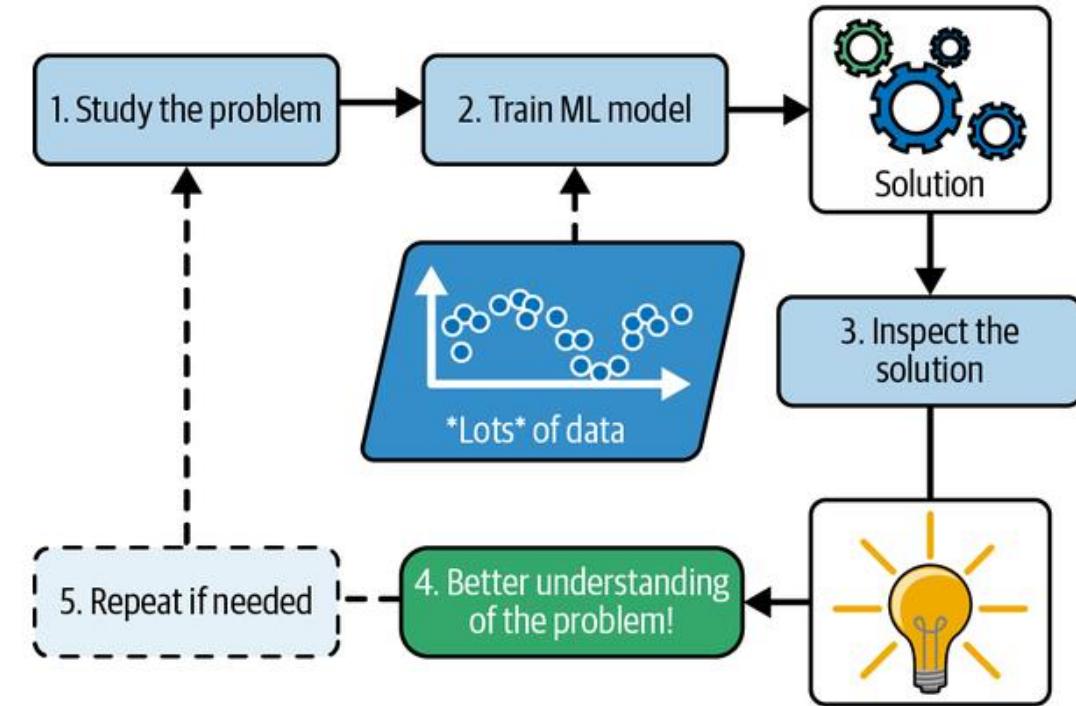
Una volta creato un modello del task possiamo sfruttarlo per risolvere il problema sul quale è stato addestrato.

Nel corso vedremo diversi modelli, dalle semplici regressioni alle reti neurali. Tutti i modelli che apprendono o fanno uso di dati sono modelli di Machine Learning.

Esempio – Spam Filter

Mettiamo che il nostro modello non sia altro che un Naive Bayes (spiegherò in dettaglio in seguito).

Andiamo a vedere le probabilità della frequenza di ciascuna parola, mail dove molte volte vengono ripetute ricchezza, denaro, vincite, ecc. verranno automaticamente classificate come spam senza che io debba scrivere delle regole specifiche affinché accada.



Tipi di Machine Learning

A seconda del problema che dobbiamo risolvere abbiamo a disposizione diversi algoritmi di ML

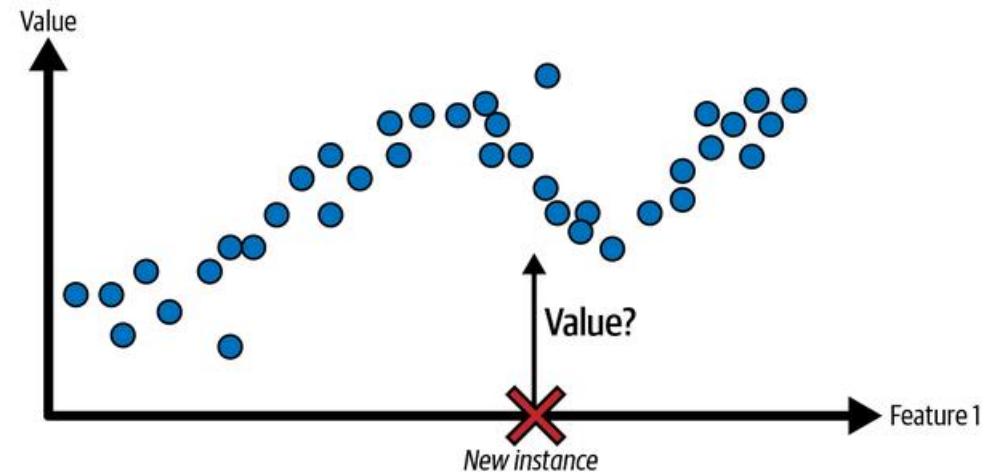
- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

Supervised Learning

È la più classica delle applicazioni, in cui il nostro dataset è costituito sia dalle X (ingressi) che dalle Y (uscite).

Lo spam-filter ne è un esempio, allo stesso modo se dovessimo prevedere il prezzo di una casa avremo a disposizione uno storico di metrature e zone (X) e dei prezzi (Y).

La maggior parte degli algoritmi di questo tipo si basano sul minimizzare la distanza tra la y vera e la \hat{y} predetta dal modello per fare **predizioni o classificazioni**.

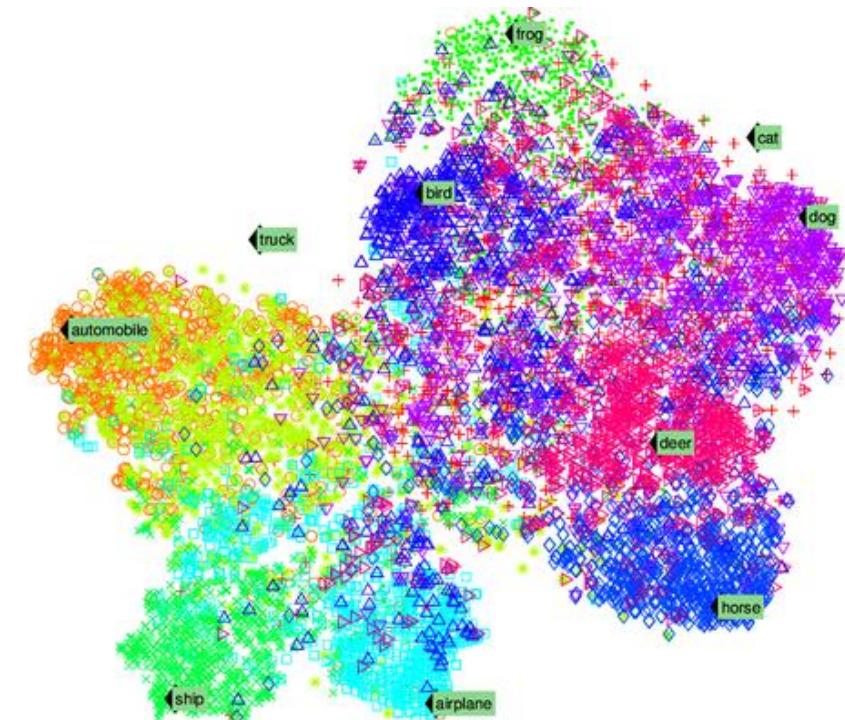


Unsupervised Learning

In questi casi abbiamo a disposizione solamente le X
ma non le Y.

Si vogliono cercare dei pattern nei dati che ci
consentano di capire a ritroso cosa contengono.

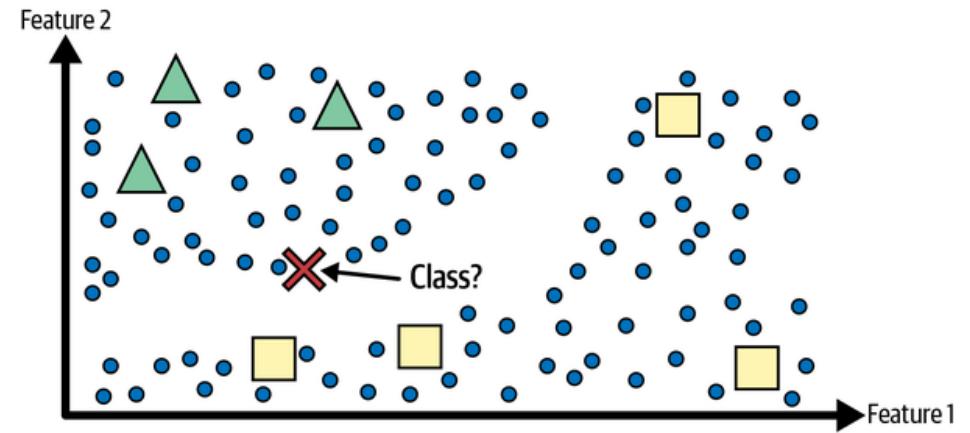
Un esempio ne è il **clustering**, in cui si cercano
gruppi di dati simili, o l'**anomaly detection**, in cui
andiamo ad osservare quelli che si scostano dalla
normalità.



Semi-supervised Learning

In questo caso abbiamo un sacco di dati senza etichette (label, Y, uscite) e alcuni esempi invece etichettati.

Nell'ultimo anno sta prendendo molto piede per sfruttare grandi basi di dati tramite tecniche di self-supervised learning, ovvero fornendo task diversi da quello da risolvere ma che consentono al modello di imparare la distribuzione dei dati per poi specializzarlo sui pochi esempi etichettati.

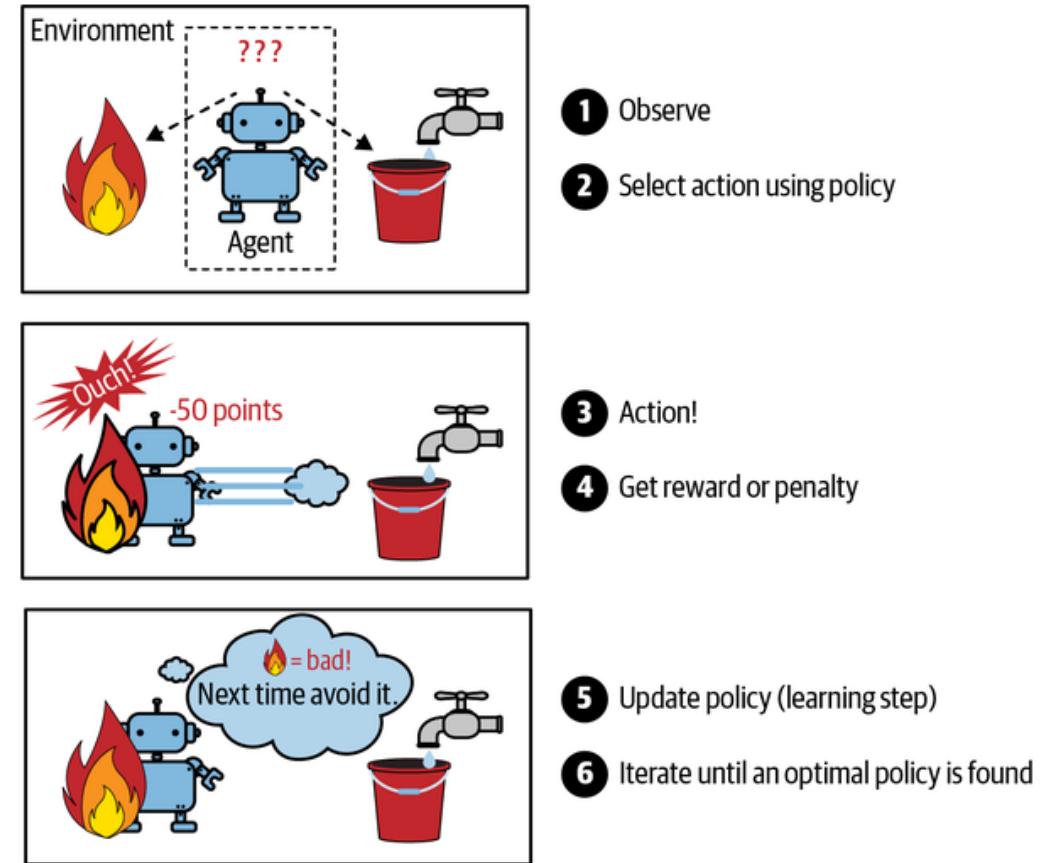


Reinforcement Learning

È la tipologia di ML che possiamo sfruttare quando abbiamo un ambiente con cui interagire.

L'algoritmo sperimenta e viene addestrato per 'reward', ne sono un esempio molte AI negli scacchi, go e altri giochi online.

L'algoritmo esplora diverse strategie e ottimizza rispetto al reward che gli viene fornito



Model based vs Instance based

Esiste una ulteriore distinzione

Model Based o parametrici

Ovvero algoritmi dove dai dati si apprende un modello che li descrive, una rappresentazione semplificata del problema da risolvere tramite l'apprendimento di alcuni parametri

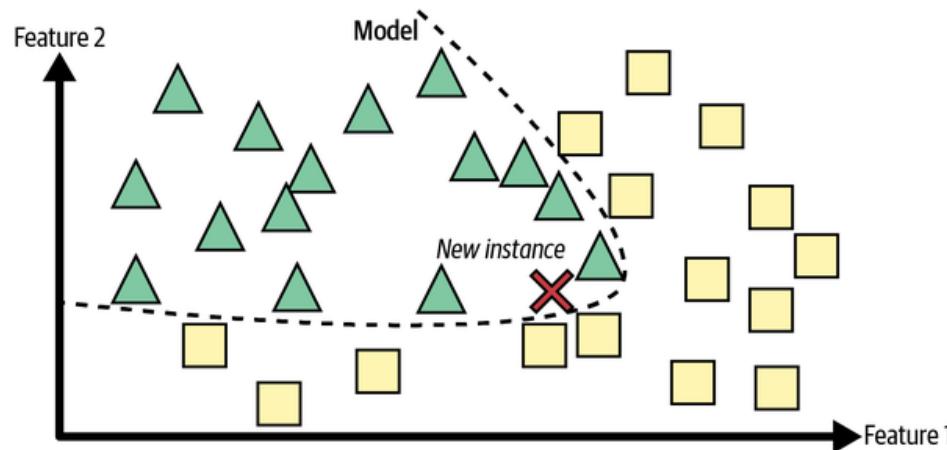
Instance Based o Non-parametrici

Algoritmi che non apprendono dei parametri ma sfruttano i dati stessi per confrontare un nuovo esempio tramite metriche di similarità.

Model based vs Instance based

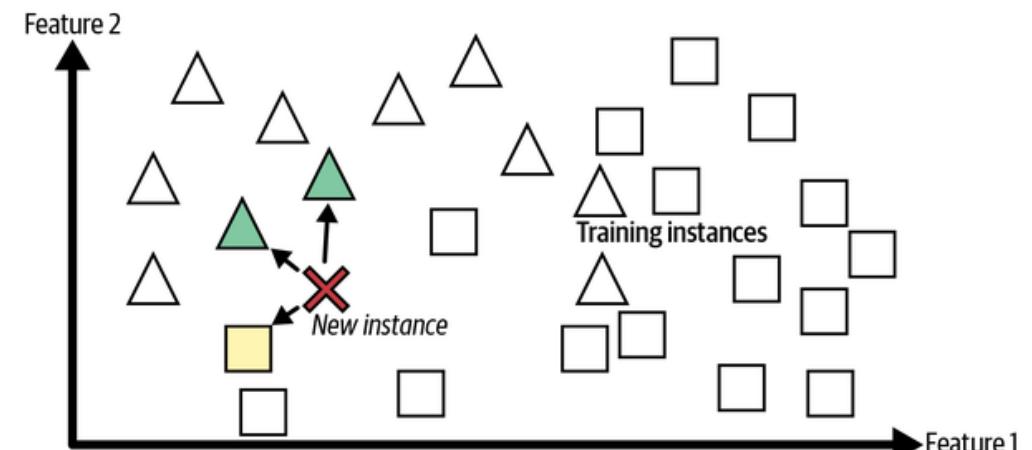
Model based - parametrico

La curva che separa le due classi sarà descritta da un modello $\alpha x_1^2 + \beta x_2^2 + \gamma$ con α, β, γ parametri

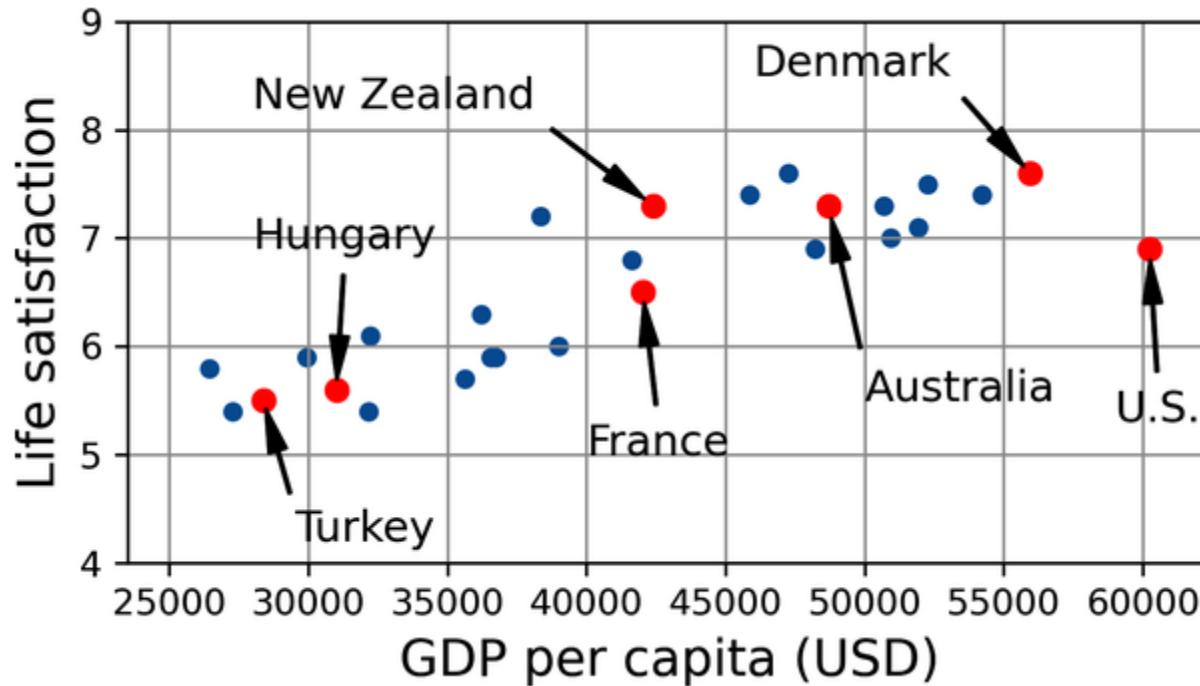


Intance based - non parametrico

Il nuovo esempio verrà confrontato con gli esempi vicini e classificato secondo la policy di somiglianza



E0101 – I soldi rendono più felici?



https://colab.research.google.com/github/ageron/handson-ml3/blob/main/01_the_machine_learning_landscape.ipynb



02 – Regression Models

Daniele Gamba

2022/2023

Premessa

Ogni autore, framework ed esercitazione usa convenzioni diverse per indicare features, parametri e matrici.

Per orientarvi nelle slide ricordate che solitamente

- x, u, φ rappresentano ingressi, input, features
- β, θ, ω rappresentano i parametri
- $\hat{y}, y(\omega)$ rappresentano le nostre stime
- x è il singolo dato, X è il vettore delle variabili di un esempio, \mathbf{X} la matrice completa

Nel caso non capiste a cosa fa riferimento una equazione chiedetemi senza problemi

Non vi chiederò una notazione piuttosto che l'altra purché si capisca

Regressione

Lo scopo della regressione è identificare i parametri che descrivono il **modello generatore dei dati**.

Generalmente parliamo di regressione per problemi **continui di supervised learning**.

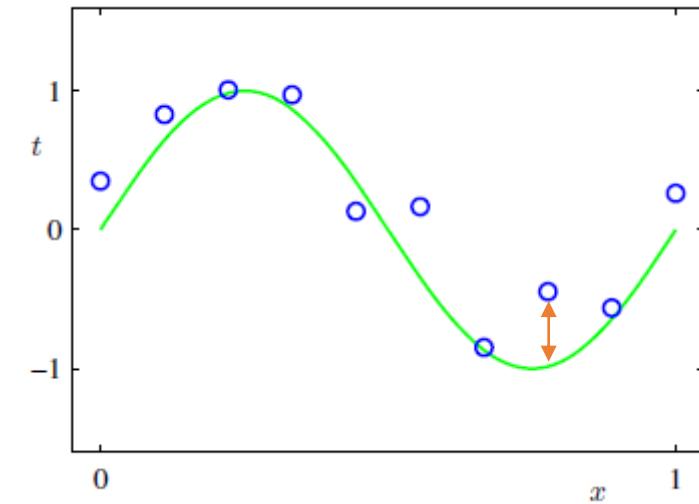
Questo argomento dovrebbe esser già stato trattato ampiamente in altri corsi (sotto) e quindi sarà trattato brevemente

- IMAD
- Modelli Stocastici
- Statistical Learning

Errore

Tutti i dati sono afflitti da diversi tipi di errore

- Errori di misura
- Errori di quantizzazione
- Errori di rappresentazione
- Errori umani
- ...



L'obiettivo è lasciare nel termine di errore solamente questi contributi, identificando correttamente gli altri parametri. Come avrete visto in Modelli Stocastici, uno dei modi per valutare se gli errori contengono altre informazioni è fare un «test di bianchezza», o assicurarsi che la correlazione tra errore e la nostra y sia nulla.

Modello

Il classico modello di regressione è descritto da

$$y = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_0 + e$$

dove

- y è la nostra uscita
- x_1, x_2 sono le variabili in ingresso
- θ_1, θ_2 sono i parametri che stiamo cercando, θ_0 è un parametro costante chiamato anche intercetta
- e è l'errore irriducibile dei dati

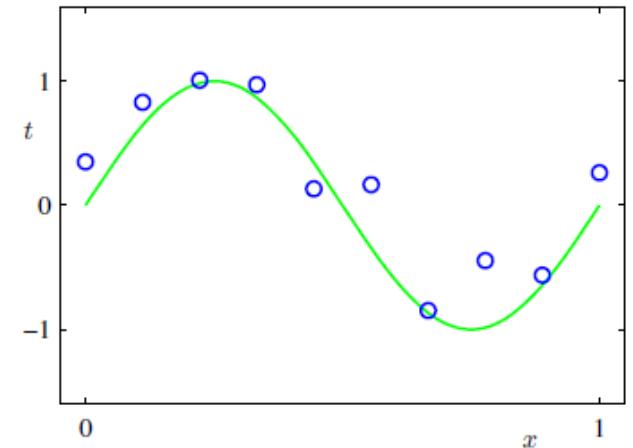
Immaginiamoci di dover prevedere l'indice di soddisfazione di uno stato data il suo Prodotto Interno Lordo pro capite.

Regressione polinomiale

Riprendiamo il grafico di prima e assumiamo di dover regredire il modello solamente tra una x ed una y (Single Input, Single Output).

Dato che abbiamo solamente una variabile possiamo applicarvi una serie di **funzioni di base** che descrivano comportamenti diversi.

Come **features** possiamo usare lo sviluppo polinomiale della nostra variabile in ingresso.



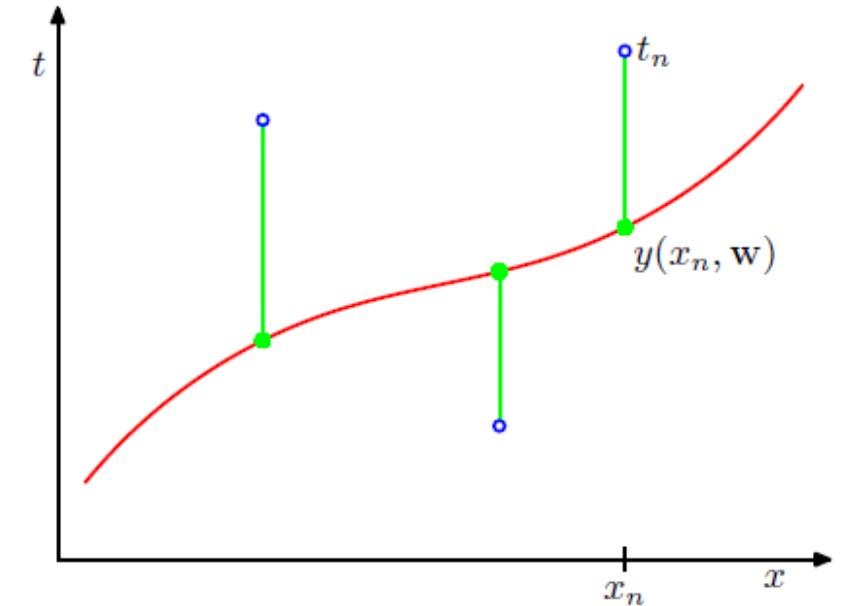
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j$$

Regressione polinomiale

Definita la nostra funzione, andiamo a definire la nostra metrica.

In molti casi, ma non tutti, è molto comodo per i problemi di regressione usare il Mean Squared Error, l'errore quadratico medio.

Andiamo quindi a definire come

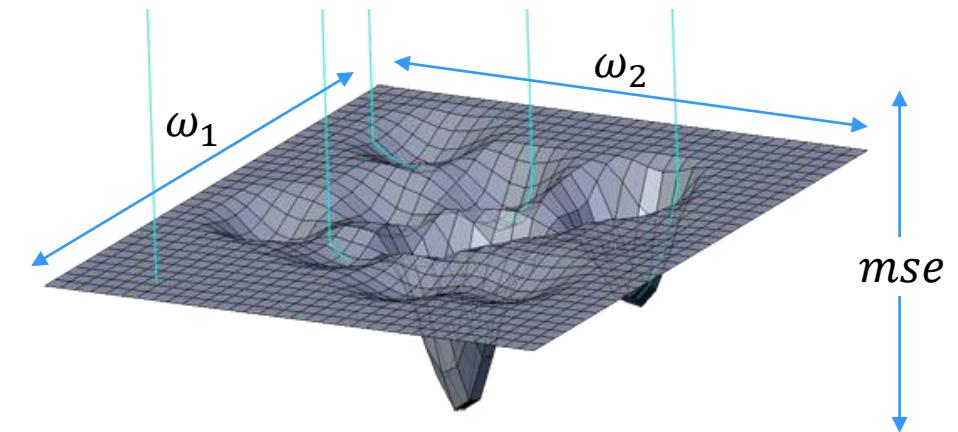


Spazio delle soluzioni / parametri

A seconda dei valori ω_1, ω_2 , ecc. avremo una diversa performance, ovvero sarà diverso l'errore tra le nostre predizioni ed i dati reali.

Esistono combinazioni diverse di parametri che danno la stessa performance, esiste solitamente una sola soluzione ottima, cioè con il minor valore assoluto di errore.

Normalmente si ricercano i parametri ottimi per ottimizzazione numerica tramite discesa del gradiente.



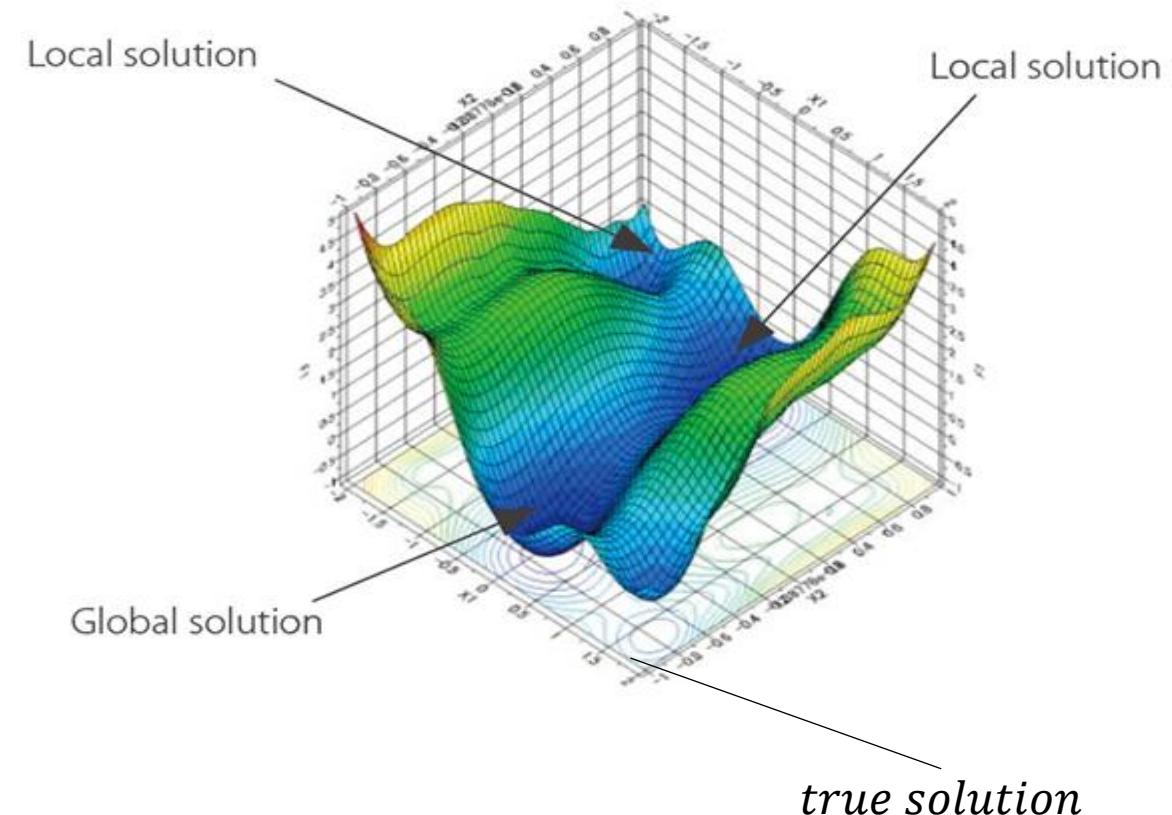
Spazio delle soluzioni / parametri

A seconda dei parametri avremo ottimi locali o globali.

L'ottimo raggiungibile non per forza coincide con l'ottimo reale.

Non è detto che riusciremo mai a risolvere esattamente un problema perché

- Non abbiamo abbastanza informazioni
- Non esiste una soluzione numerica al problema
- C'è troppo rumore nei dati



Ordinary Least Squares

Solo nel caso di **regressioni lineari con mse** (mean squared error) possiamo trovare in un colpo solo la soluzione ottima tramite gli OLS.

Se riscriviamo il nostro problema come

$$\mathbf{X} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

$$S(\boldsymbol{\beta}) = \sum_{i=1}^n \left| y_i - \sum_{j=1}^p X_{ij} \beta_j \right|^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2.$$

allora possiamo ottenere i **parametri ottimi** in un colpo solo

$$(\mathbf{X}^\top \mathbf{X}) \hat{\boldsymbol{\beta}} = \mathbf{X}^\top \mathbf{y}.$$

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

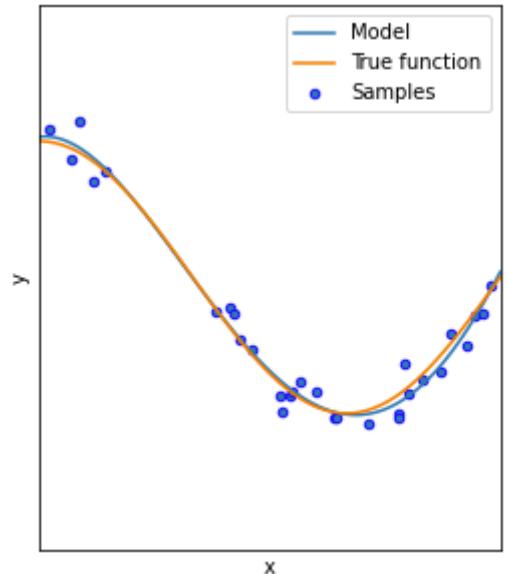
Regressione polinomiale

Concludiamo l'esempio della regressione polinomiale con un rapido accenno alla selezione delle features che vedremo meglio nella lezione successiva.

Se abbiamo

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

a che grado di x arriviamo per descrivere y ?

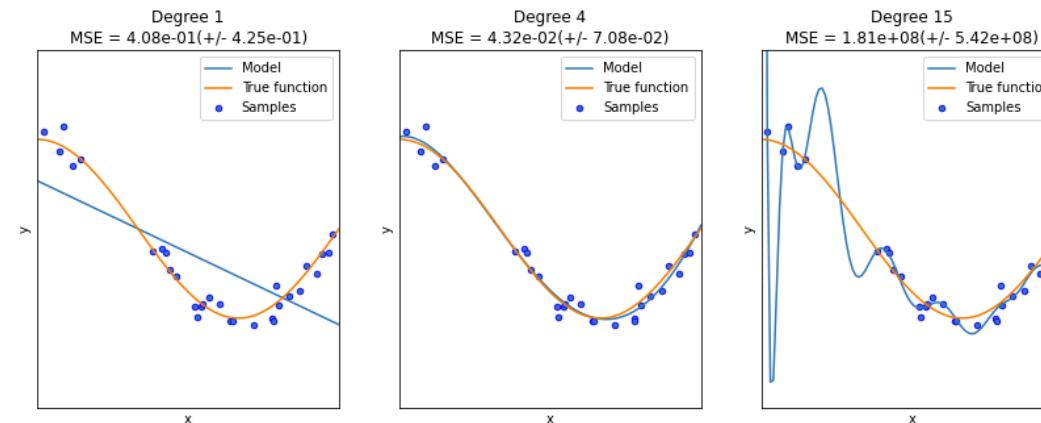


Regressione polinomiale – Overfitting e Underfitting

Ci troviamo davanti al problema dell'overfitting o dell'underfitting.

Scegliere le giuste features, tra le variabili e le funzioni di base da applicare alle variabili, è difficile.

Metterne troppe significherà over-fittare il modello e sbagliare ad imparare il modello dei dati, metterne troppe poche ci fornirà una performance molto bassa. Vediamolo con un esempio pratico



https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=AqQ_mcg-m0yi



03 – Regularization & Model selection

Daniele Gamba

2022/2023

Extra

E se conoscessimo la funzione di base che genera i nostri dati?

<https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=duRMAKBDGsxo>

Link al Teams con le slide

[t.ly/oMf0](https://teams.microsoft.com/l/meetup-join/19%3a9f3a2a2a-1a0c-4e0d-90a0-1a0a2a2a2a2a/0)



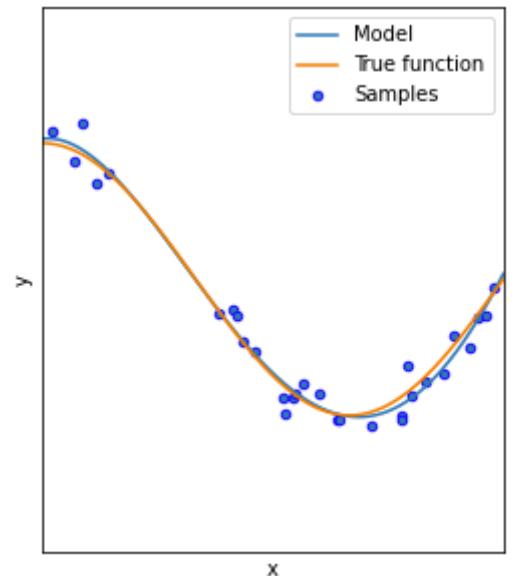
Regressione polinomiale

Riprendiamo l'esempio della regressione polinomiale con un rapido accenno alla selezione delle features che vedremo meglio nella lezione successiva.

Se abbiamo

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

a che grado di x arriviamo per descrivere y ?

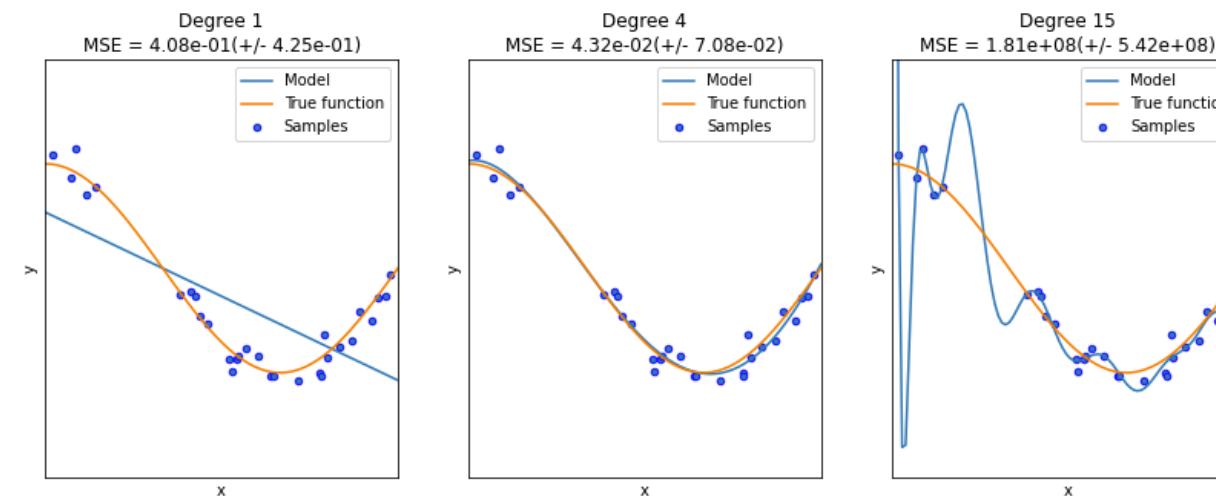


Regressione polinomiale – Overfitting e Underfitting

Ci troviamo davanti al problema dell'overfitting o dell'underfitting.

Scegliere le giuste features, tra le variabili e le funzioni di base da applicare alle variabili, è difficile.

Metterne troppe significherà over-fittare il modello e sbagliare ad imparare il modello dei dati, metterne troppe poche ci fornirà una performance molto bassa.



Overfitting

Partiamo dal concetto di overfitting.

Il nostro modello sta cercando di minimizzare l'errore anche a discapito dell'apprendere un modello performante. Non conoscendo a priori quanto è complesso il modello che genera i dati cerca di ottimizzare finché può.

Più features abbiamo o meno dati per feature abbiamo, più è alto il rischio di avere overfitting.

Possiamo quindi

- Cercare di raccogliere più dati
- Selezionare meglio le nostre features
- Limitare l'apprendimento del modello

Più dati?

Se avessimo un problema da cui possiamo facilmente campionare più dati è meglio raccoglierne il più possibile, purché siano IID, ovvero **indipendenti e identicamente distribuiti**.

Nel caso campionassimo più volte dati non indipendenti non aggiungeremmo nessuna informazione al modello.

Domande

- Campionare frame diversi da uno stesso video è utile?
- Eseguire più volte lo stesso esperimento scientifico?

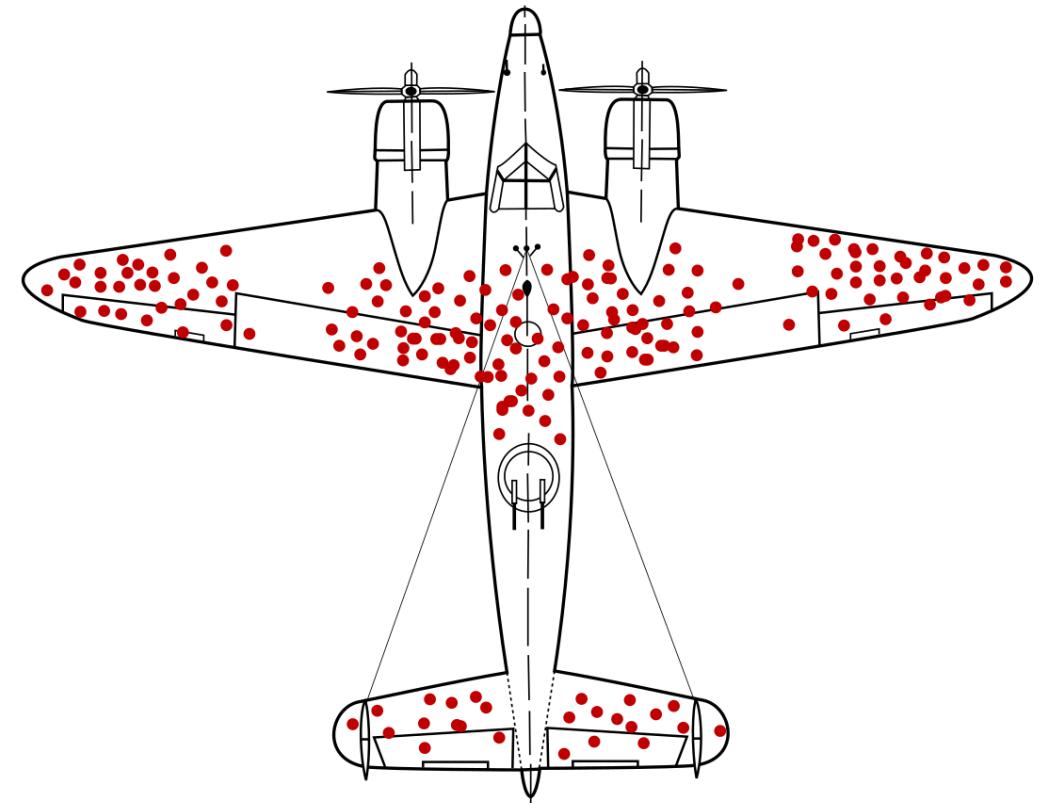
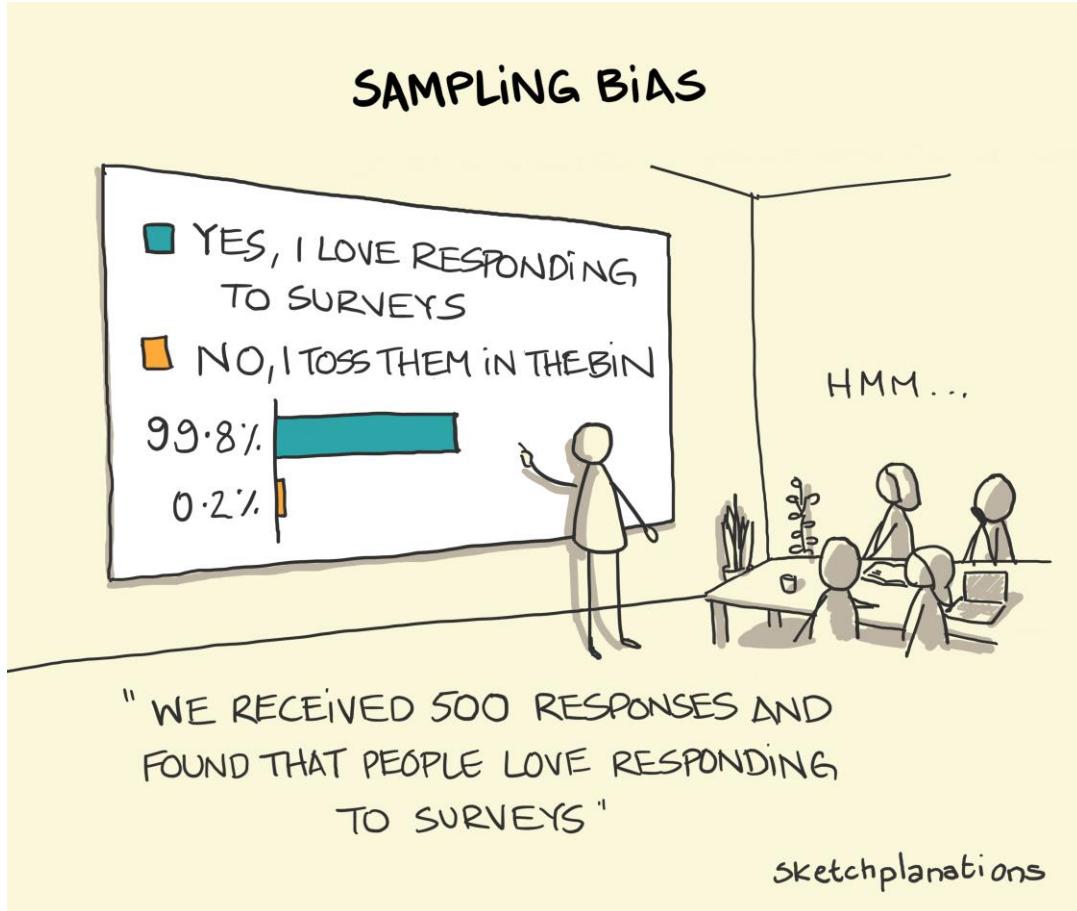
La maggior parte delle volte non è possibile aver accesso a più dati, perché magari è molto costoso raccoglierli ed etichettarli o perché non è proprio possibile.

Problemi dei dati

Abbiamo visto che i dati contengono **errori** di diversa natura, questo pone agli algoritmi una serie di sfide

- **Dati insufficienti** (es. addestrare un LLM su 10 frasi)
- **Dati di scarsa qualità** (es. pieni di errori di annotazione)
- **Dati non rappresentativi** (es. acquisiti in condizioni diverse dalla produzione)
- **Sampling (selection) bias** (next slide)
- **Features irrilevanti**
- **Features confondenti** (lo vedremo alla fine del corso)

Sampling Bias



Distribuzione dei fori di proiettili sugli aerei rientrati dalle missioni

Regola del pollice

La regola del pollice è di avere 20 esempi per parametro.

Nel nostro caso della regressione polinomiale, se avessimo avuto 20 esempi potevamo usare al più 1 grado di libertà (ovvero la media), con 40 potevamo aggiungere la dipendenza lineare. Per la performance migliore del modello di quarto ordine avremmo dovuto avere 100 esempi.

Non è sempre possibile avere 20 esempi per parametro, quindi dobbiamo studiare tecniche che ci consentano di essere robusti anche se non raggiungiamo questa soglia.

Features selection

Un primo modo è selezionare solamente le features che hanno più *significatività* per il problema che stiamo provando a risolvere.

Meno features = meno capacità espressiva del modello ma più esempi per parametro.

Come possiamo selezionare le diverse features?

Un modo è prendere solamente le features con la correlazione con l'uscita più alta (Es. Pearson's Correlation Coefficient PCC).

Un altro modo è utilizzare tecniche di **regolarizzazione** che ci consentano di mandare a zero i termini meno «utili».

Regolarizzazione

Ritornando al nostro esempio della regressione polinomiale, se andiamo a vedere i coefficienti di ciascuna feature osserviamo che i parametri esplodono di grandezza con il crescere del numero di features.

```
Grado 1  
↳ [array([-1.60931179]),  
 array([-7.31956683,  5.55955392]),  
 array([ 0.46754142, -17.78954475,  23.5926603 , -7.26289872]),  
 array([-2.98291188e+03,  1.03898766e+05, -1.87415056e+06,  2.03715125e+07,  
       -1.44872551e+08,  7.09311979e+08, -2.47064676e+09,  6.24558367e+09,  
       -1.15676035e+10,  1.56894317e+10, -1.54005437e+10,  1.06456871e+10,  
       -4.91375763e+09,  1.35919168e+09, -1.70380199e+08])]  
  
Grado 15
```

Ridge Regression

Possiamo quindi ipotizzare di mettere un termine che vada a pesare negativamente l'esplosione dei parametri, aggiungendo un termine di **penalità** alla nostra funzione di costo.

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

Questa regressione è chiamata **Ridge Regression** o **L2** e tiene conto del quadrato della norma dei parametri.

È l'unico altro caso in cui possiamo usare OLS per ottenere i \mathbf{w}^* ottimi per il nostro problema.

Ridge Regression

Grazie alla Ridge possiamo comandare la complessità del modello solamente da un parametro λ .

A fianco un esempio di come diversi valori di λ impattano sui parametri imparati da una regressione polinomiale di ordine fisso.

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

Lasso Regression

Possiamo estendere il caso di della penalizzazione della norma a qualsiasi ordine q .

Nel caso generale

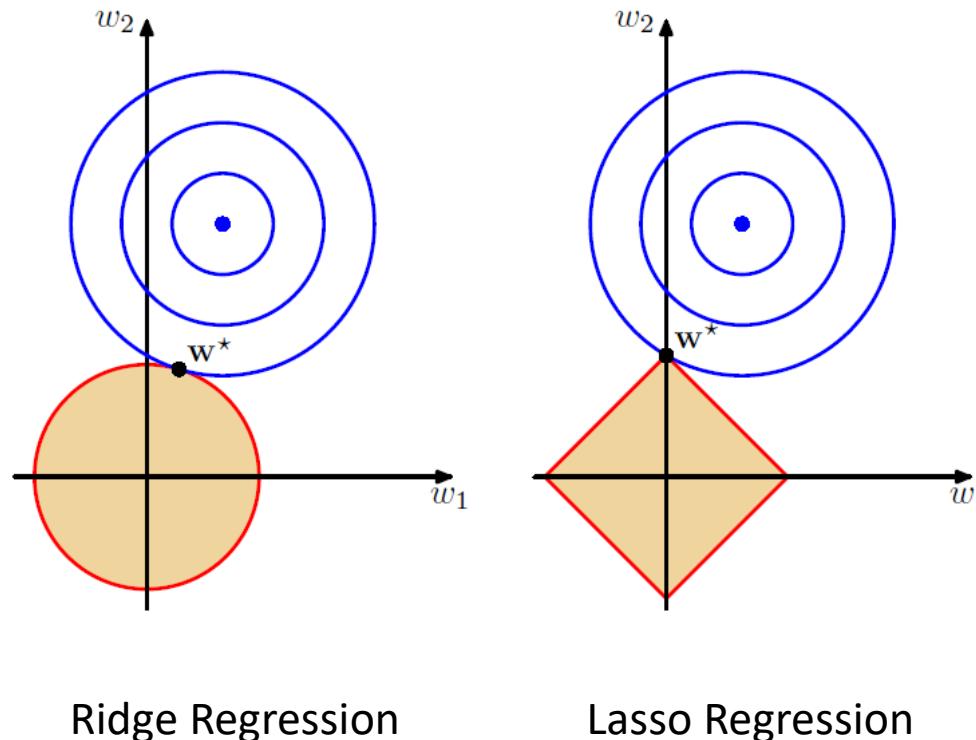
$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$

Il caso speciale $q=1$ si chiama **Lasso Regression** o **L1**.

La Lasso si ottiene per ottimizzazione numerica tramite iterazioni (vedremo poi come).

Lasso Regression

La Lasso ha lo svantaggio di non poter usare gli OLS per trovare la soluzione ottima ma ha il grosso vantaggio che tende a mandare a zero le features meno significative.



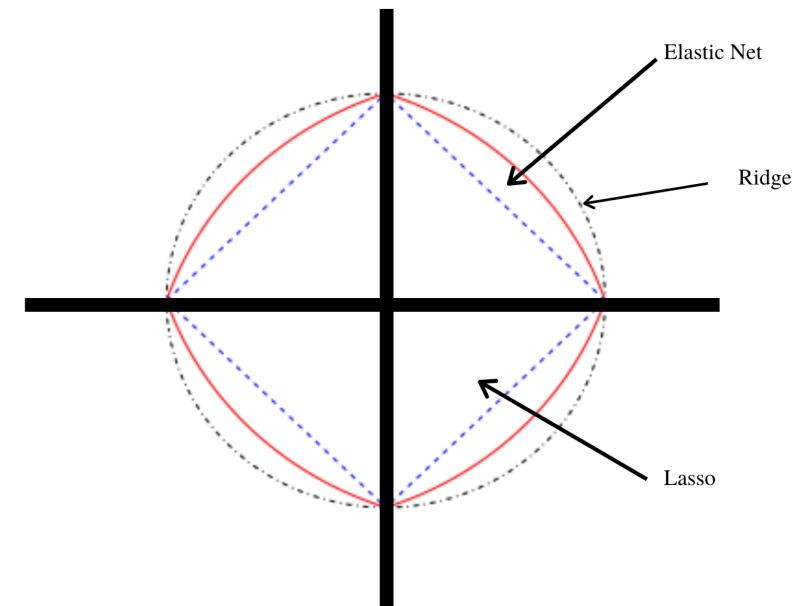
Ridge Regression

Lasso Regression

Elastic Net

La regolarizzazione Elastic Net combina linearmente la Ridge e la Lasso in un'unica equazione.

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}} (\|y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1).$$



È utilizzata in una tecnica non parametrica che vedremo più avanti chiamata Support Vector Machine.

Regressione penalizzata

Vediamo ora in pratica come si comportano Ridge e Lasso sui parametri della nostra regressione polinomiale della lezione precedente.

<https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=duRMAKBDGsxo>

Akaike Information Criterion

Un altro metodo per penalizzare i modelli è andando a valutare il guadagno che si ha ad aggiungere una feature in più sul modello. Supponendo di avere d parametri e N dati, si aggiunge alla funzione di costo la dipendenza al numero di features.

$$\text{AIC}(d) = 2 \cdot \frac{d}{N} + \ln [E(\hat{\theta}_N; d)]$$

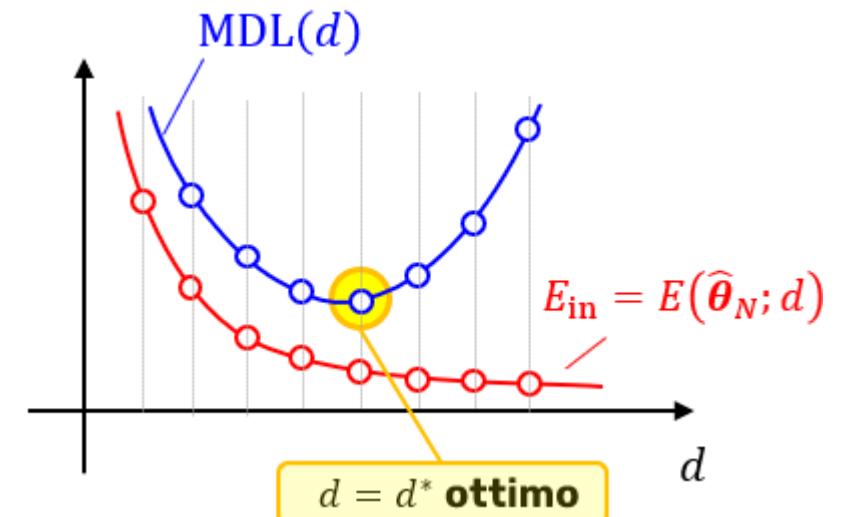
Dove $E(\hat{\theta}_N; d)$ è la nostra misura di performance. Più parametri abbiamo più il valore d/N cresce e penalizza la nostra performance.

Information Criteria

Oltre alla Akaike, esistono altri criteri basati sulle «informazioni» che andiamo ad aggiungere al modello, tra cui Bayesian Information Criterion (BIC) e Minimum Description Length (MDL).

In tutti i casi si vanno a costruire dei modelli che penalizzano l'aggiunta di parametri in funzione dei dati e della loro capacità di espressione.

$$\text{MDL}(d) = \ln[N] \cdot \frac{d}{N} + \ln[E(\hat{\theta}_N; d)]$$



Recap

Fino ad ora abbiamo visto dei semplici modelli di regressione lineare.

Anche a fare una semplice regressione dobbiamo stare attenti a non overfittare i nostri dati.

Esistono una serie di tecniche che ci consentono di controllare la complessità del nostro modello.

Fin'ora abbiamo visto solo tecniche che intervengono sul modello, e se intervenissimo sui dati?

Train, validazione, test

La validazione è l'operazione di misurare le performance del nostro modello su un set di dati sui quali non ha fatto ottimizzazione.

Nel caso in cui abbiamo abbastanza dati torna molto comodo dividere i nostri dati in

- **Train:** il dataset sul quale il nostro modello impara
- **Validation:** il dataset che usiamo per fare model selection
- **Test:** il dataset sul quale misuriamo solamente la nostra performance finale

I tre dataset devono esser campionati dallo stesso processo, altrimenti prenderemo decisioni sbagliate (es. per un sampling bias tutti i casi difficili finiscono nel test set).

Train, validazione, test

Se abbiamo tanti dati possiamo dividere il nostro dataset in modo **random** tra i 3 dataset, normalmente si usano percentuali tipo

- 60, 20, 20
- 70, 15, 15
- 70, 20, 10

Il dataset di training avrà la maggior quantità dovendo basare il nostro addestramento solamente su quello.

In alcuni casi può aver senso riguardare a posteriori le distribuzioni dei dati in modo da essere sicuri che siano distribuiti coerentemente.

Model selection

Nel momento in cui andiamo ad addestrare più modelli di regressione si sceglierà il più performante sul dataset di validazione.

Esempio

- Creiamo modelli di regressione polinomiali dal grado 2 al grado 15
- Misuriamo la performance di ciascun modello sul dataset di validazione
- Prendiamo il modello più performante sulla validazione
- Misuriamo la sua performance sul test, quel risultato *dovrebbe essere* la performance del modello in produzione

Usare il dataset di test per scegliere il modello migliore equivale a «barare» finché non mettete in produzione il modello.

Una volta scelto il modello migliore possiamo riaddestrarlo su training e validazione.

Validazione

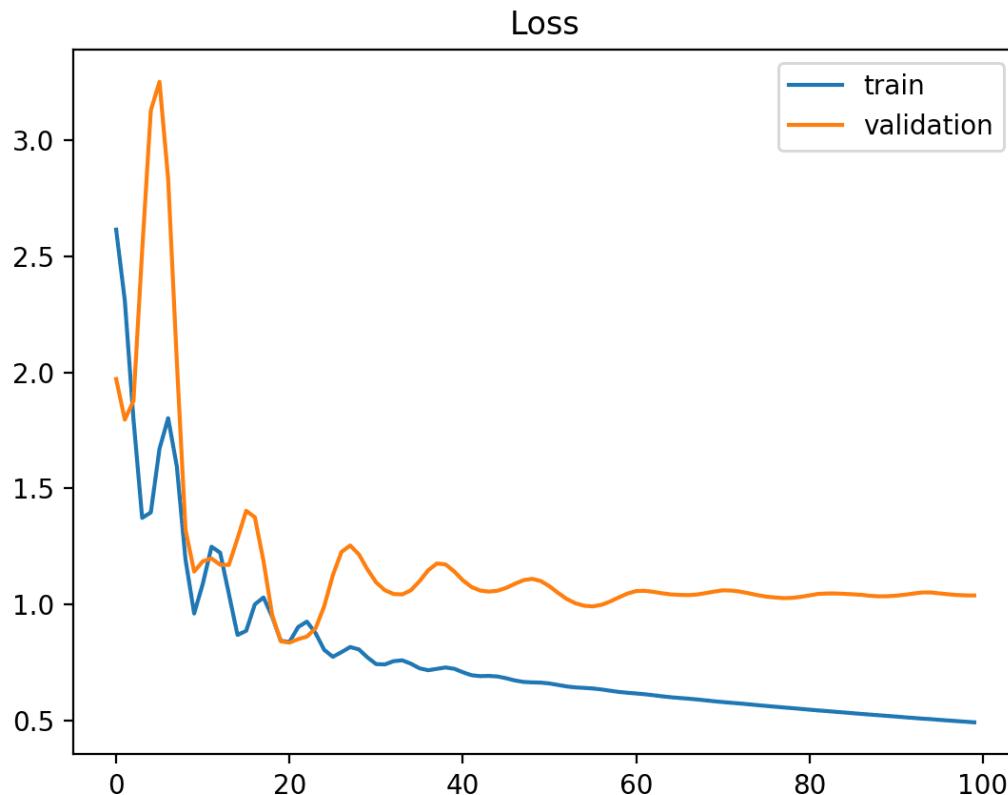
Il dataset di validazione a volte è chiamato anche out-of-sample e la sua performance è indicata come

$$MSE_{val} \text{ o } MSE_{out}$$

nel caso della MSE.

Nel caso dell'apprendimento iterativo, per discesa del gradiente, possiamo misurare la performance sul dataset di validazione anche durante l'apprendimento. In questo modo valuteremo se e quando il modello inizia a performare.

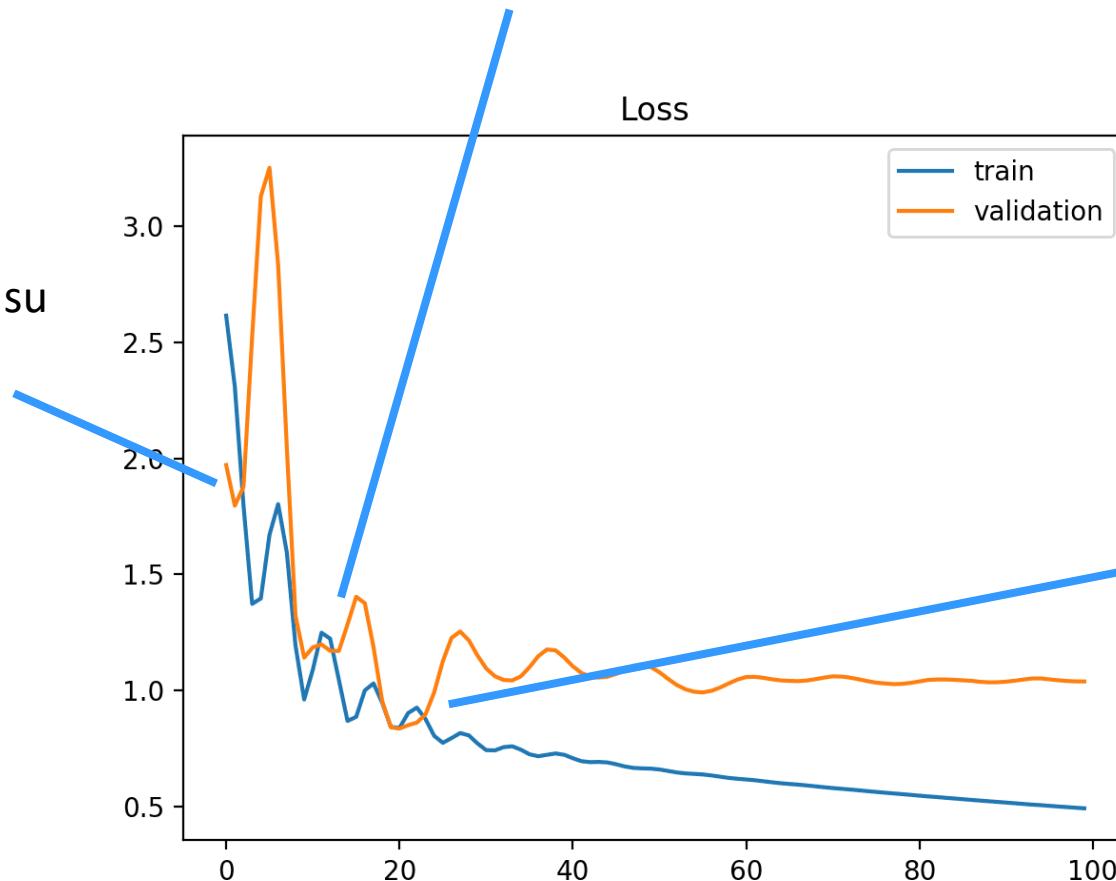
Validazione



Validazione

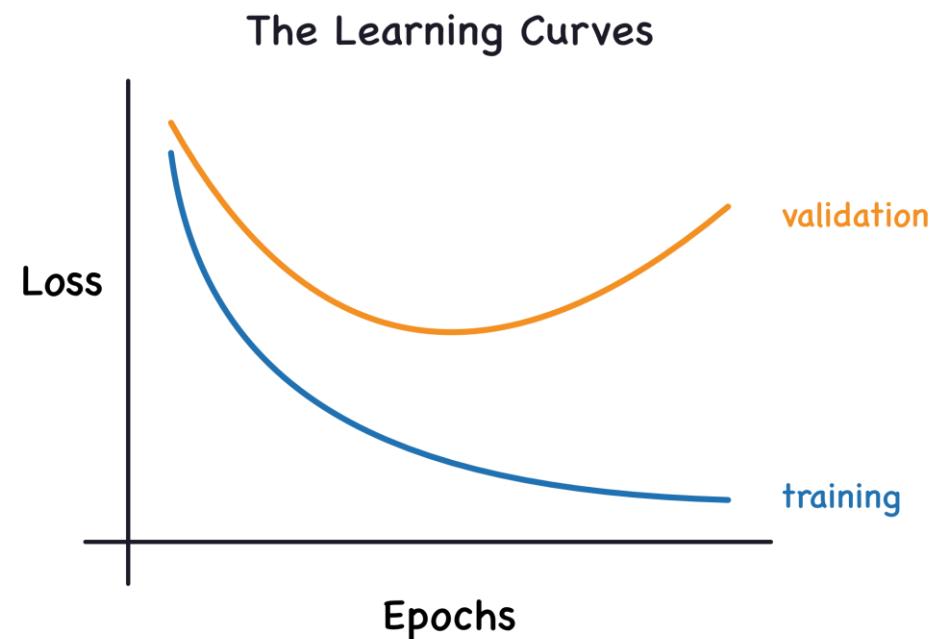
Inizio dell'apprendimento, i pesi random del modello possono performare meglio su validazione che sul training

ottimizzazione

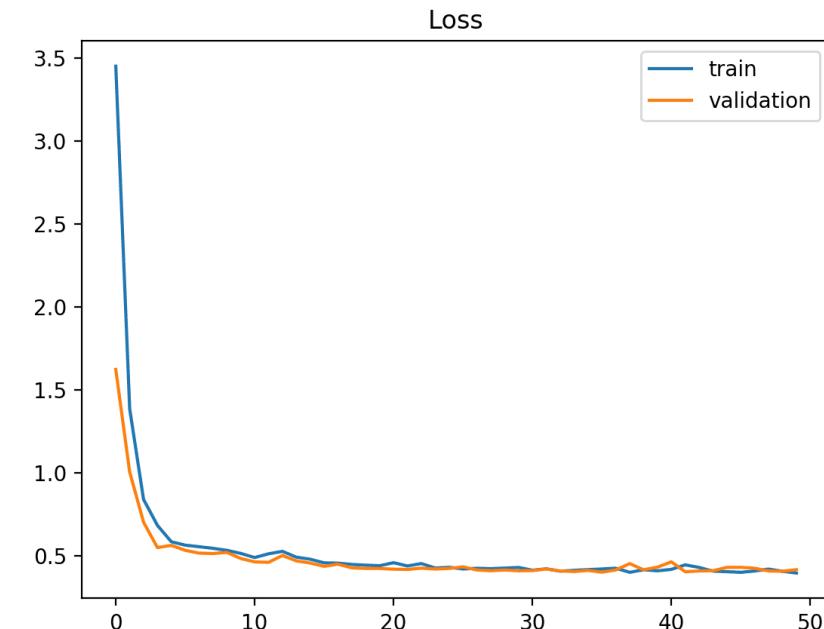


Training e validazione iniziano a divergere, il modello inizia a overfittare il training e perde di generalizzazione su dati non visti

Casi



Overfitting -> validazione cresce
Early Stop

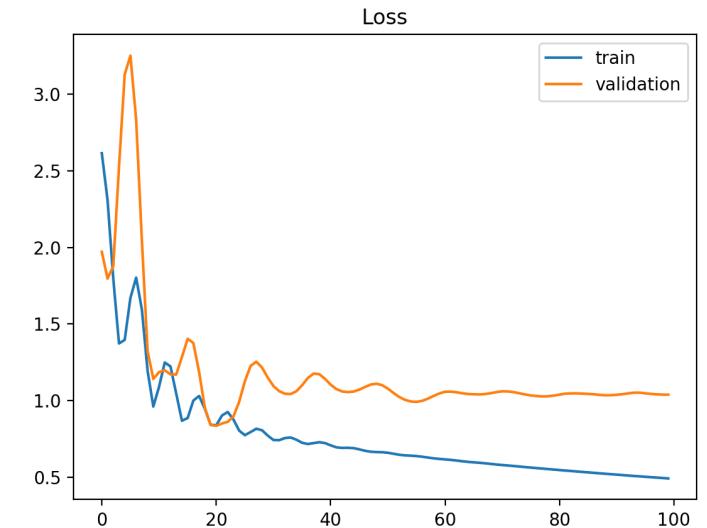


Underfitting o modello ideale-> validazione e
training non hanno differenze

Validazione durante il training

Ad ogni iterazione stiamo confrontando un nuovo modello (nei parametri) con i dati di validazione.

Nel caso di apprendimento iterativo dovremo confrontare non solo i modelli durante il loro training, ma anche tra tutti i possibili altri modelli.



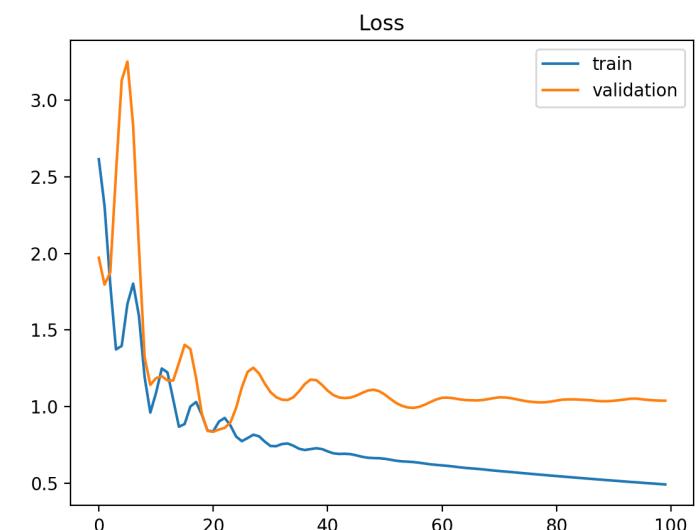
Early Stop – Best in validation

Esistono diverse tecniche per prendere un buon modello durante l'addestramento

- **Early stop**, ovvero fermiamo l'addestramento quando osserviamo divergere train e validazione
- **Best validation**, ovvero ci salviamo il modello che performa meglio in validazione

Entrambe le tecniche sono molto prone a noise nei dati e a momenti «fortunati» in cui i nostri parametri esprimono meglio la varianza dei dati in validazione rispetto al training.

Specie in questi casi osservare il grafico di traing-validation **ci deve guidare** nel scegliere il modello e dimensionarlo opportunamente (quali features, quanti neuroni – vedremo poi, ecc.)



Cross-validation

Nel caso in cui non abbiamo abbastanza dati per dividere in modo aprioristico tra train, validazione e test, dobbiamo utilizzare altre tecniche che ci consentono di validare le nostre performance.

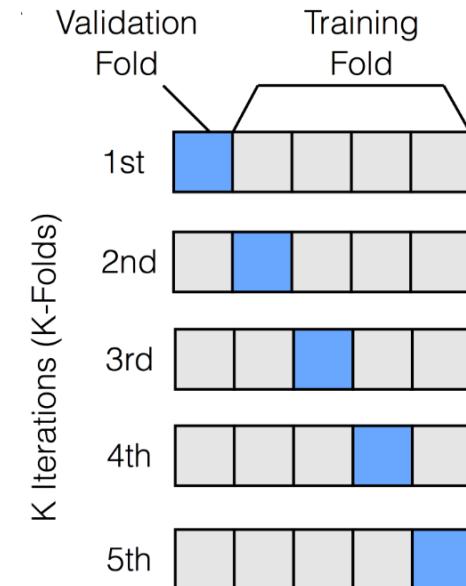
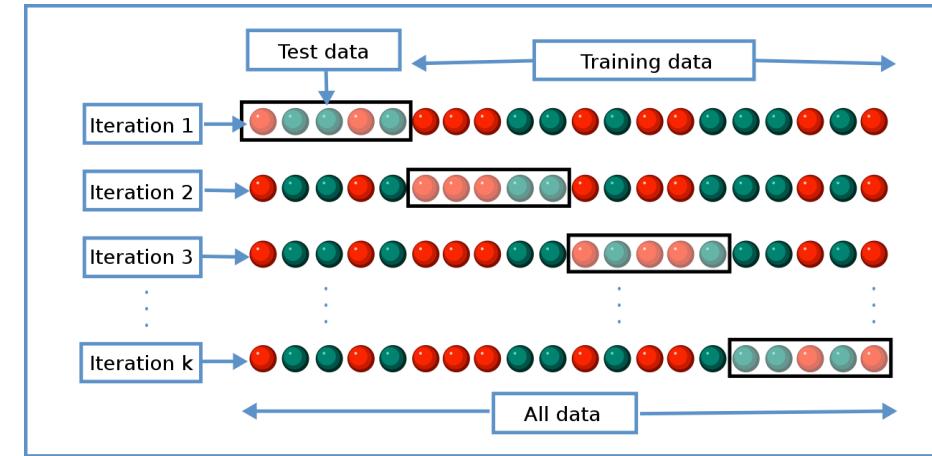
La cross-validation consiste nel riutilizzare un pezzo del training set stesso per validare le performance ri-addestrando non uno ma N modelli.

Fare cross-validation impone quindi di fare N addestramenti e può risultare più onerosa in termini di computazione.

k-fold Cross-Validation

Nel caso della **k-fold cross-validation** andiamo a dividere il nostro dataset in k gruppi, ed eseguiamo k volte l'addestramento su $\frac{N}{k}(k - 1)$ dati e misuriamo la performance sugli ultimi $\frac{N}{k}$ dati.

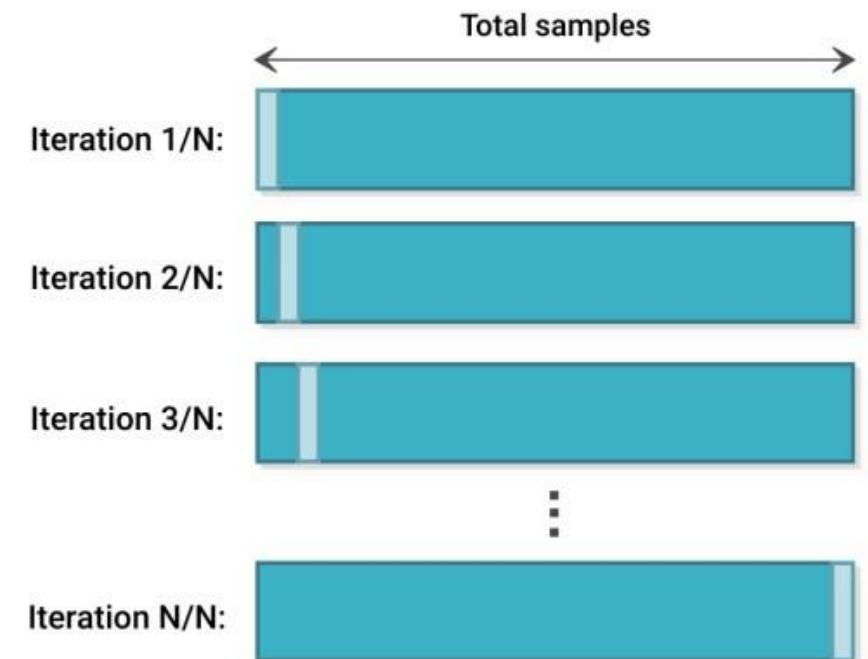
La nostra performance in cross-validazione sarà la media delle performance dei k test.



Leave-one-out

Nel caso in cui abbiamo **davvero pochi dati**, ma proprio pochi per cui anche dividere per k risulterebbe in un set di training troppo piccolo, possiamo portare $k = N$ ed eseguire N training lasciando fuori ogni volta un solo dato.

La nostra performance in cross-validation sarà la media di tutti gli errori di predizione sui singoli esempi esclusi dal training.



Validazione e cross-validation

Ora che sappiamo come misurare la capacità del modello di generalizzare a dati non osservati riprendiamo l'esempio della regressione polinomiale e andiamo ad osservare come il modello che visivamente è più performante lo è anche in cross-validation.

<https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=duRMAKBDGsxo>

Recap

Abbiamo visto

- Come andare ad intervenire sui dati per scegliere il modello migliore
- Come leggere le differenze tra training e validazione
- Come portare avanti l'analisi con pochi dati

Vediamo tutto assieme in un esempio

[https://colab.research.google.com/github/ageron/handson-
ml3/blob/main/04_training_linear_models.ipynb#scrollTo=vk57NkQKoS0j](https://colab.research.google.com/github/ageron/handson-ml3/blob/main/04_training_linear_models.ipynb#scrollTo=vk57NkQKoS0j)



04 – Classification

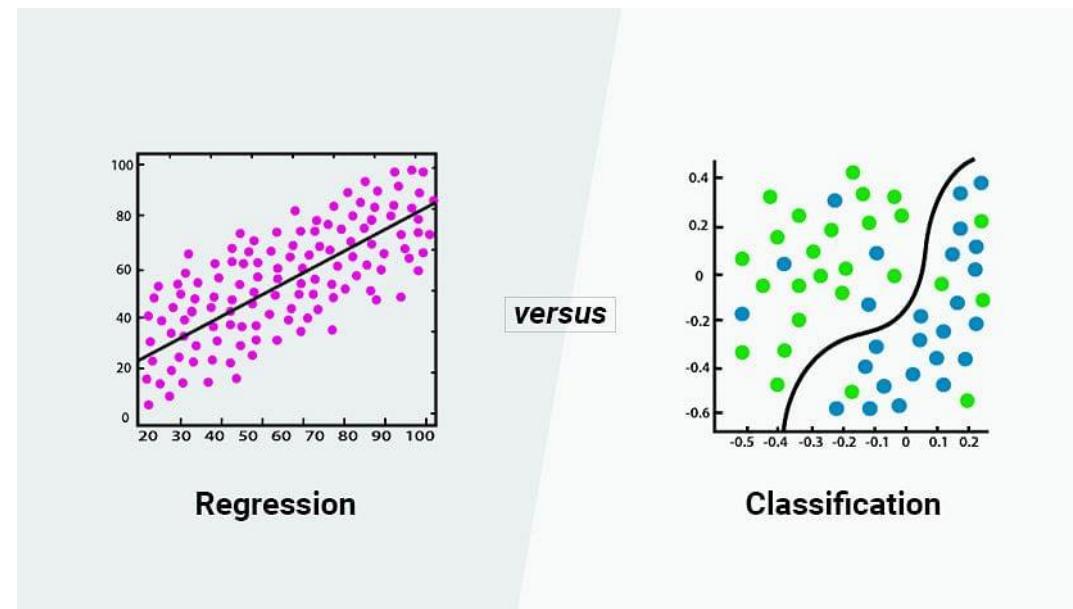
Daniele Gamba

2022/2023

Classificazione

I problemi di **classificazione** sono problemi in cui dobbiamo determinare a che classe appartiene un esempio.

Determinare la razza di un gatto o se un paziente è sano o malato sono problemi di classificazione.



Classificazione

Un modello di classificazione avrà quindi un set di features e in uscita avrà una o più classi.

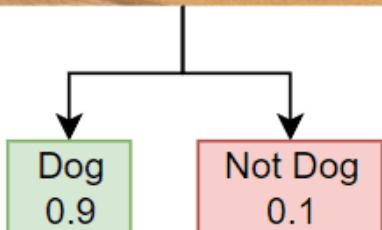
La classificazione potrà essere

- **Binary**, nel caso abbiamo due classi (0/1) o one-vs-rest (es. capelli neri vs altro)
- **Multi-class**, nel caso in cui abbiamo N classi tra cui scegliere (es. siamese, persiano, birmano, ...)

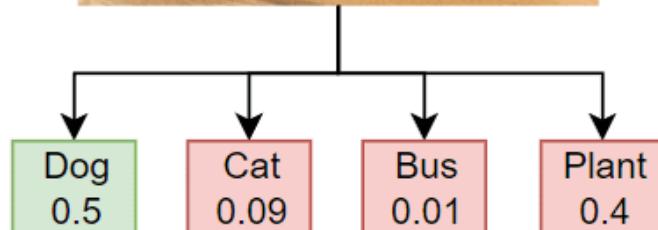
Inoltre un problema può anche essere **multi-label** nel caso in cui il nostro esempio può appartenere a più classi contemporaneamente

Classificazione

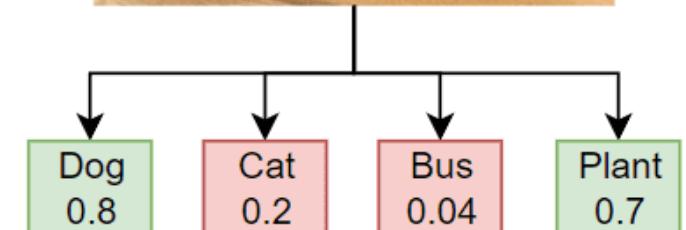
Binary Classification



Multiclass Classification



Multilabel Classification



Performance

Nella regressione abbiamo visto che ci sono diverse metriche di performance, riprendendole

Mean Squared Error

La principale, anche nella versione Root MSE è la più robusta e ci consente di usare OLS

Mean Absolute Error

permette di avere dati più sporchi e per un livello di «dettaglio più alto» nei modelli generativi

Mean Absolute Percentage Error

Più difficile da far convergere ma ottima per i problemi in cui l'errore percentuale è importante.

Confusion Matrix

Allo stesso modo per i problemi di classificazione abbiamo le misure di performance.

		True condition			
		Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR}^+}{\text{LR}^-}$
		False negative rate (FNR), Miss rate $= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

Accuracy, F1-score

L'accuratezza rappresenta il totale di quanti prediciamo correttamente sul totale.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

È la metrica più importante ma sbaglia tanto più le classi sono sbilanciate.

Un modello che dato il giorno ti dice che non è mai il giorno di Natale avrà una accuratezza del 99.7%.

Per questo ci servono altre metriche, tra cui l' F_1 score definito come media armonica di precision e recall.

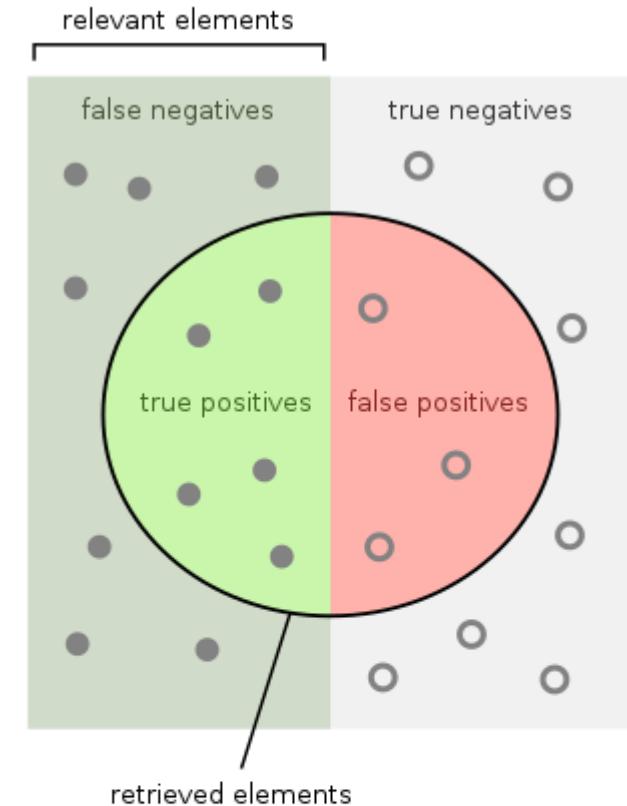
$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2\text{tp}}{2\text{tp} + \text{fp} + \text{fn}}.$$

Precision, Recall

Oltre alla accuratezza (Accuracy) totale del modello ci interessa spesso andare a valutare

Precision

Quanti predetti corretti sul totale di giusti e falsi positivi
«Quante in quante immagini in cui dico che c'è un cane c'è davvero un cane?»



Recall

Quanti predetti corretti sul totale di giusti e falsi negativi.
«Quante immagini di cani riesco ad identificare sul totale delle immagini di cani?»

How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{green}}{\text{green} + \text{red}}$$

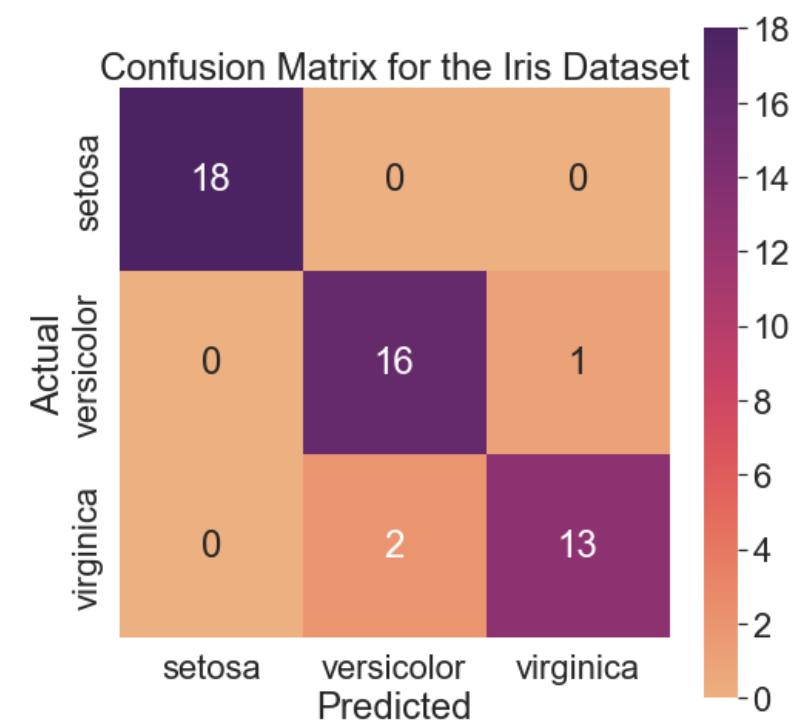
How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{green}}{\text{green} + \text{grey}}$$

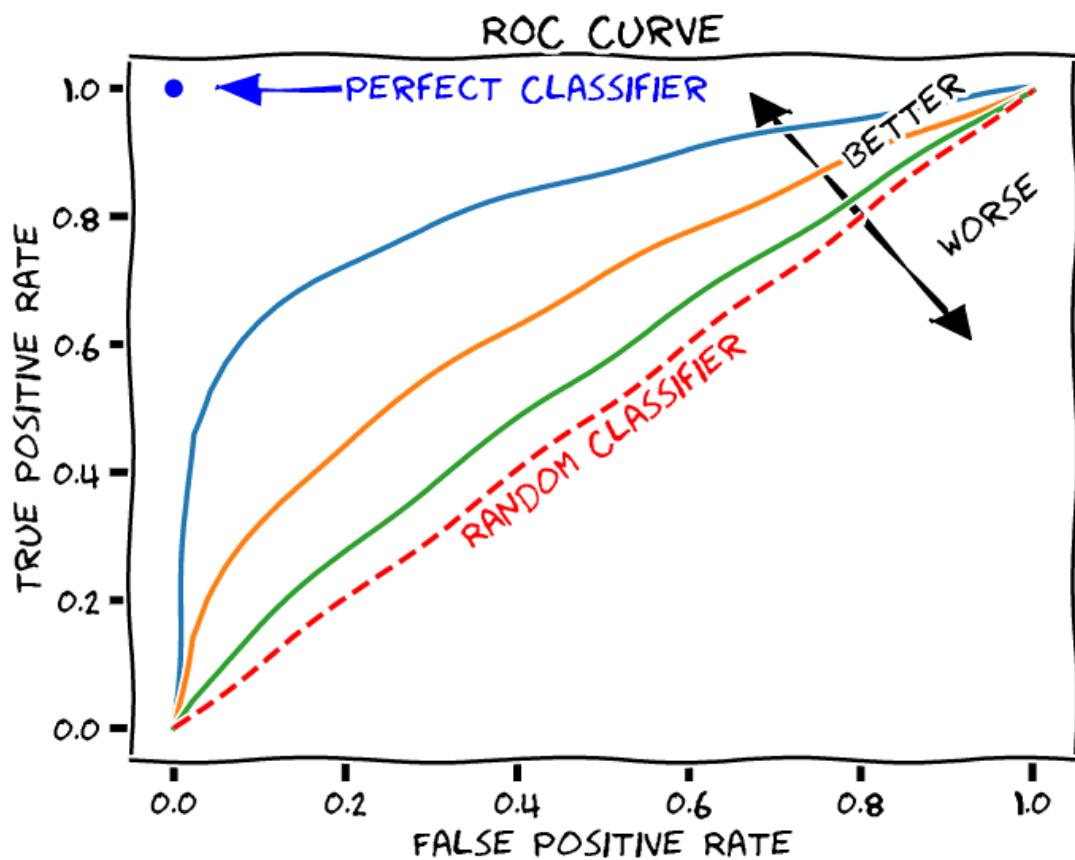
Confusion Matrix

La confusion matrix diventa tanto più utile tanto più cresce il numero di classi.

Questa ci consente di vedere immediatamente le classi in cui sbagliamo maggiormente a predire e per cui si *confondono* maggiormente.



ROC - Receiver Operating Curves



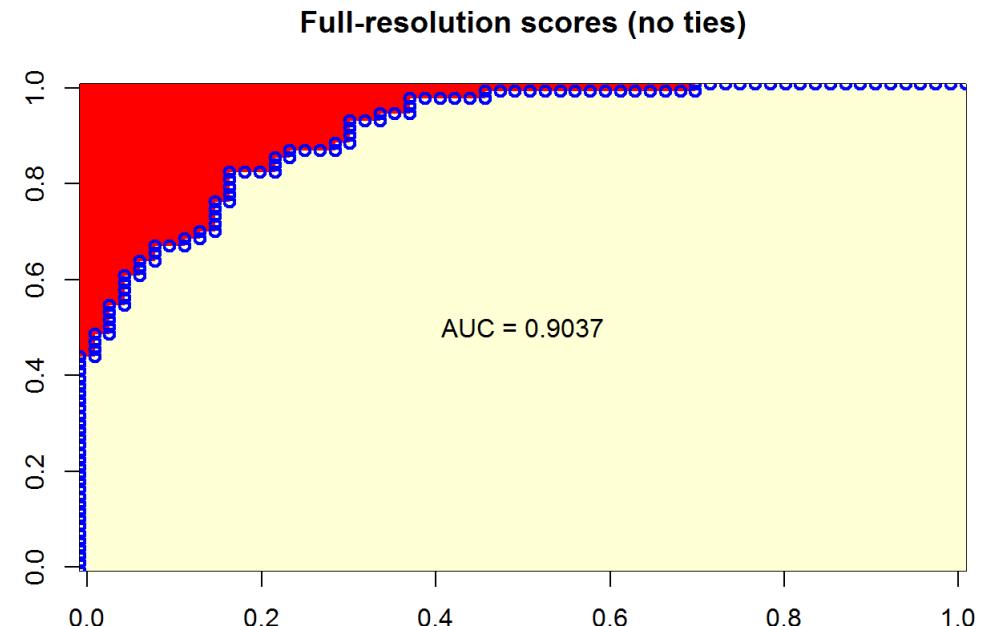
La ROC curve ci consente di

- Confrontare diversi classificatori
- Stimare il tasso di falsi positivi / negativi in funzione delle soglie che impostiamo per classificare

AUC – Area Under Curve

Come dice il nome l'AUC è una metrica per confrontare i diversi modelli riassumendo la ROC in un unico numero.

Maggiore l'area sotto la curva maggiore la performance del modello.



Classification Metrics

Vediamo alcune di queste metriche in un esempio

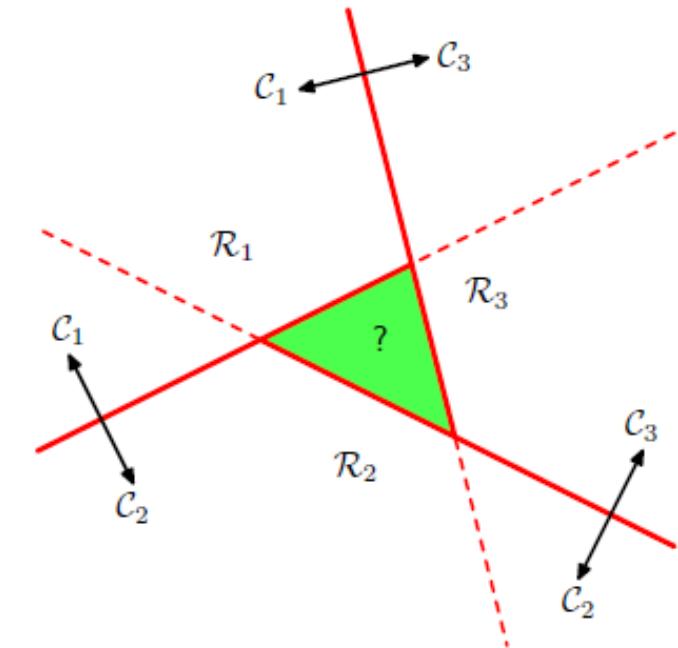
0401_classification

https://colab.research.google.com/drive/1_Rt6aEGC63LyZLGNE8NyaCph939nni1s#scrollTo=43XuLeYqU5ep

Classificazione binaria

Quasi tutti i problemi di classificazione possono essere interpretati come problemi one-vs-rest e quindi di classificazione binaria. Nel caso avessi solamente due classi mi basta un modello, nel caso avessi N classi dovrà addestrare N classificatori.

Un modello parametrico cerca l'iperpiano che consente di dividere le classi tra loro in modo che le distanze dei punti dal piano (o superficie di decisione) sia massima.



Classificazione binaria

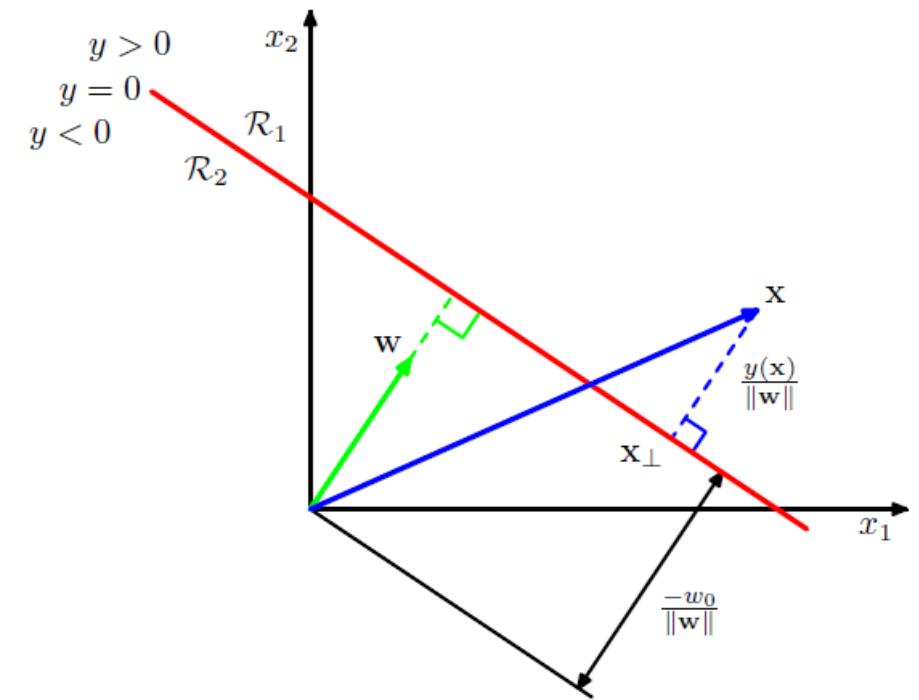
Esattamente come nei problemi di regressione, per i modelli lineari di classificazione, andremo a cercare una formula

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

che descriva la superficie di decisione.

Assegneremo alla classe A nel caso in cui $y(x) \geq 0$,
a B altrimenti.

Il piano di decisione sarà quindi descritto da $y(x) = 0$



Perceptron

Nel 1962 Rosenblatt definisce il **perceptron**.

Il perceptron è un modello di classificazione binario in cui le x vengono prima passate per una funzione di trasformazione non lineare (funzione di base) e poi costruire un modello lineare.

$$y(x) = f(w^T \phi(x))$$

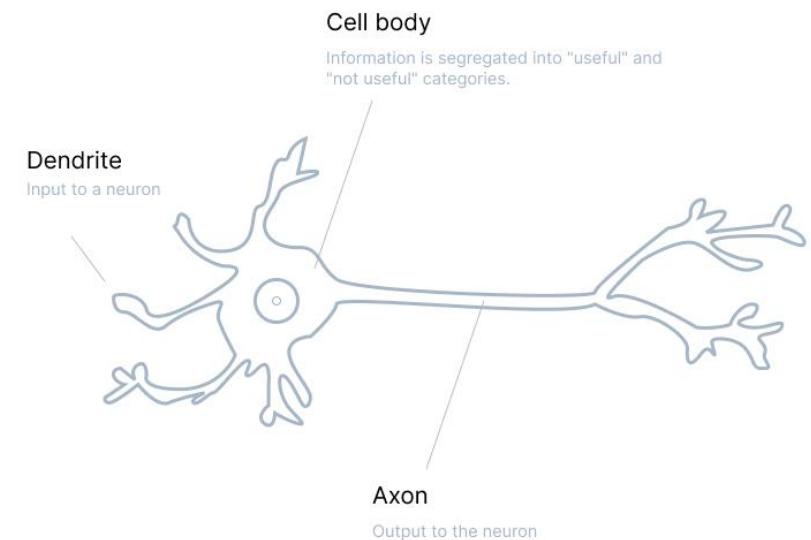
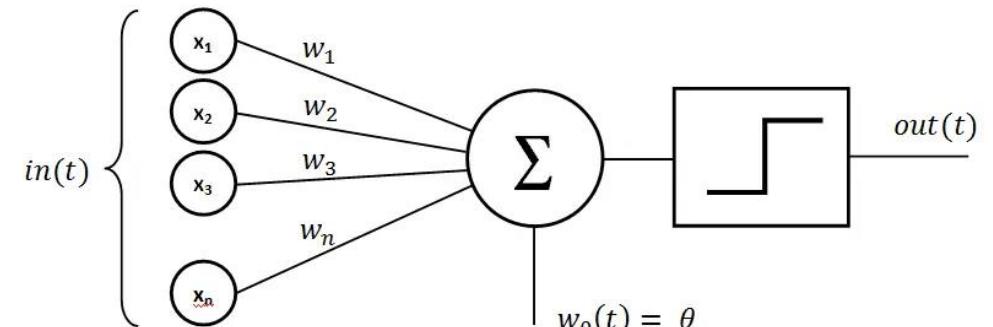
La **funzione di attivazione f** è

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

Neurone

Il perceptron di Rosenblatt è considerato il primo neurone artificiale.

Ricorda la struttura dei neuroni biologici in cui vengono presi degli input da una serie di ingressi, sommati nella cellula e passati avanti se si raggiunge un livello soglia.



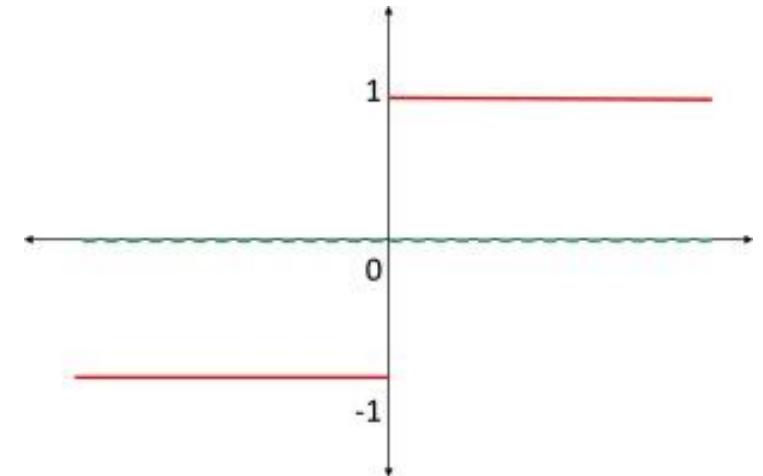
Funzione di attivazione

La funzione di attivazione è la funzione che pesa i valori in ingresso per generare un risultato, spesso non lineare, in uscita.

L'attivazione del perceptron non è nient'altro che la verifica del segno.

Dato un numero ritorna 1 se positivo, -1 se negativo

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



Perceptron Criterion

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

Idealemente vorremmo minimizzare il numero di esempi classificati erroneamente.

Il problema del perceptron è che il suo gradiente è spesso nullo e difficile da ottimizzare.

Si può riscrivere la quantità di errore in questo modo

$$E_P(\mathbf{w}) = - \sum_{n \in M} \mathbf{w}^T \phi_n t_n$$

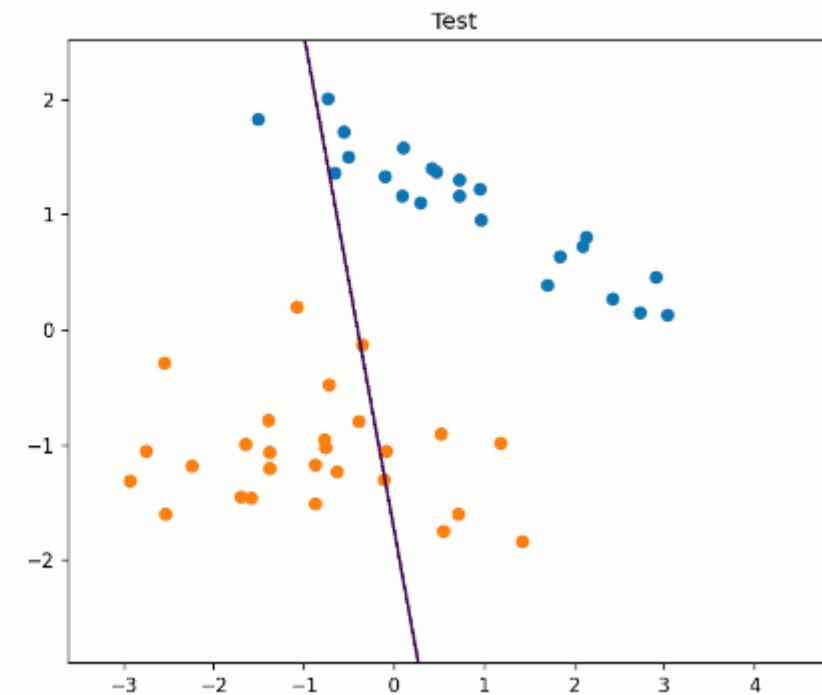
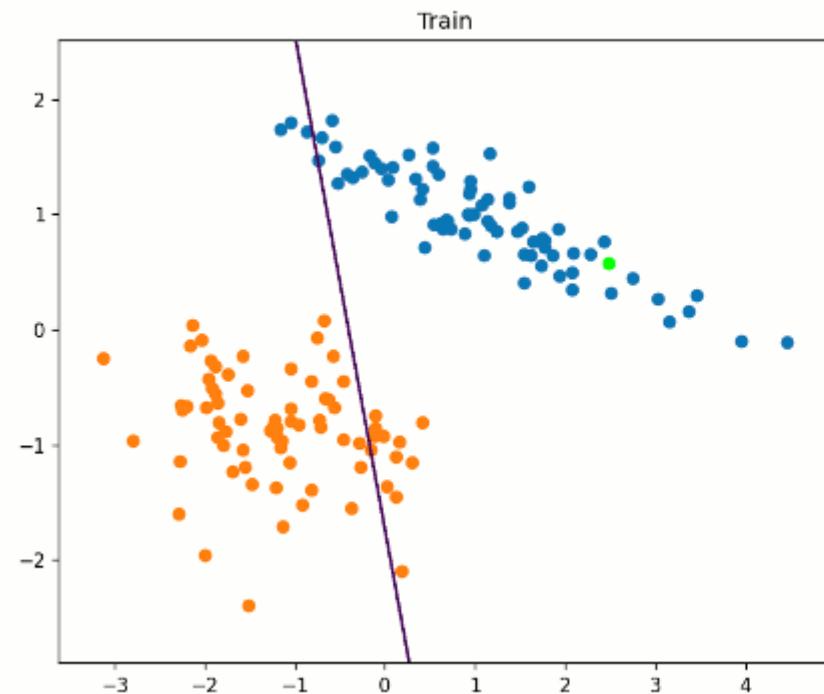
così da cercare un vettore \mathbf{w} in modo che gli esempi \mathbf{x} appartenenti ad A abbiano $\mathbf{w}^T \phi(x)$ maggiore di zero e gli esempi \mathbf{x} appartenenti a B minori di zero. Moltiplicandoli per la propria label risulta sempre positiva la quantità nella sommatoria per gli esempi corretti, e negativa per gli esempi errati.

Selezionando solamente quelli errati ($n \in M$) abbiamo un valore continuo da ottimizzare.

Perceptron

$$E_P(\mathbf{w}) = - \sum_{n \in \mathcal{M}} \mathbf{w}^T \phi_n t_n$$

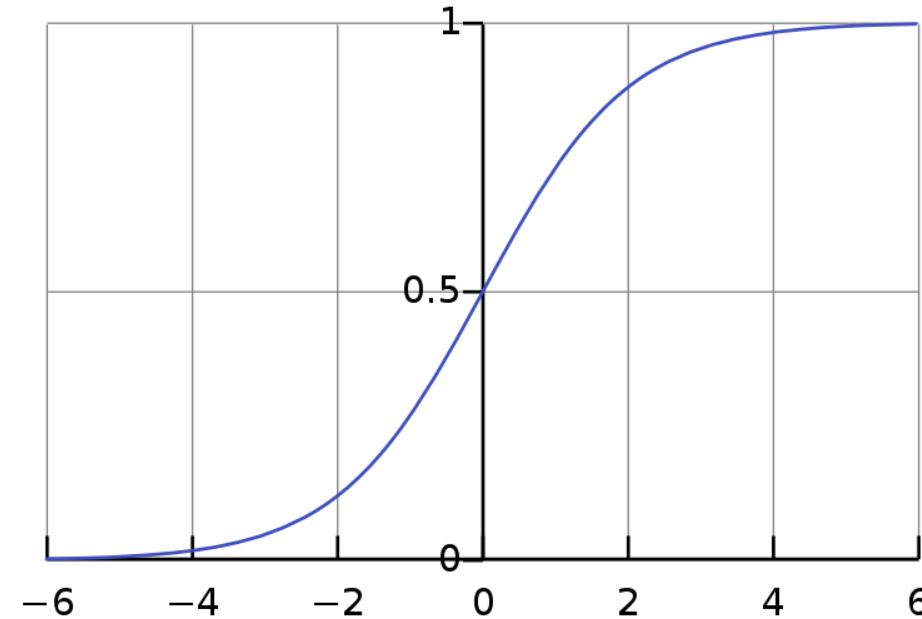
Iteration: 1/2; Point: 1/150



Logistic Regression

La **regressione logistica** (chiamata anche logit o logreg) è un'evoluzione del perceptron andando a cambiare la funzione di attivazione da una funzione pesantemente discontinua (il segno) ad una continua chiamata **sigmoide** e spesso rappresentata con $\sigma(\cdot)$

$$f(x) = \frac{1}{1 + e^{-x}}$$



Logistic Regression

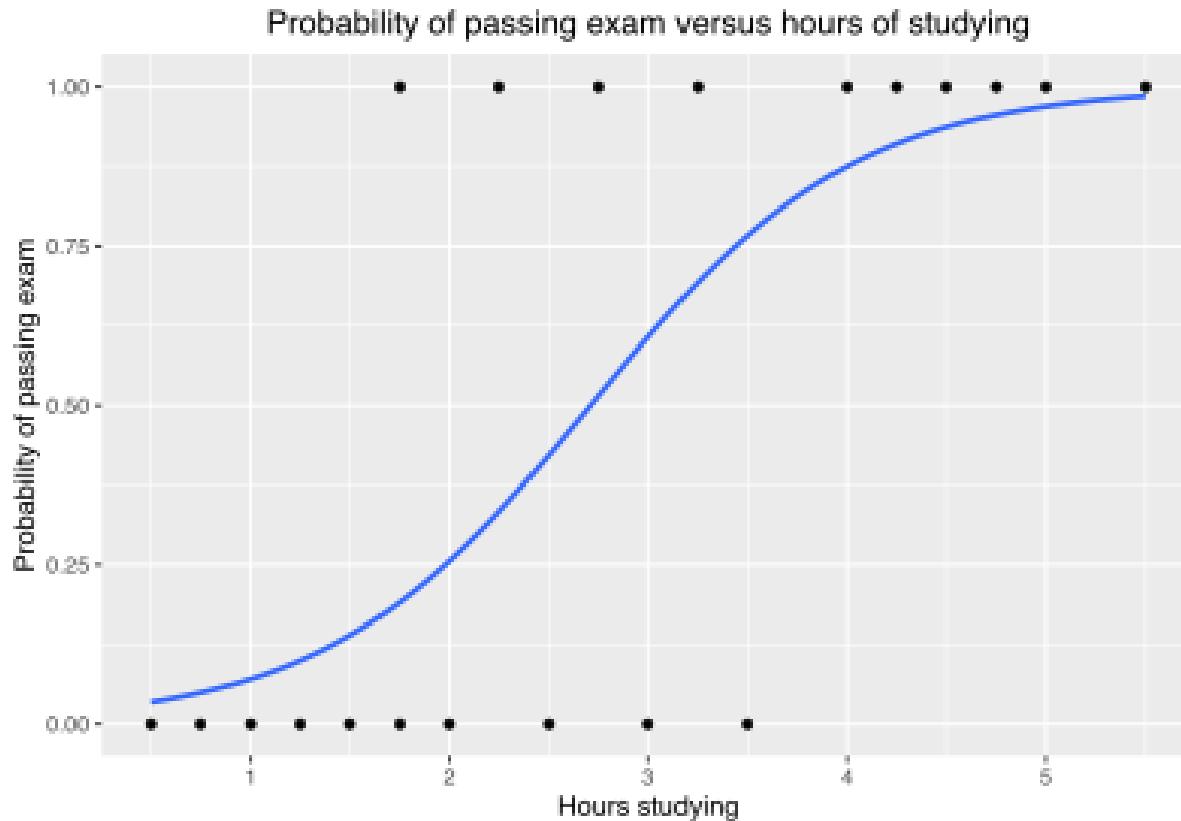
La regressione logistica ha tante qualità, la principale è che varia in modo continuo tra zero ed uno.

Per questo è molto comoda da usare per rappresentare la probabilità di appartenere ad una classe.

- Probabilità di un motore guasto
- Probabilità che il paziente sia malato
- Probabilità che una mail sia spam

Una volta stimata la probabilità di appartenere alla classe A, l'effettiva predizione avviene semplicemente impostando un valore di soglia (es. 0.5).

Esempio monovariato



$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

$$p(x) = \frac{1}{1 + e^{-(x - \mu)/s}}$$

Log loss

La logistic regression introduce una nuova funzione di costo, chiamata **Log Loss** o **Cross-Entropy**.

Possiamo definire che per il punto k -esimo, con p_k la probabilità predetta, la log loss sarà

$$\begin{cases} -\ln p_k & \text{if } y_k = 1, \\ -\ln(1 - p_k) & \text{if } y_k = 0. \end{cases}$$

Che equivale alle likelihood della predizione conoscendo la label.

Nel caso in cui p_k tendesse a zero con $y_k = 1$ il logaritmo crescerebbe molto. Lo stesso vale per il caso opposto con p_k tendente ad 1. Il costo sarà invece quasi nullo se la probabilità segue la label.

Log loss

Combinando in un'unica equazione si ottiene

$$-y_k \ln p_k - (1 - y_k) \ln(1 - p_k).$$

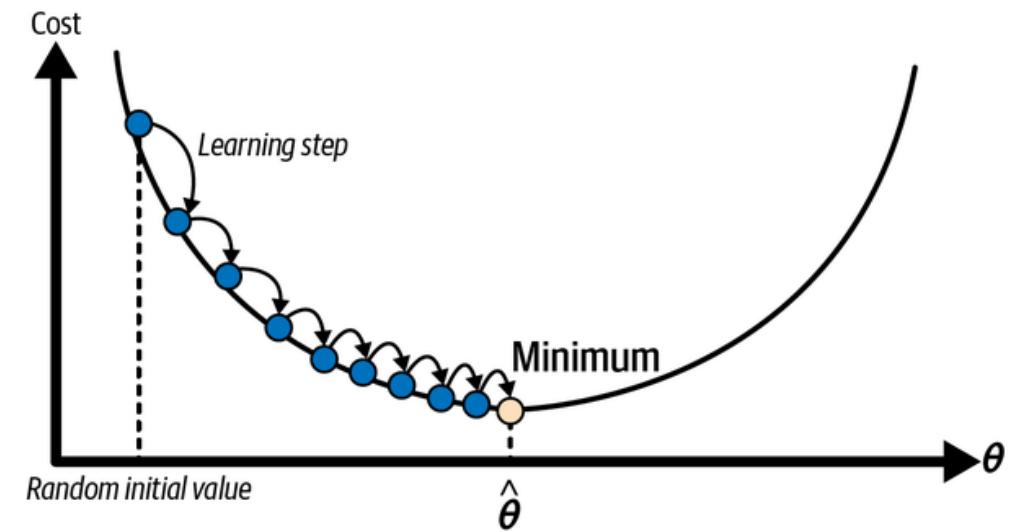
che, sommando su tutti i k esempi del dataset, rappresenta la nostra funzione di costo da minimizzare.

La log loss non ha una soluzione in forma chiusa come l'MSE nei problemi di regressione lineare, ma applicata alla regressione logistica è convessa e quindi si raggiunge sempre un unico ottimo globale.

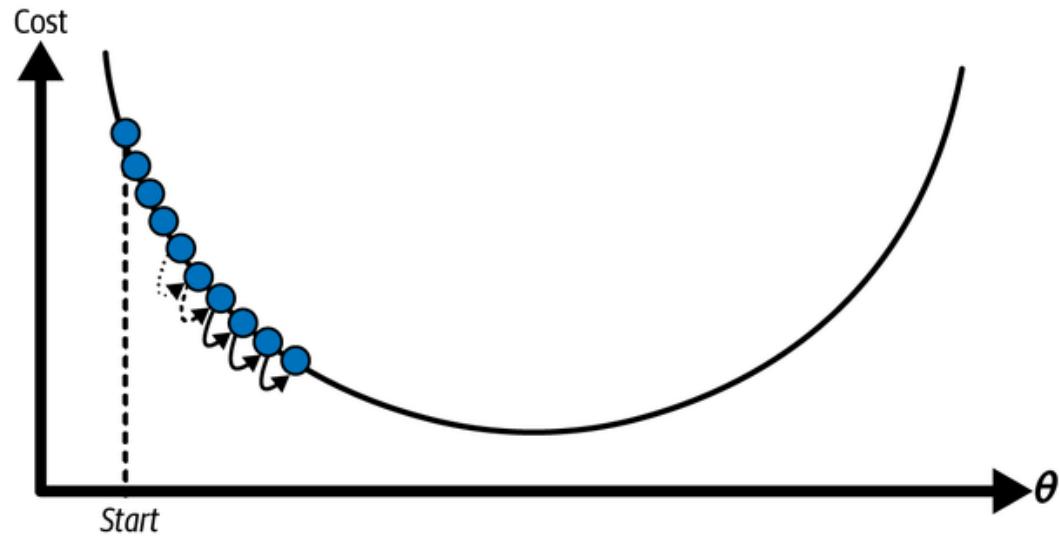
Gradient descent

Sapendo che la nostra funzione di costo è convessa
dobbiamo scendere il gradiente per trovare i parametri ottimi.

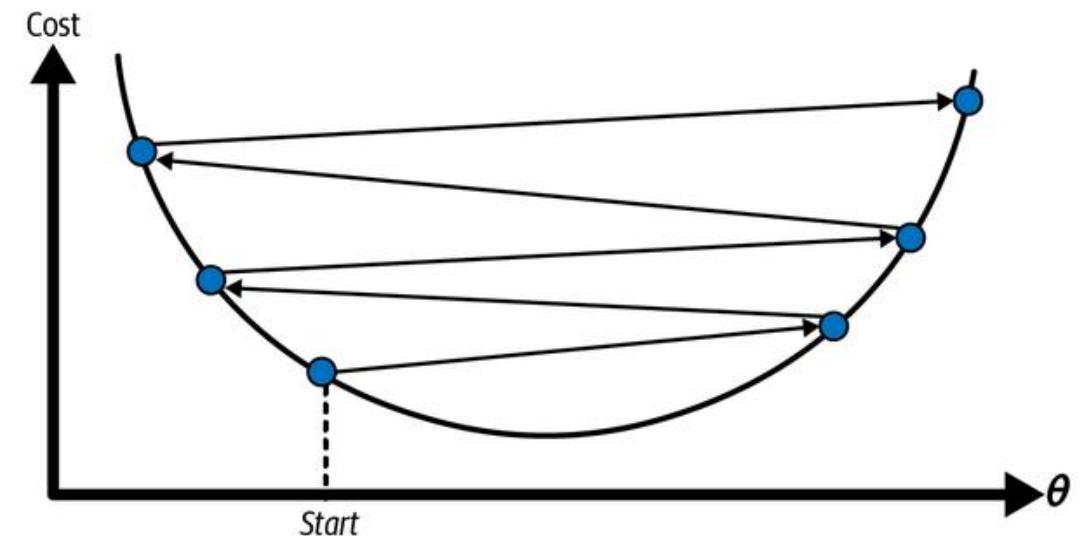
È buona norma randomizzare i parametri θ iniziali.
È altrettanto importante cercare un **learning rate** adeguato al problema



Gradient descent

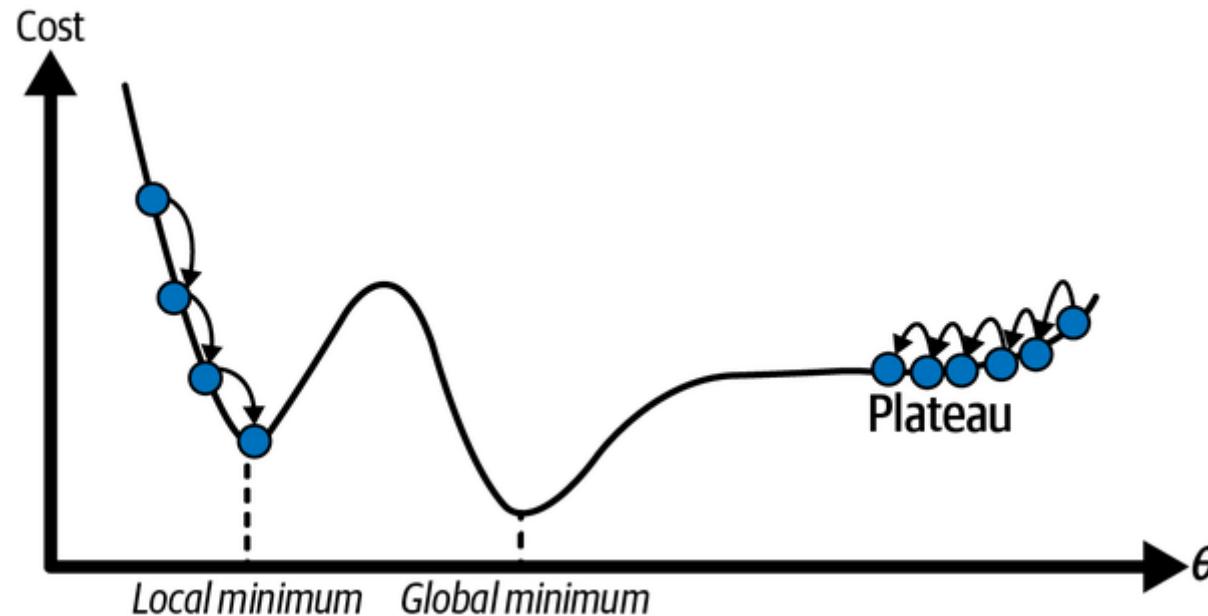


Learning rate troppo basso



Learning rate troppo alto

Gradient descent



Gradient descent

Ad ogni ciclo di ottimizzazione, per ogni esempio, andiamo a calcolare le derivate parziali rispetto ai parametri e ri-aggiorniamo i parametri moltiplicandole per il learning rate.

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

Un intero ciclo di ottimizzazione su tutti i dati è chiamato **epoca**.

Possiamo velocizzare l'apprendimento applicando l'aggiornamento dei parametri non ad ogni esempio ma per un **batch** di dati insieme.

Lo *stochastic gradient descent* aggiorna i parametri per ogni punto, o per un set di punti campionario, mentre il classico avviene mediando tutte le derivate parziali. Il primo ha diversi vantaggi, tra cui l'abilità ad uscire da minimi locali campionando sempre in modo diverso il dataset per l'ottimizzazione.

Gradient descent

Una volta impostate le equazioni base della Regressione Logistica e definita la relativa funzione di costo (per semplicità di notazione, si considera il caso con una sola osservazione, $m=1$)

$$\begin{aligned} z &= x * w + b \\ h_{\theta} &= \frac{1}{1 + e^{-z}} \\ cost_{(\theta,y)} &= \frac{1}{m} \sum_{j=1}^m cost_{j(\theta,y=0)} + cost_{j(\theta,y=1)} = \frac{1}{m} \sum_{j=1}^m -y_j * log(h_{\theta_j}) - (1 - y_j) * log(1 - h_{\theta_j})) \end{aligned}$$

Gradient descent

il solito obiettivo è la minimizzazione di questa funzione attraverso l'identificazione dei valori ottimali di w e b e per fare ciò entra in gioco nuovamente il **Gradient Descent**.

Per cui calcoliamo i valori che ci permettono di aggiornare le nostre variabili di sistema: $\frac{\partial \text{cost}}{\partial w}$ e $\frac{\partial \text{cost}}{\partial b}$

$$\frac{\partial \text{cost}_{(\theta,y=1)}}{\partial w} = \frac{-y}{h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial w} = \frac{-y}{h_\theta} * h_\theta * (1 - h_\theta) * x = -y * (1 - h_\theta) * x$$

$$\frac{\partial \text{cost}_{(\theta,y=0)}}{\partial w} = \frac{1-y}{1-h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial w} = \frac{1-y}{1-h_\theta} * h_\theta * (1 - h_\theta) * x = (1-y) * h_\theta * x$$

$$\begin{aligned}\frac{\partial \text{cost}_{(\theta,y)}}{\partial w} &= \frac{\partial \text{cost}_{(\theta,y=1)}}{\partial w} + \frac{\partial \text{cost}_{(\theta,y=0)}}{\partial w} = -y * (1 - h_\theta) * x + (1-y) * h_\theta * x \\ &= x * (-y + y * h_\theta + h_\theta - y * h_\theta) = x * (h_\theta - y)\end{aligned}$$

Gradient descent

$$\frac{\partial \text{cost}_{(\theta,y=1)}}{\partial b} = \frac{-y}{h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial b} = \frac{-y}{h_\theta} * h_\theta * (1 - h_\theta) * 1 = -y * (1 - h_\theta)$$

$$\frac{\partial \text{cost}_{(\theta,y=0)}}{\partial b} = \frac{1-y}{1-h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial b} = \frac{1-y}{1-h_\theta} * h_\theta * (1 - h_\theta) * 1 = (1-y) * h_\theta$$

$$\begin{aligned}\frac{\partial \text{cost}_{(\theta,y)}}{\partial b} &= \frac{\partial \text{cost}_{(\theta,y=1)}}{\partial b} + \frac{\partial \text{cost}_{(\theta,y=0)}}{\partial b} = -y * (1 - h_\theta) + (1-y) * h_\theta \\ &= -y + y * h_\theta + h_\theta - y * h_\theta = h_\theta - y\end{aligned}$$

Calcolati gli aggiornamenti da applicare nell'iterazione attuale del Gradient Descent, applichiamo l'aggiornamento per poi passare all'iterazione successiva:

$$w_{i+1} = w_i - \alpha * \frac{\partial \text{cost}_{(\theta,y)_i}}{\partial w} = w_i - \alpha * (h_{\theta_i} - y) * x$$

$$b_{i+1} = b_i - \alpha * \frac{\partial \text{cost}_{(\theta,y)_i}}{\partial b} = b_i - \alpha * (h_{\theta_i} - y)$$

Features scaling

In quasi tutti i problemi è molto importante assicurarsi che i valori in ingresso abbiano scale uguali o simili.

Se avessimo un parametro che varia di migliaia di unità, ed un secondo parametro che varia di centesimi, il gradiente farà molta più fatica a confrontare l'apporto di ciascun fattore, rispetto a due variabili che stanno entrambe nel range 0-1.

	PassengerId	Survived	Pclass	Age
count	183.000000	183.000000	183.000000	183.000000
mean	455.366120	0.672131	1.191257	35.674426
std	247.052476	0.470725	0.515187	15.643866
min	2.000000	0.000000	1.000000	0.920000
25%	263.500000	0.000000	1.000000	24.000000
50%	457.000000	1.000000	1.000000	36.000000
75%	676.000000	1.000000	1.000000	47.500000
max	890.000000	1.000000	3.000000	80.000000

Features scaling

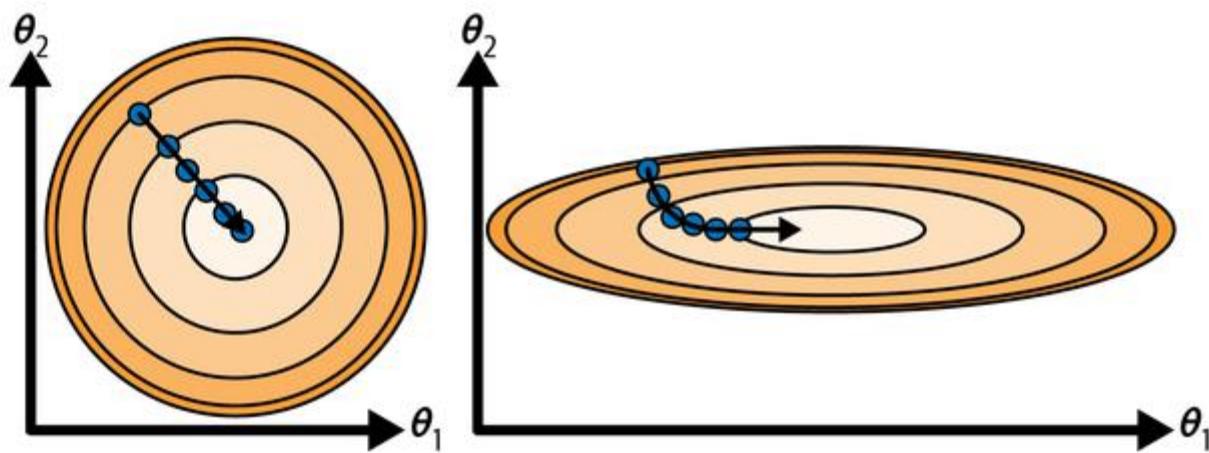


Figure 4-7. Gradient descent with (left) and without (right) feature scaling

Features scaling

Un altro grosso vantaggio del features scaling è che a fronte di features con lo stesso range, il coefficiente w_i che moltiplica $x_{i\ scaled}$ ci indica la sua «importanza».

Questo è vero solamente se tutte le features sono **scorrelate** tra loro.

Osservare i coefficienti del modello ci fornisce sempre una buona indicazione di cosa si sta rivelando essere più utile ai fini dell'apprendimento.

Multiclass

Nel caso multiclass, ovvero con più di due classi tra cui discriminare, si può rivedere il caso singolo come likelihood della classe dato il vettore

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

Chiamato anche «normalized exponential» o «**softmax**» in quanto ripesa le uscite come probabilità in relazione agli altri valori di output.

Classification 101

Prendiamo ora un esempio e applichiamo un po' dei concetti visti

https://colab.research.google.com/drive/1_Rt6aEGC63LyZLGNE8NyaCph939nni1s#scrollTo=BD8nqalsRmqi



05 – Regression & Classification methods

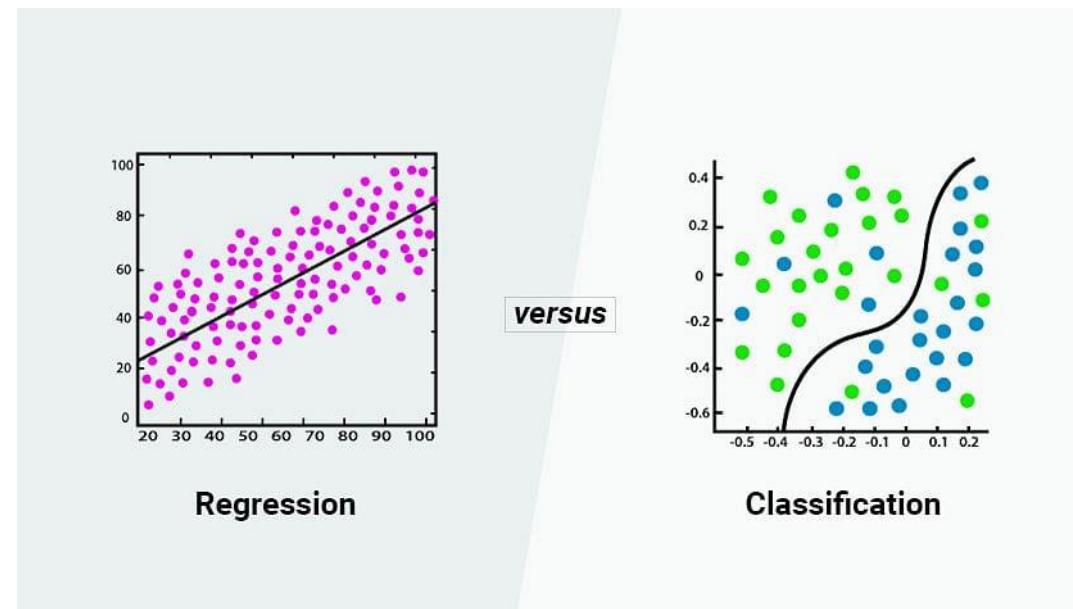
Daniele Gamba

2022/2023

Classificazione

I problemi di **classificazione** sono problemi in cui dobbiamo determinare a che classe appartiene un esempio.

Determinare la razza di un gatto o se un paziente è sano o malato sono problemi di classificazione.



Classificazione

Binary Classification



Multiclass Classification



Multilabel Classification



Confusion Matrix

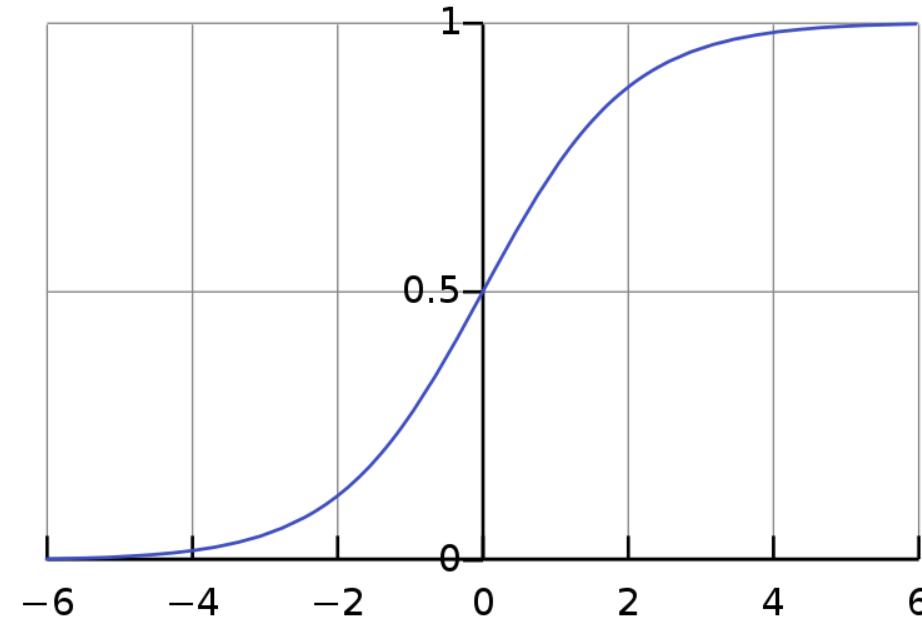
Allo stesso modo per i problemi di classificazione abbiamo le misure di performance.

		True condition			
		Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive , Power	False positive , Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative , Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR}^+}{\text{LR}^-}$
		False negative rate (FNR), Miss rate = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

Logistic Regression

La **regressione logistica** (chiamata anche logit o logreg) è un'evoluzione del perceptron andando a cambiare la funzione di attivazione da una funzione pesantemente discontinua (il segno) ad una continua chiamata **sigmoide** e spesso rappresentata con $\sigma(\cdot)$

$$f(x) = \frac{1}{1 + e^{-x}}$$



Logistic Regression

La regressione logistica ha tante qualità, la principale è che varia in modo continuo tra zero ed uno.

Per questo è molto comoda da usare per rappresentare la probabilità di appartenere ad una classe.

- Probabilità di un motore guasto
- Probabilità che il paziente sia malato
- Probabilità che una mail sia spam

Una volta stimata la probabilità di appartenere alla classe A, l'effettiva predizione avviene semplicemente impostando un valore di soglia (es. 0.5).

1-vs-all

Abbiamo accennato al fatto che a fronte di un problema di multiclass classification possiamo risolvere il problema addestrando N classificatori 1-vs-all.

Vediamolo insieme (e come sklearn lo fa in automatico per noi)

https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY_xR36oYsFOLp7cpsQf

Valori di soglia

Come abbiamo la lezione precedente, impostando diversi valori di soglia otteniamo classificatori che tendono ad avere più o meno falsi positivi rispetto ai falsi negativi.

A seconda del problema che vogliamo risolvere useremo valori di soglia che spostino le predizioni errate nella direzione in cui abbiamo più tolleranza.

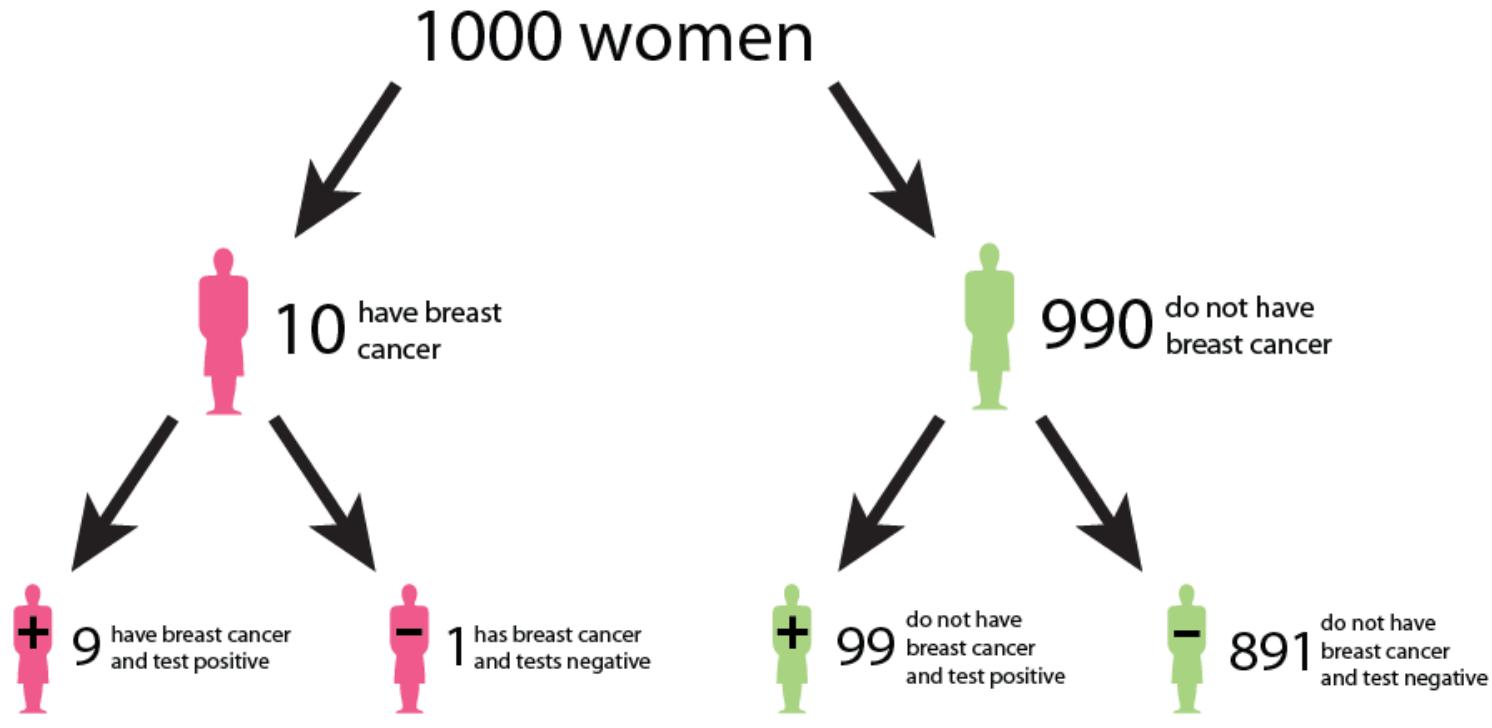
Test diagnostico

Vogliamo che i falsi negativi (pazienti malati segnalati come sani) siano virtualmente nulli, per cui imposteremo valori di soglia molto bassi (es. 0.1 con 1 paziente sicuramente malato). Questo avrà come controindicazione che molti pazienti avranno dei falsi positivi e si dovranno sottoporre ad altri esami.

Acquisto azioni in borsa

Vogliamo un modello che ci dica se è il momento giusto per acquistare un titolo. In questo caso associando ad 1 il momento ottimo per acquistare in borsa, verosimilmente vorremo comprare solamente i titoli di cui il modello è molto sicuro che siano buoni impostando un valore di soglia alto (es. 0.9)

Valori di soglia



Cancro al seno per donne sopra i 50 anni

Se diagnosticate come positive (108) c'è solo 1 possibilità su 10 di averlo.

Il modello ha il 90% di accuratezza

Dataset sbilanciato

L'esempio del cancro è una condizione reale in cui si misura la performance del modello a fronte di un dataset sbilanciato (i pazienti malati sono l'1%).

Riconoscere fin da subito i problemi in cui il dataset è molto sbilanciato ci consente di tenerne conto. È importante tener focalizzato il problema che stiamo cercando di risolvere e la direzione in cui il falso positivo o il falso negativo può impattare sul problema.

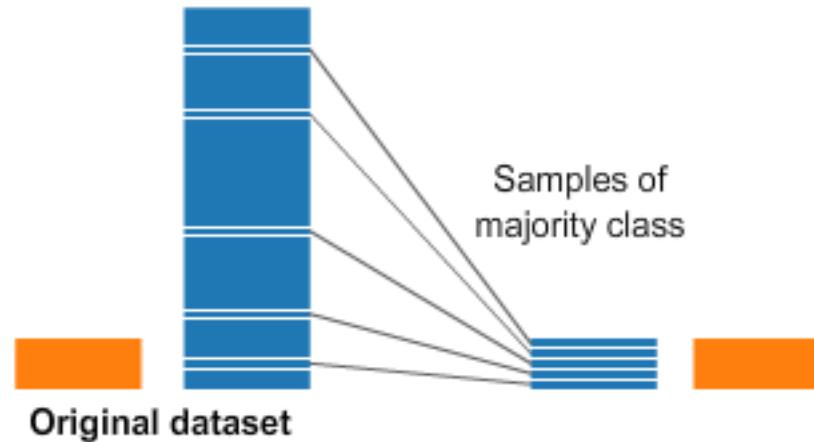
Dataset sbilanciato

Per gestire dataset sbilanciati possiamo usare diverse tecniche

- **Ricampionare il dataset**
 - Sottocampionando la classe più numerosa
 - Sovracampionando la classe più scarsa
- Usare **metriche corrette in validazione** per scegliere modelli più bilanciati (f1 o AUC invece che accuratezza)
- **Pesando** diversamente le classi durante la fase di training

Resampling

Undersampling



Oversampling



Class weight

Nel riapplicare il gradiente ai nostri pesi per correggerne il valore

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

possiamo pesare il ∇E_n in modo che pesi maggiormente nelle classi meno frequenti del dataset moltiplicandolo per un fattore.

In questo modo compensiamo il fatto che il modello ottimizzi un numero N maggiore di volte i parametri sulla classe più rappresentata rispetto a quella sottocampionata.

Dataset sbilanciato

Vediamo ora un esempio di come gestire un dataset molto sbilanciato

https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY_xR36oYsFOLp7cpsQf

<https://replicate.com/lambdal/text-to-pokemon>



Classification algorithms

Abbiamo visto la regressione logistica come algoritmo per classificare tra due (o più) classi.

Di algoritmi ne esistono svariati, iniziamo ora a vederne una carrellata dei metodi più usati e dei loro vantaggi.

Naive Bayes

Il Naive Bayes è un algoritmo molto semplice di classificazione binaria.

Assume che ogni variabile indipendentemente dalle altre porti una componente di probabilità che l'esempio appartenga alla classe positiva basandosi sulla regola di Bayes della probabilità condizionata.

$$P(y | x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n | y)}{P(x_1, \dots, x_n)}$$
$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

Con l'assunzione di indipendenza condizionata della seconda equazione possiamo calcolare

$$P(y | x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i | y)$$

↓

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y),$$

https://scikit-learn.org/stable/modules/naive_bayes.html

Naive Bayes

Ne esiste anche la versione con distribuzioni Gaussiane

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

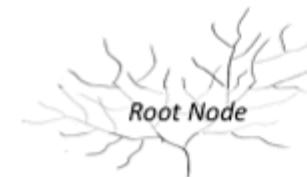
L'assunzione di indipendenza tra le variabili è quello che rende naive questo algoritmo perché

- Le variabili spesso non sono indipendenti tra loro
- Le variabili non hanno tutte lo stesso impatto sul risultato del predittore

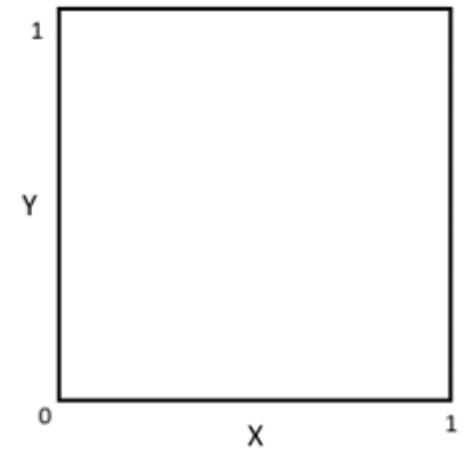
Pur essendo un classificatore molto semplice e che non viene più usato si è dimostrato molto utile in svariati casi, tra cui gli spam-filter.

Decision tree

L'albero decisionale è uno degli algoritmi più semplici e più interpretabili di classificazione e regressione.



Iterativamente viene preso una dimensione del problema e scelto un valore di soglia in grado di discriminare tra due gruppi. Dalla radice in questo modo creiamo un albero decisionale che ad ogni passaggio discrimina meglio tra i punti.

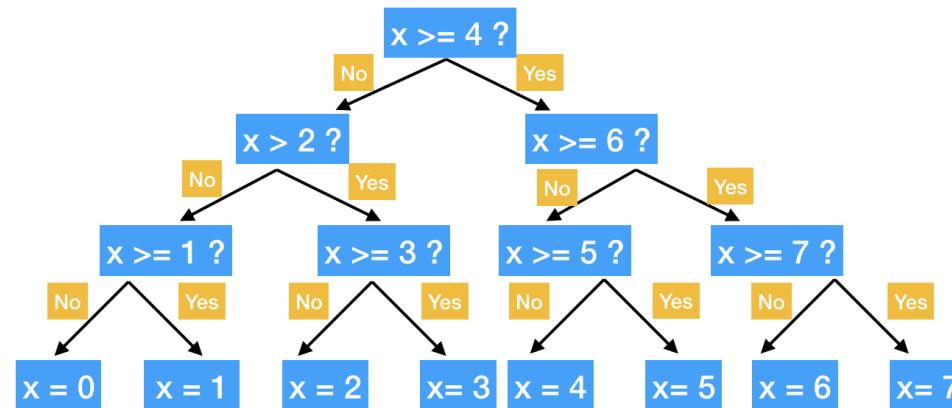
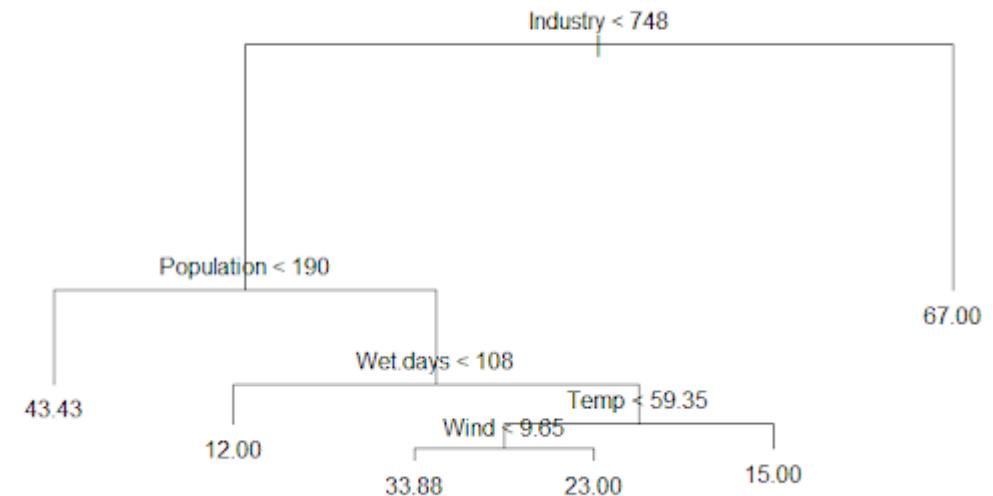


For more tutorials: annalyzin.wordpress.com

Regression tree

Nei problemi di regressione usiamo la stessa tecnica per discriminare tra gruppi di dati con valori simili.

Il valore predetto finale sarà la media degli esempi che rimangono nel gruppo «foglia» del nostro albero.



Decision tree

Il primo nodo su cui si va a partizionare il dataset è il nodo radice.

Gli ultimi a cui si arriva sono i nodi fogli o terminali, in mezzo ci sono i rami.

Come nel caso della regressione logistica i pesi ci indicavano l'importanza delle features (riscalate) nel prendere la decisione, negli alberi il primo nodo è quello con il maggior impatto, e così via i successivi.

È molto più facile visualizzare il processo decisionale di un albero di altre tecniche, ma è anche limitato nella sua espressività

- Per i problemi di regressione ad un solo valore
- In generale non raggiunge livelli di performance molto alti

Decision tree

Per apprendere l'albero decisionale divide lo spazio dei regressori (lo spazio dei valori possibili per le features) in M regioni distinte.

Per ogni regione la predizione sarà la media dei valori presenti all'interno.

Per ogni regione si va a calcolare il valore soglia migliore e un grado di «impurità del nodo» che consente di scegliere i nodi più determinanti a discriminare i gruppi.

Decision tree

Oltre alla fase di apprendimento bisogna capire quando fermarsi a dividere ulteriormente lo spazio.

Questo è fatto con due approcci

- Mettere un valore di soglia di errore / impurità tollerato all'interno del nodo foglia
- Costruire tutto l'albero fino in fondo (ovvero 1 nodo per ogni valore nel caso non si possa fare altrimenti) e poi intervendere con delle tecniche di «**pruning**» per tagliare i nodi foglia con una complessità troppo alta bilanciando tramite un valore di regolarizzazione

$$y_\tau = \frac{1}{N_\tau} \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} t_n$$

Predizione ottima per la regione tau

$$Q_\tau(T) = \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} \{t_n - y_\tau\}^2.$$

Contributo all'errore quadratico dei residui (regressione)

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda |T|$$

Criterio di pruning

Gini index

L'indice di Gini è un indice che, nei problemi di classificazione, indica una impurità del nodo data la regione k.

Possiamo usarlo sia per settare il valore soglia a cui fermarci, sia come criterio per fare pruning nei problemi di classificazione

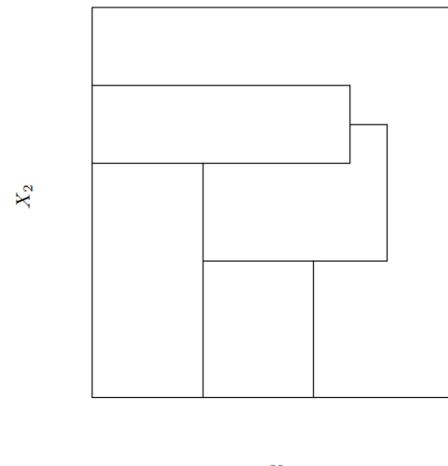
$$Q_\tau(T) = \sum_{k=1}^K p_{\tau k} (1 - p_{\tau k}).$$

con p proporzione dei dati appartenenti alla classe k del nodo.

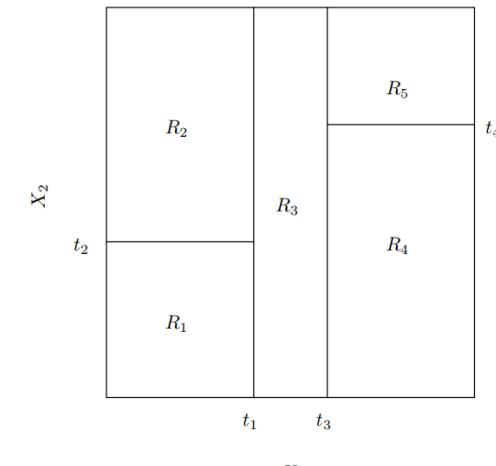
Limiti

Uno dei limiti di usare un albero decisionale è che non possiamo facilmente rappresentare aree di separazione complesse.

L'albero decisionale binario è in grado di dividere solo aree per iterazioni e non per gruppi.



Area non divisibile



Area divisibile

Superare i limiti

Gli alberi sono limitati nel loro potere espressivo ma si sono dimostrati estremamente efficaci nel risolvere problemi più semplici.

Uno dei modi di migliorare la nostra performance è quello di non usarne uno solo modello ma di usarne N e mediarne le predizioni. In questo modo medieremo il valore in uscita aspettandoci di migliorare le performance nei casi più complessi.

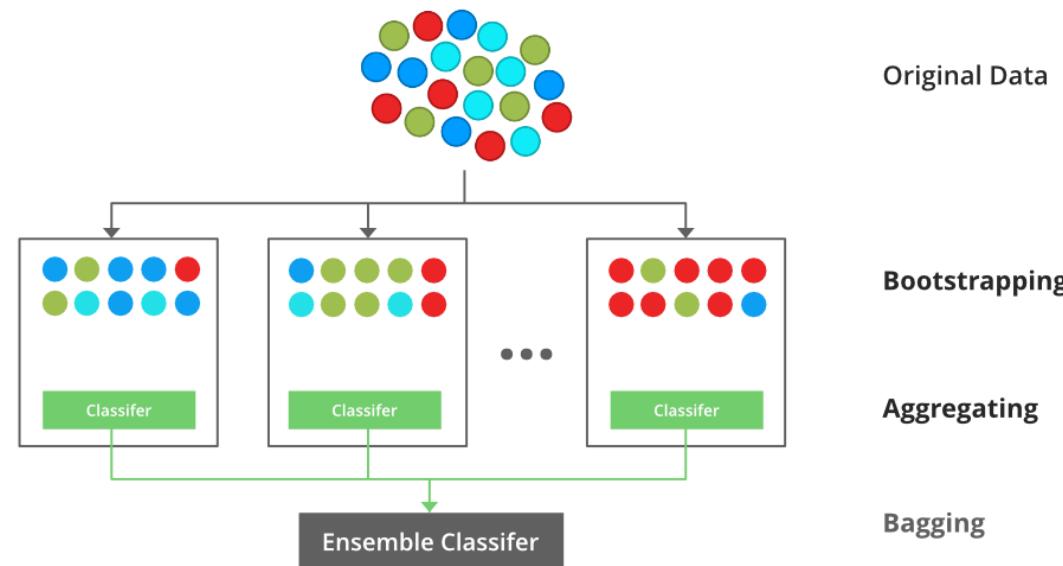
Usando N predittori, in uscita avremo

- Nei problemi di **regressione**: la **media** dei valori predetti
- Nei problemi di **classificazione**: il **majority voting** della classe

Bagging

Il **bagging**, o bootstrapping, è una tecnica per cui andiamo ad addestrare N modelli su N campionamenti diversi del nostro dataset.

In questo modo tutti i modelli avranno predizioni diverse, le medieremo alla fine per ottenere un unico valore.



Random forest

La random forest è una tecnica in cui andiamo ad allenare N decision tree e farne bagging.
N alberi fanno una foresta.

Il random forest prova a decorrelare anche gli alberi tra di loro.

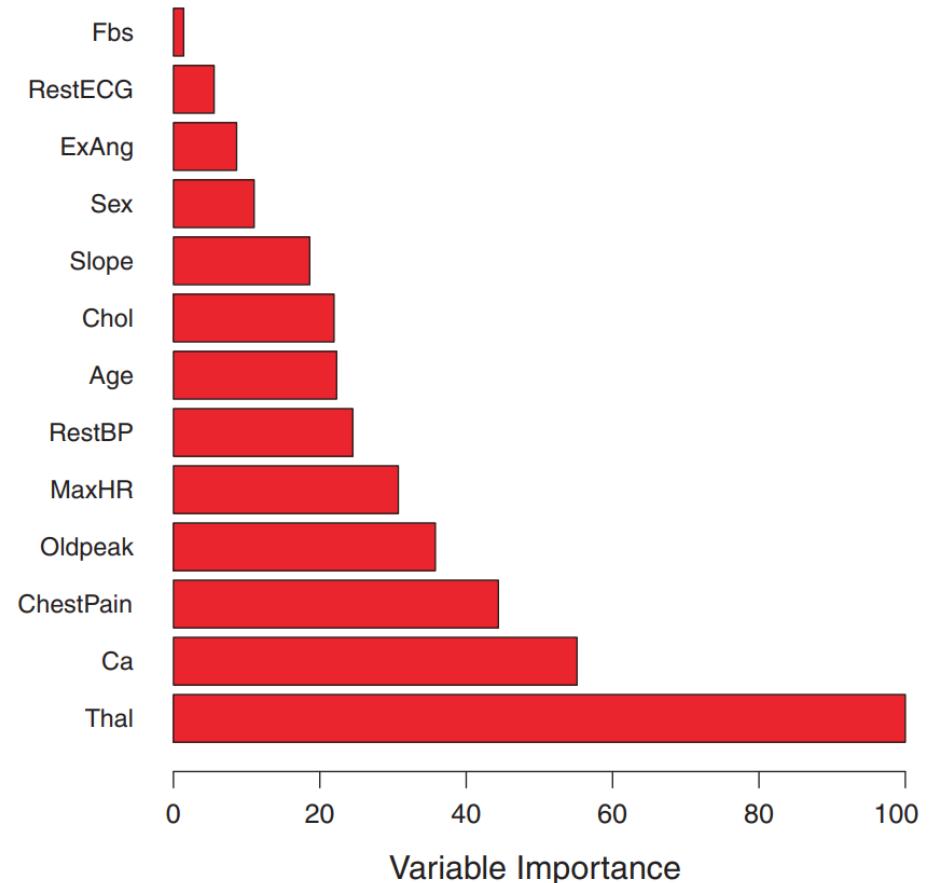
Ad ogni split seleziona a random solo q regressori tra i d totali e limita lo split a quei q regressori.

In questo modo il singolo albero è costretto a prendere percorsi per apprendere diversi dagli altri.

Random forest

Come nel caso precedente possiamo andare a vedere l'importanza di ogni feature andando a misurare quanta varianza (o RSS – Sum of Squared Residuals – $Q(t)$) nella slide 20) spiega ogni nodo che divide quel regressore.

Scommendo la quantità di errore spiegata da ciascun regressore possiamo farne una classifica di importanza.



Ensemble methods

Il concetto di unire modelli diversi per ottenere predizioni più accurate ricade sotto il cappello degli «ensemble methods» o ensemble learning.

Ne esistono di diversi tipi, tra cui

- **Bagging**, che abbiamo visto precedentemente
- **Boosting**, in cui i modelli addestrati pesano diversamente rispetto alla propria performance. Tanto più il classificatore precedente sbaglia su un set di dati, tanto più vengono pesati per quello successivo
- **Stacking**, in cui le predizioni di N modelli entrano in altri modelli per un raffinamento

Molte volte si fa ensemble con molti «weak learner», rispetto ad avere pochi modelli complessi.

Random forest

Riprendiamo l'esempio dell'iris dataset e proviamo a far classificazione con il random forest
Vedremo come le performance rispetto al singolo albero vengono migliorate incredibilmente.

https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY_xR36oYsFOLp7cpsQf

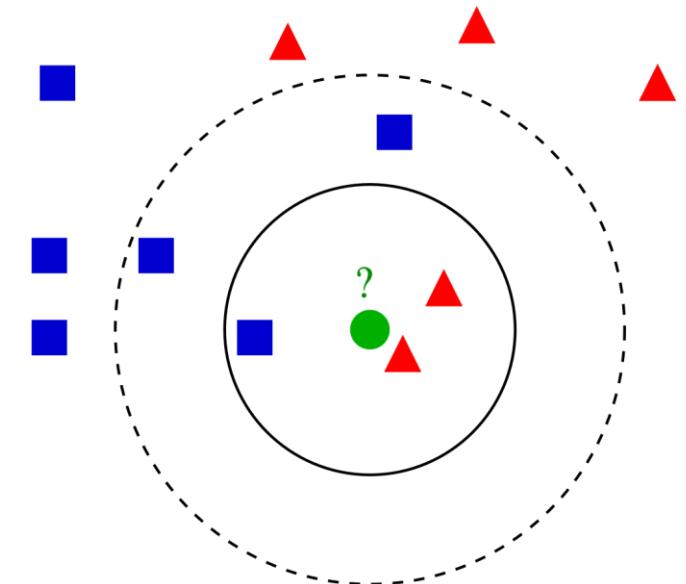
k-Nearest Neighbor

Il k-NN è un semplice algoritmo di classificazione e regressione **non-parametrico**.

Nel dover decidere a che classe appartiene un esempio possiamo confrontare il nostro vettore x con i vettori di altri k esempi più vicini secondo una *metrica*.

La decisione viene presa per majority voting nei casi di classificazione, come media nei casi di regressione.

Vengono scelti k dispari, $k=1$ indica di prendere il più vicino.



k-Nearest Neighbor

Tutti gli esempi nel dataset devono essere confrontabili tra loro secondo una metrica chiamata **kernel**.

$$K(x_1, x_2) = \langle f(x_1), f(x_2) \rangle$$

La funzione kernel applica indipendentemente a x_1 e x_2 una trasformazione in uno spazio m -dimensionale, normalmente maggiore della dimensione del vettore.

Alla fine viene eseguito il prodotto scalare delle due componenti risultando in uno scalare. In questo modo siamo in grado di confrontare esempi diversi del dataset e darne un ordinamento.

k-Nearest Neighbor

Esistono numerose funzioni kernel a seconda del confronto che si vuole eseguire tra i dati

Alcune che vengono spesso usate

- Lineare
- Polinomiale
- Gaussiana o RBF (Radial Basis Function)
- Sigmoidale
- Cosine distance
- ...

k-Nearest Neighbor

Il principale svantaggio del k-NN è la sua scarsa scalabilità.

Non avendo un modello parametrico del nostro problema, ad ogni nuovo esempio dobbiamo rieseguire tanti confronti quanti sono i nostri dati etichettati.

Questo è tanto più oneroso tanto più cresce il dataset

Non c'è fase di training, ha costo 0

Ogni inferenza è lineare o poco più con la lunghezza del dataset

Recap

Abbiamo visto

- 1-vs-all per la regressione logistica e come si estende con un modello a singolo output
- Come impatta il nostro valore di soglia tra falsi positivi e falsi negativi
- Come trattare dataset sbilanciati
- Iniziato a vedere una carrellata di algoritmi più complessi per regressione e classificazione
 - Naive Bayes
 - Decision tree
 - Random forest & ensamble methods
 - k-NN

La prossima lezione proseguiamo con metodi parametrici e non per concludere una carrellata di tecniche.



06 – Regression & Classification methods 2

Daniele Gamba

2022/2023

Recap

Abbiamo visto

- 1-vs-all per la regressione logistica e come si estende con un modello a singolo output
- Come impatta il nostro valore di soglia tra falsi positivi e falsi negativi
- Come trattare dataset sbilanciati
- Iniziato a vedere una carrellata di algoritmi più complessi per regressione e classificazione
 - Naive Bayes
 - Decision tree
 - Random forest & **ensamble methods**
 - **k-NN**

Ensemble methods

Il concetto di unire modelli diversi per ottenere predizioni più accurate ricade sotto il cappello degli «ensemble methods» o ensemble learning.

Ne esistono di diversi tipi, tra cui

- **Bagging**, che abbiamo visto precedentemente
- **Boosting**, in cui i modelli addestrati pesano diversamente rispetto alla propria performance. Tanto più il classificatore precedente sbaglia su un set di dati, tanto più vengono pesati per quello successivo
- **Stacking**, in cui le predizioni di N modelli entrano in altri modelli per un raffinamento

Molte volte si fa ensemble con molti «weak learner», rispetto ad avere pochi modelli complessi.

AdaBoost

AdaBoosting è l'implementazione del classico algoritmo di Boosting direttamente in sklearn.

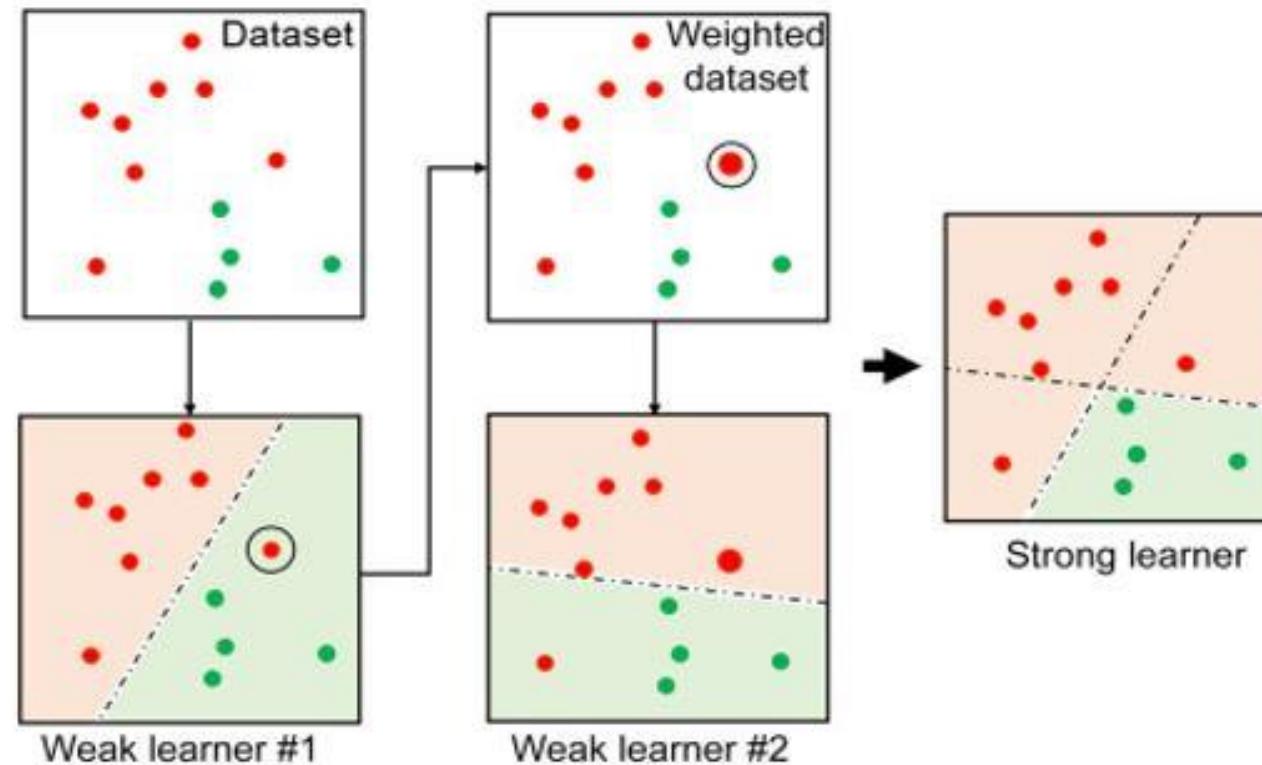
Riceve in ingresso il regressore (estimator) scelto, di default un DecisionTree, e applica il boosting andando a fittare N stimatori ripesando i dataset ogni volta in funzione degli esempi su cui si è sbagliato maggiormente.

`sklearn.ensemble.AdaBoostClassifier`

```
class sklearn.ensemble.AdaBoostClassifier(estimator=None, *, n_estimators=50,  
learning_rate=1.0, algorithm='SAMME.R', random_state=None, base_estimator='deprecated')
```

[\[source\]](#)

AdaBoost



Gradient Boosting

Il Gradient Boosting prevede che i weak learner successivi vadano ad imparare solamente i residui degli stimatori precedenti.

Lo stimatore S1 andrà a fare una prima stima prevedendo un valore.

Invece di ripesare il dataset come AdaBoost calcoliamo i residui e addestriamo un secondo stimatore S2 a prevedere i residui e così via a predire i residui dei residui.

In questo modo costruiamo uno stimatore via via sempre più raffinato.

Normalmente gli stimatori sono decision tree.

Gradient Boosting

Esempio, prevediamo il *salary* con un regressore

Age	Work Exp	Degree	Salary
27	4	B.E.	60k
30	7	B.Com	50k
32	9	B.Sc	65k

Calcoliamo i residui dalla nostra stima

Age	Work Exp	Degree	Salary	Predicted Salary	Residual
27	4	B.E.	60k	58k	12
30	7	B.Com	50k	58k	-8
32	9	B.Sc	65k	58k	7

Gradient Boosting

Addestriamo quindi un secondo regressore che preveda i residui.

Age	Work Exp	Degree	Salary	Predicted Salary	R1(Actual Residual)	R2(Predicted Residual)
27	4	B.E.	60k	58k	12	8
30	7	B.Com	50k	58k	-8	-4
32	9	B.Sc	65k	58k	7	5

La stima finale sarà data da

$$\hat{y} = \hat{y}_{R1} + \hat{y}_{R2} + \hat{y}_{R3} + \dots$$

XGBoost

eXtreme Gradient Boosting è un algoritmo che estende il Gradient Boosting includendo la regolarizzazione (L1 e L2).

Un famoso pacchetto di Machine Learning si chiama appunto XGBoost ed implementa in modo molto efficiente e raggiungendo performance molto elevate il Gradient Boosting.

La libreria è nota per poter scalare facilmente in ambienti distribuiti (Hadoop, Spark, etc.), quindi è particolarmente comoda per creare modelli per dataset enormi.



Kernel Methods

Abbiamo introdotto la scorsa lezione i metodi **non-parametrici** chiamati anche Kernel Methods in quanto basati sulla metrica di distanza «kernel».

Ripartiamo dalle slide del K-NN

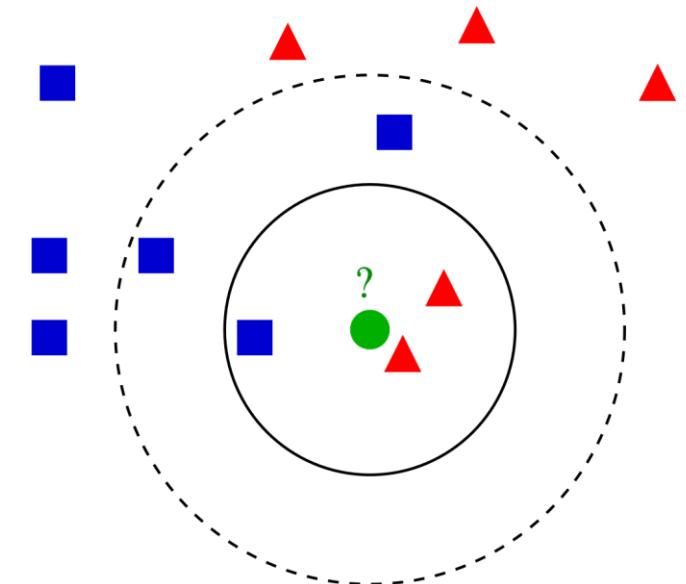
k-Nearest Neighbor

Il k-NN è un semplice algoritmo di classificazione e regressione **non-parametrico**.

Nel dover decidere a che classe appartiene un esempio possiamo confrontare il nostro vettore x con i vettori di altri k esempi più vicini secondo una *metrica*.

La decisione viene presa per majority voting nei casi di classificazione, come media nei casi di regressione.

Vengono scelti k dispari, $k=1$ indica di prendere il più vicino.



k-Nearest Neighbor

Tutti gli esempi nel dataset devono essere confrontabili tra loro secondo una metrica chiamata **kernel**.

$$K(x_1, x_2) = \langle f(x_1), f(x_2) \rangle$$

La funzione kernel applica indipendentemente a x_1 e x_2 una trasformazione in uno spazio m -dimensionale, normalmente maggiore della dimensione del vettore.

Alla fine viene eseguito il prodotto scalare delle due componenti risultando in uno scalare. In questo modo siamo in grado di confrontare esempi diversi del dataset e darne un ordinamento.

k-Nearest Neighbor

Il principale svantaggio del k-NN è la sua scarsa scalabilità.

Non avendo un modello parametrico del nostro problema, ad ogni nuovo esempio dobbiamo rieseguire tanti confronti quanti sono i nostri dati etichettati.

Questo è tanto più oneroso tanto più cresce il dataset

Non c'è fase di training, ha costo 0

Ogni inferenza è lineare o poco più con la lunghezza del dataset

k-NN

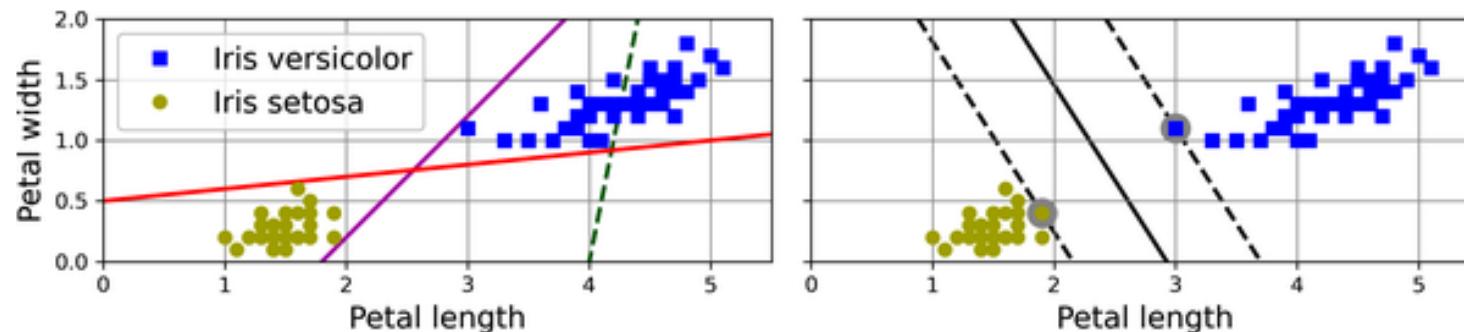
Prima di procedere oltre andiamo a vedere concretamente come si differenzia da un modello tradizionale.

<https://colab.research.google.com/drive/1HzaZ7ajAkpH8Izlw9MB250VjVfwbQP2R>

Support Vector Machines

Come abbiamo visto, uno dei limiti del kNN è che in inferenza (in teoria, a meno di ottimizzazioni da parte di sklearn) deve fare N confronti.

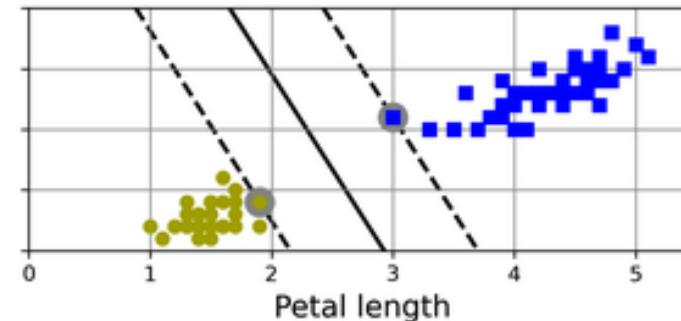
L'idea alla base di un'altra tecnica non parametrica, Support Vector Machine, è riassunta nell'immagine qui sotto. Con dei classificatori lineari siamo in grado di distinguere i due gruppi, ma l'ideale è trovare il classificatore con il margine maggiore, anche detto «*large margin classification*».



Support Vector Machines

SVM è una tecnica non-parametrica in quanto seleziona all'interno del dataset stesso quei punti che gli consentono di avere il massimo margine.

Aggiungere punti nei rispettivi gruppi non cambia il decision boundary perché è *supportato* dai punti sul margine, ovvero i **support vectors**.

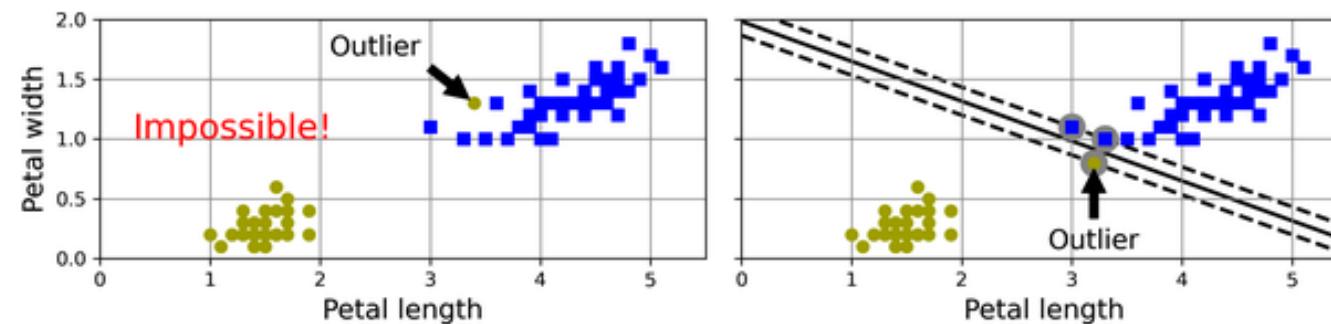


Hard Margin vs Soft Margin

Se imponessimo che tutti i punti di training devono esser correttamente classificati stiamo imponendo un **hard-margin**.

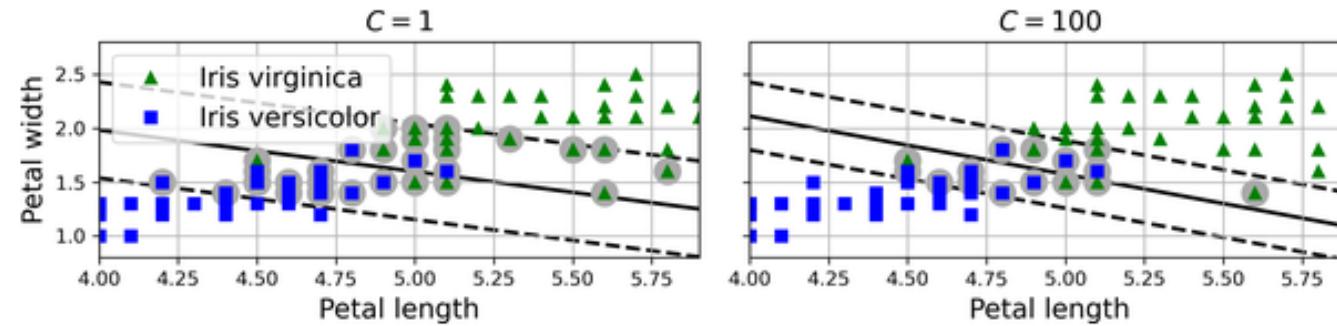
Quello che chiediamo è di candidare tutti i punti nelle aree più prossime come support vectors e quindi imporre che il nostro decision boundary segua un percorso articolato.

Questo approccio è possibile solamente se siamo in assenza di outliers, altrimenti punti che sono classificati erroneamente o condizioni anomale risultano in decision bounday impossibili.



Hard Margin vs Soft Margin

Nel caso in cui tollerassimo invece delle violazioni, anche in training, del decision boundary allora trattiamo un *soft-margin*. Nella SVM questo viene gestito tramite un termine di **regolarizzazione C**. Valori di C più bassi consentono di allargare l'area di supporto ma comporta maggiori violazioni, alti valori riduce l'area di supporto ed il numero di violazioni. Abbassandola troppo si rischia però di avere underfitting cercando in quanto siamo troppo «tolleranti».



Support Vector Machine

A differenza degli altri modelli visti, SVM non ha il `predict_proba`, in quanto osserva solamente da che lato del decision boundary si trova il punto, non ha la possibilità di stimarne una probabilità associata.

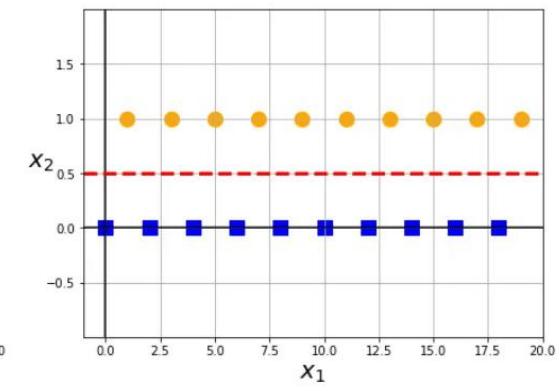
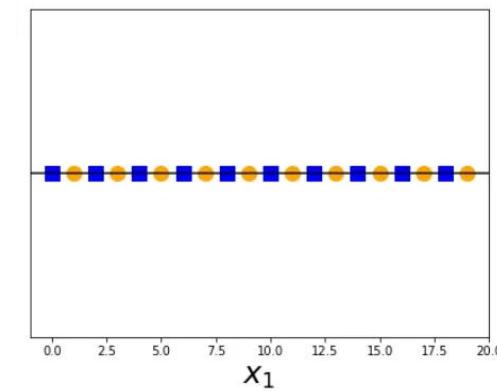
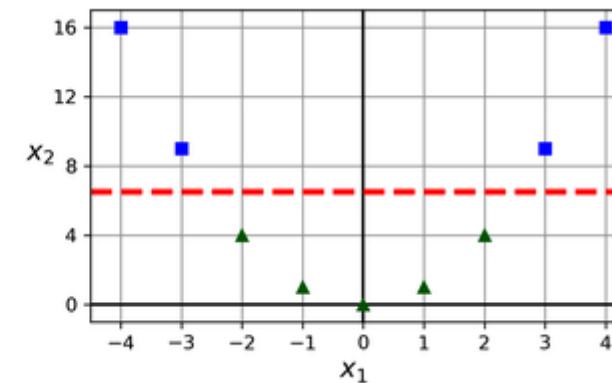
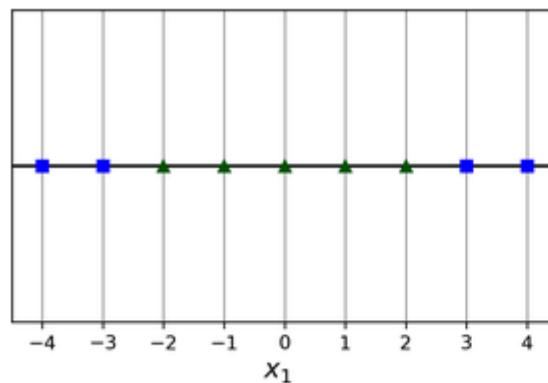
Come abbiamo visto fino ad ora, SVM prova a fissare un decision boundary lineare per separare le classi.

Come possiamo separare però classi non facilmente separabili (come l'iris) con SVM?

Linearly Separable dataset

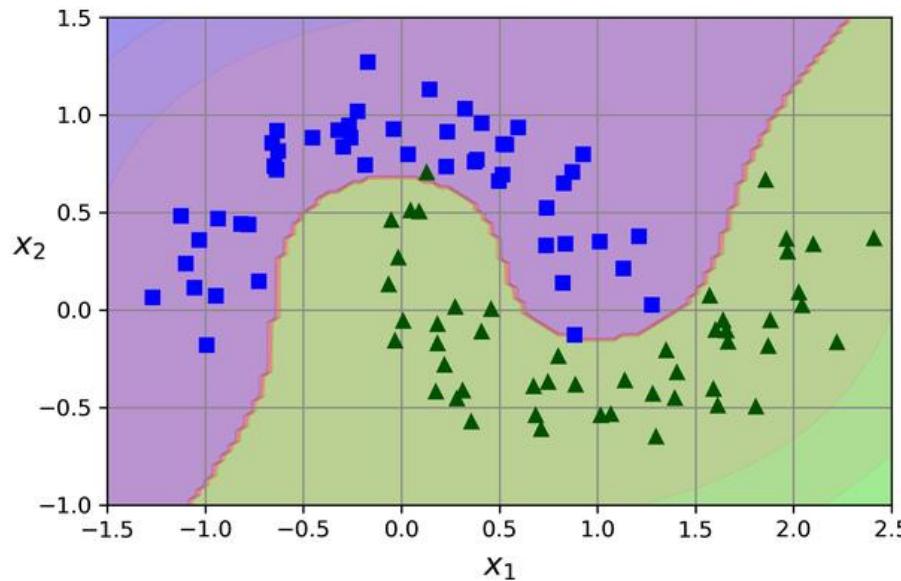
Possiamo trasformare le nostre features in modo che siano linearmente separabili.

In questo caso la nostra x_1 non sarebbe mai separabile linearmente, ma se calcolassimo x_1^2 potremmo facilmente separare le due classi. Applicando delle funzioni di base possiamo proiettare il nostro spazio delle features in uno più grande in cui le features possono essere separate linearmente.



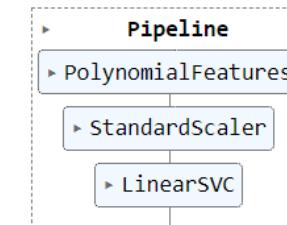
Linear SVC

Possiamo usare le SVM per fare classificazione con un Linear Support Vector Classifier in Sklearn. Opportunamente preparando le features con uno sviluppo polinomiale possiamo separare anche forme molto complesse senza preoccuparci del fatto che sotto vi sia un decision boundary lineare.



```
[121] X,y=make_moons(n_samples=100,noise=0.15,random_state=42)

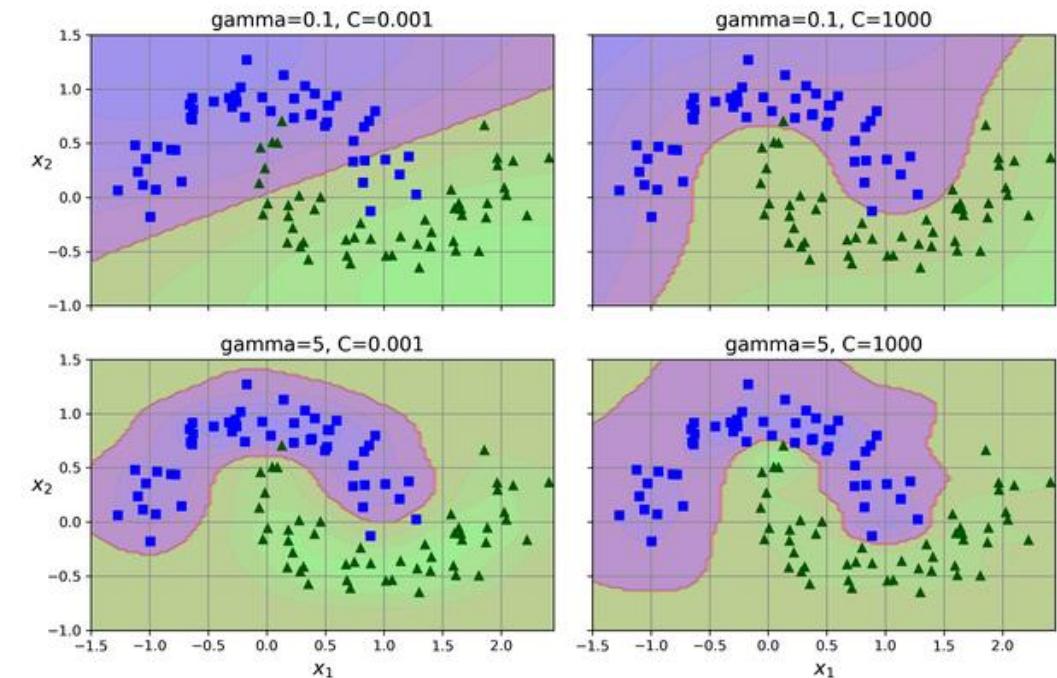
polynomial_svm_clf = make_pipeline(
    PolynomialFeatures(degree=3),
    StandardScaler(),
    LinearSVC(C=10,max_iter=10_000,random_state=42)
)
polynomial_svm_clf.fit(x,y)
```



Non-linear SVM

Possiamo anche scegliere di non usare un decision boundary lineare.

Basandosi sui kernel possiamo scegliere quello che ci fa più comodo per risolvere il nostro problema di classificazione tra cui 'poly' per gli sviluppi polinomiali o 'rbf', per i kernel gaussiani, ciascuno con i suoi parametri (es. degree per il kernel poly)



Kernel Trick

Un passaggio fondamentale nell'SVM è il kernel trick.

Abbiamo già visto che nei metodi a kernel la distanza tra i punti è calcolata da una funzione kernel

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

che applica la stessa trasformazione indipendentemente ai due esempi e poi ne esegue il prodotto scalare.

Se le nostre funzioni di base ci consentono di proiettare i punti in uno spazio in cui sono linearmente separabili abbiamo l'ottimo per il nostro problema.

Kernel trick

Partiamo da un esempio, ipotizziamo

$$\mathbf{x} = (x_1, x_2, x_3)^T$$
$$\mathbf{y} = (y_1, y_2, y_3)^T$$

e come funzione di base lo svolgimento polinomiale al terzo ordine

$$\phi(\mathbf{x}) = (x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2)^T$$
$$\phi(\mathbf{y}) = (y_1^2, y_1y_2, y_1y_3, y_2y_1, y_2^2, y_2y_3, y_3y_1, y_3y_2, y_3^2)^T$$

moltiplicandoli scalarmente otterremo

$$\phi(\mathbf{x})^T \phi(\mathbf{y}) = \sum_{i,j=1}^3 x_i x_j y_i y_j$$

Kernel trick

Potremmo ottenere lo stesso risultato andando semplicemente ad eseguire il prodotto scalare tra x^T e y

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}^T \mathbf{y})^2 \\ &= (x_1 y_1 + x_2 y_2 + x_3 y_3)^2 \\ &= \sum_{i,j=1}^3 x_i x_j y_i y_j \end{aligned}$$

Non figura da nessuna parte le funzioni di base che abbiamo scelto e abbiamo ottenuto lo stesso risultato, questo è il kernel trick.

Altri kernel

In generale il kernel trick per uno sviluppo polinomiale di grado d equivale a

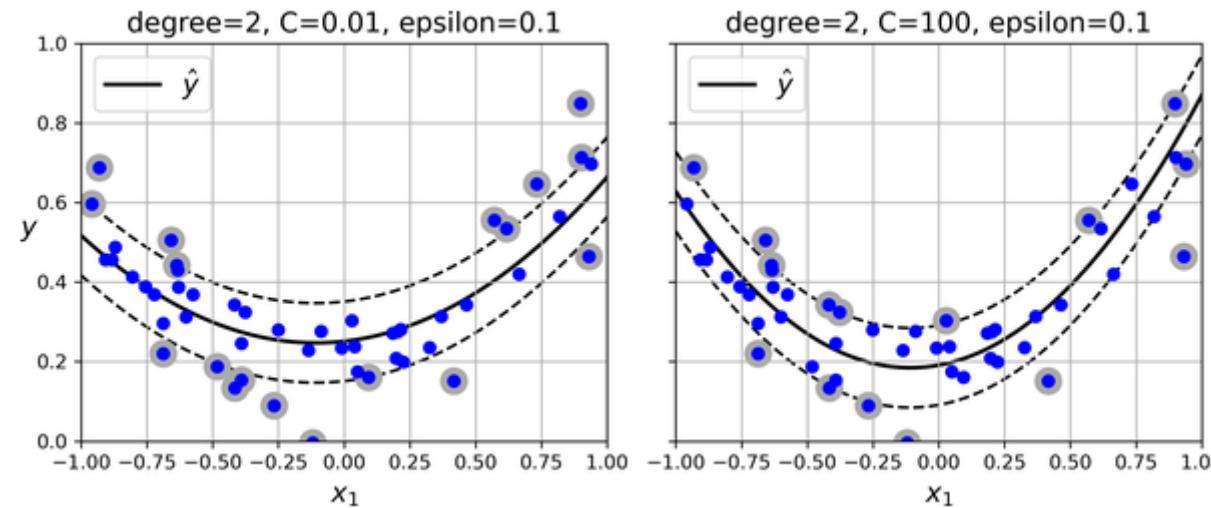
$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

mentre per kernel RBF si ha

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|^2}, \gamma > 0$$

SVM per regressioni

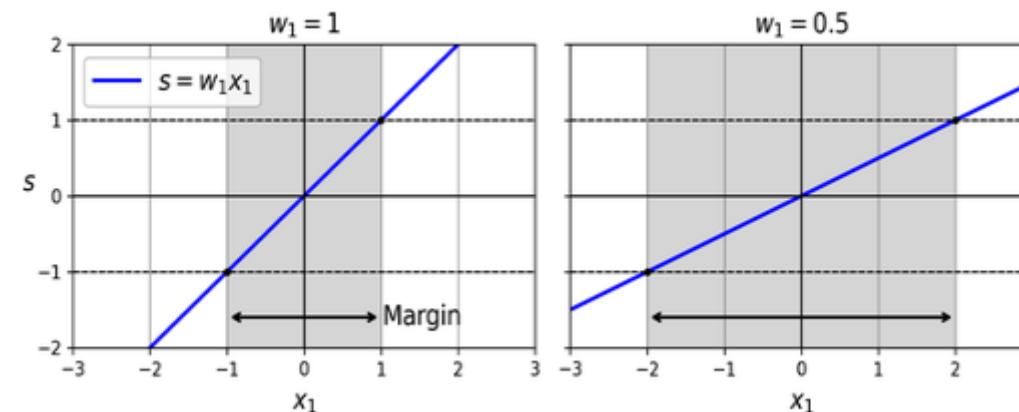
Le SVM per le regressioni riscrivono leggermente il problema, invece di cercare il decision boundary che massimizza la distanza, cerca i punti che consentono di far stare il maggior numero di punti all'interno del margine.



Training

Addestrare una SVM consiste nel trovare i support vector ed i pesi che rendono il margine più grande possibile limitando il numero di violazioni.

Pesi più piccoli comportano margini più grandi.



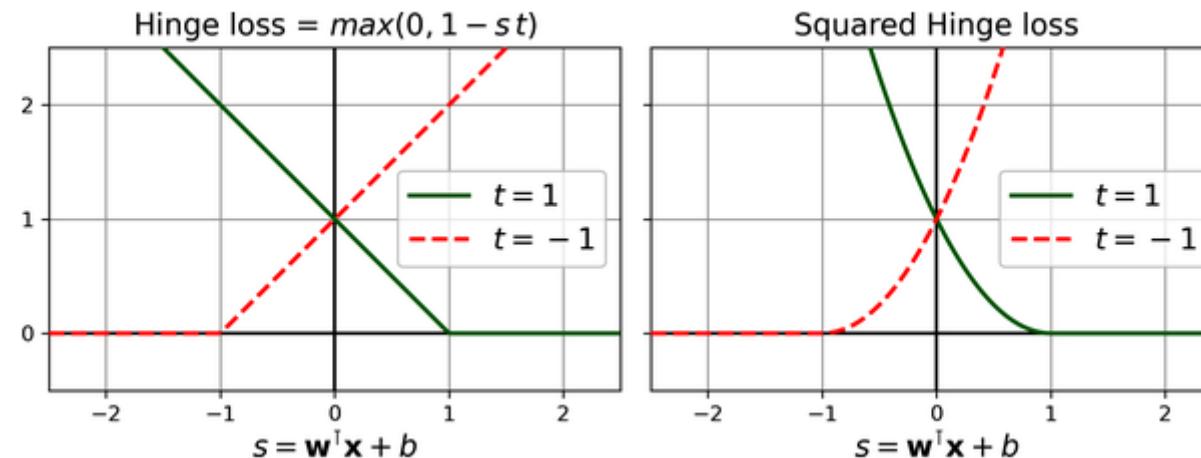
Trovare i punti che massimizzano il margine è un problema di ottimizzazione che si risolve con algoritmi di ottimizzazione vincolata o quadratic programming.

Training

Una funzione di costo che viene spesso usata per l'addestramento delle SVM è la Hinge Loss.

Il punto classificato correttamente non porta nessuna informazione (loss = 0), quelli invece errati contribuiscono linearmente all'errore (o quadraticamente nella versione Squared) e vengono ottimizzate tramite SGD.

$$\ell(y) = \max(0, 1 - t \cdot y)$$



Complessità di calcolo

La complessità dell'SVM, per cui risulta spesso molto più lenta di molte altre tecniche parametriche in addestramento, è che l'apprendimento richiede ottimizzazione numerica per trovare i migliori candidati.

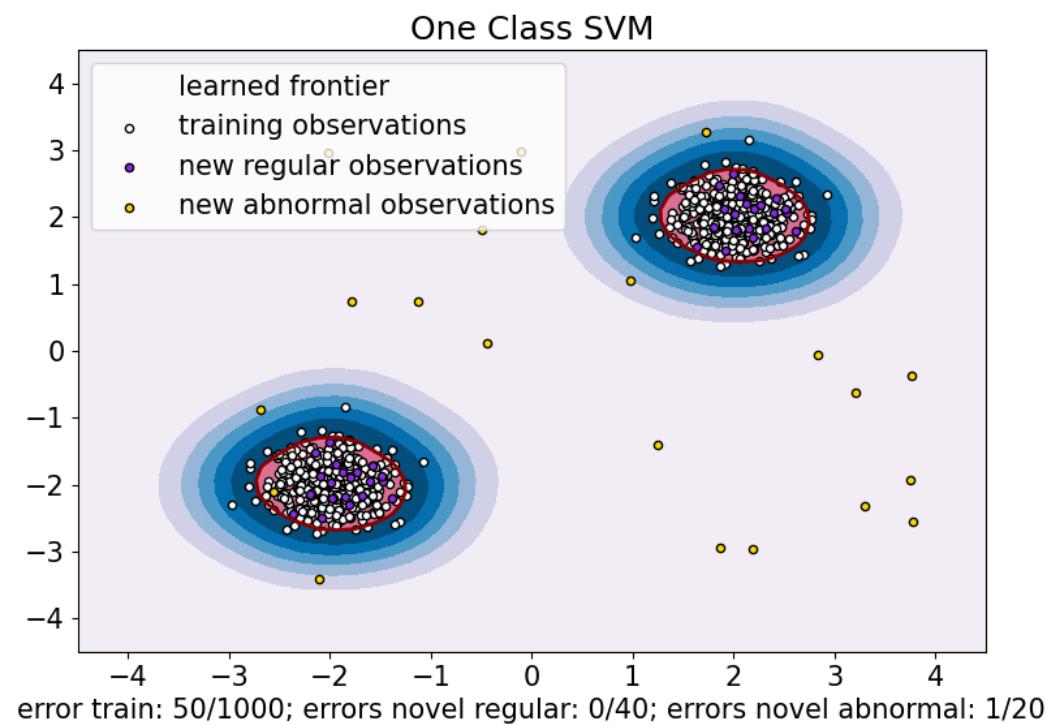
Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
LinearSVC	$O(m \times n)$	No	Yes	No
SVC	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes
SGDClassifier	$O(m \times n)$	Yes	Yes	No

One-Class SVM

Anticipiamo un concetto dei metodi unsupervised.

Le SVM possono anche essere usate per fare novelty detection, in cui non cerchiamo il massimo margine possibile ma la minima area possibile.

Valgono le stesse regole delle SVM per cui possiamo tollerare un numero di violazioni.

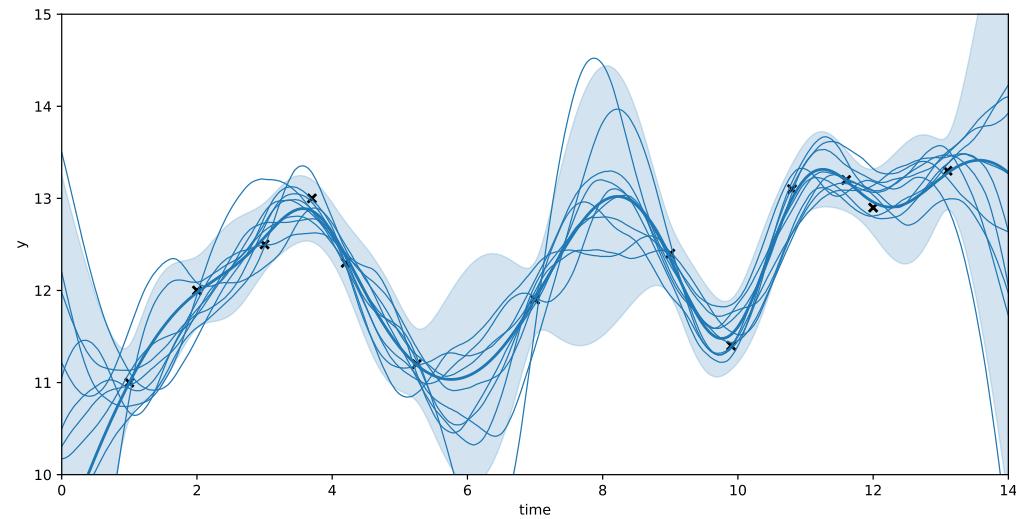


Gaussian Processes

Gaussian Process è una tecnica non-parametrica che interpreta i dati come generati da una distribuzione multivariata descritta tramite matrici di media e covarianze tra tutti i punti.

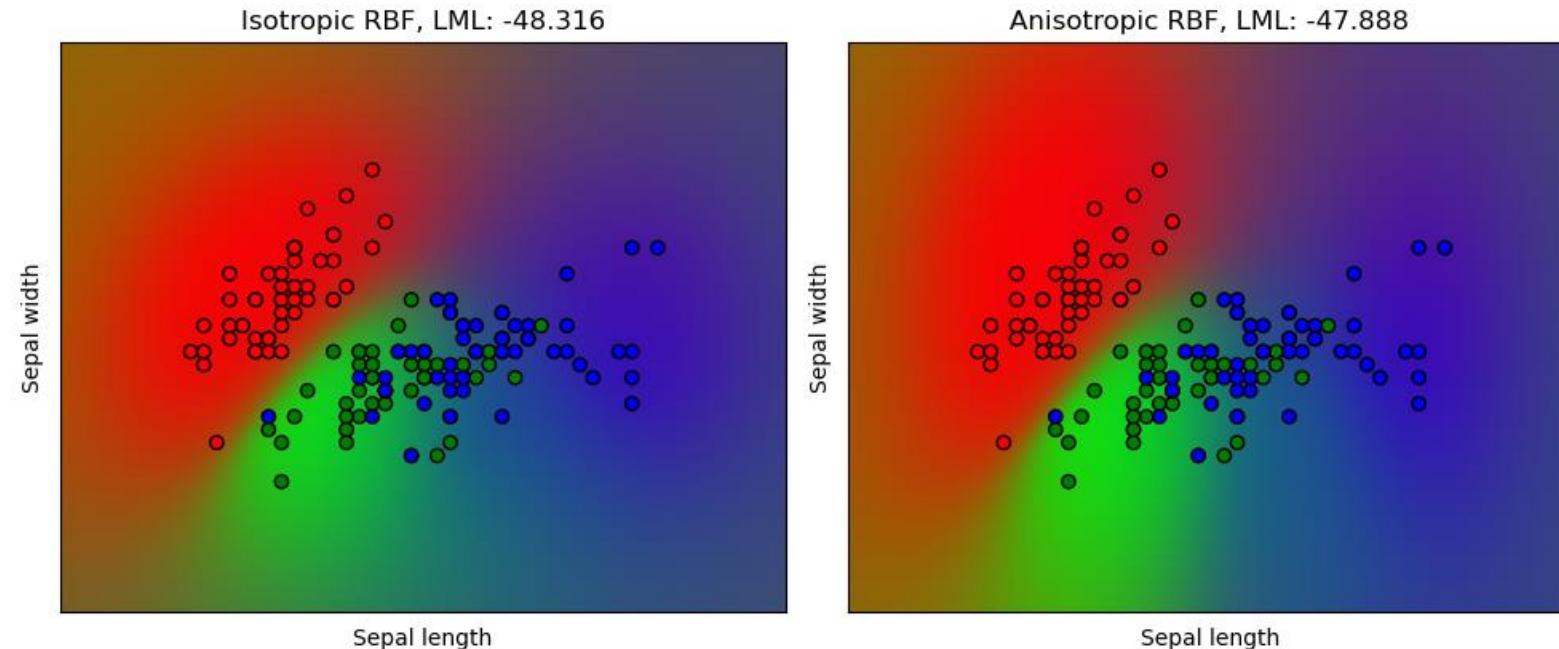
La funzione di kernel è tanto più alta tanto i punti sono vicini. La conoscenza a priori sul processo unita alle osservazioni ci consente di stimare non solo la regressione ma anche l'incertezza associata.

È una tecnica molto onerosa dal punto di vista computazionale che perde di espressività in problemi N-dimensional con N maggiore di una decina.



Gaussian Process

Normalmente sono usati in problemi di regressione, ma i Gaussian Process possono essere usati anche per problemi di classificazione.



Hyperparameter tuning

In tutti i modelli che abbiamo visto ci sono dei parametri (i pesi delle features, gli esempi selezionati, ...) e degli iperparametri che rappresentano tutte le configurazioni dei nostri regressori.

Sono esempi di iperparametri

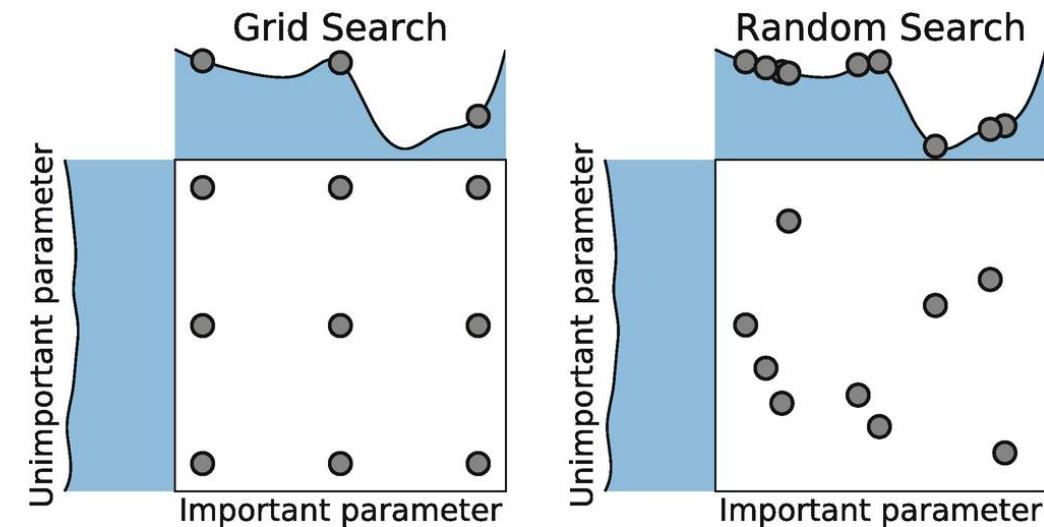
- La penalizzazione da usare ed il relativo peso nelle regressioni
- La profondità dell'albero e lo score di impurità negli alberi
- Il tipo di kernel e il grado di tolleranza alle violazioni nelle SVM

Tutti gli iperparametri vengono in larga parte tunati a mano dal data scientist / ML engineer sulla base della sua esperienza, della comprensione del problema e della conoscenza relativa alla tecnica che si sta usando. Tunando opportunamente un metodo anche meno performante si può raggiungere la performance desiderata.

Hyperparameter tuning

Esistono delle tecniche che ci consentono comunque di provare diversi iperparametri automaticamente in modo da avere la maggior performance in validazione. Queste possono essere una base con cui identificare dei punti di partenza per poi procedere iterativamente a mano nella ricerca dell'ottimo.

Un esempio di tecnica è il **Grid Search**, crea una griglia di valori ed esegue una ricerca esaustiva provando tutte le combinazioni di iperparametri impostati.



Hyperparameter tuning

Chiaramente il GridSearch, e tutti i metodi che consentono di provare N modelli con M iperparametri e K valori per iperparametro in cross validazione, sono computazionalmente molto onerosi.

Nel caso di dataset grandi e con lunghi apprendimenti per arrivare ad un buon livello di performance diventa un problema intrattabile.

In alcuni casi si tende a campionare una parte del dataset, magari limitando il problema solo ad un set di classi più piccolo del totale, per cercare in modo più o meno esaustivo la tecnica che potenzialmente è più performante per poi applicarla sull'intero dataset.

Successivamente vedremo nelle reti neurali il numero di possibili combinazioni di layer, attivazioni, kernel, neuroni, ecc. esplode rapidamente. La sensibilità del data scientist e la conoscenza del problema rimangono la strada maestra per raggiungere un buon risultato.

Recap

Abbiamo visto

- Ensamble methods
- Metodi non parametrici
- Hyperparameter Tuning

Vediamo ora tutto insieme in un problema di regressione ed uno di classificazione

<https://colab.research.google.com/drive/1HzaZ7ajAkpH8Izlw9MB250VjVfwbQP2R>



07 – Unsupervised Learning

Daniele Gamba

2022/2023

Feedback corso

Siamo quasi a metà del corso, nel caso in cui voleste lasciare un commento anonimo su come sta andando, se preferite che vada più veloce o più piano, vi lascio un form che potete compilare quando volete

<https://forms.gle/tHv1c1epEYp4zzj2A>



Lezione precedente

Abbiamo visto

- Ensamble methods
- Metodi non parametrici
- Hyperparameter Tuning

Vediamo ora tutto insieme in un problema di regressione ed uno di classificazione

<https://colab.research.google.com/drive/1HzaZ7ajAkpH8Izlw9MB250VjVfwbQP2R>

In questa lezione

Unsupervised Learning

- Dimensionality Reduction
- Clustering (parte 1)

Quando affronteremo le reti neurali, sempre di unsupervised

- Autoencoders ed embeddings
- Self-supervised
- Anomaly detection

Dimensionality reduction

Abbiamo visto che uno dei problemi è avere abbastanza dati per parametro.

Ridurre le features in ingresso ci consente di avere

- un maggiore rapporto dati-parametri
- una maggiore interpretabilità dei risultati
- una robustezza maggiore del modello

Di contro perdiamo sempre sicuramente in espressività del modello nel caso le features che selezioniamo contengono informazioni utili alla nostra stima.

Dimensionality reduction

Mentre abbiamo visto che le tecniche **regolarizzazione** (es. Lasso) ci consentono di **selezionare** alcune features più significative in automatico, esistono delle tecniche che ci consentono in modo più o meno automatico di **ridurre** la dimensionalità del nostro problema.

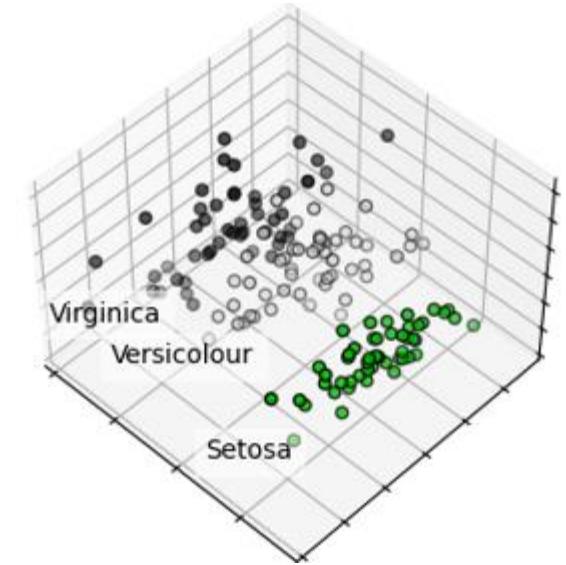
Si riduce la dimensionalità del problema quando vogliamo lavorare con meno features oppure vogliamo riportare il problema in uno spazio visualizzabile (2D-3D) per capire come si sta comportando il nostro regressore.

Data visualizaton

Nel caso del nostro problema dell'IRIS dataset abbiamo 4 features, difficili da visualizzare in un unico grafico.

Possiamo quindi scegliere di visualizzare N selezioni di 2 features o di proiettare i nostri punti in uno spazio 2D partendo da tutte le features.

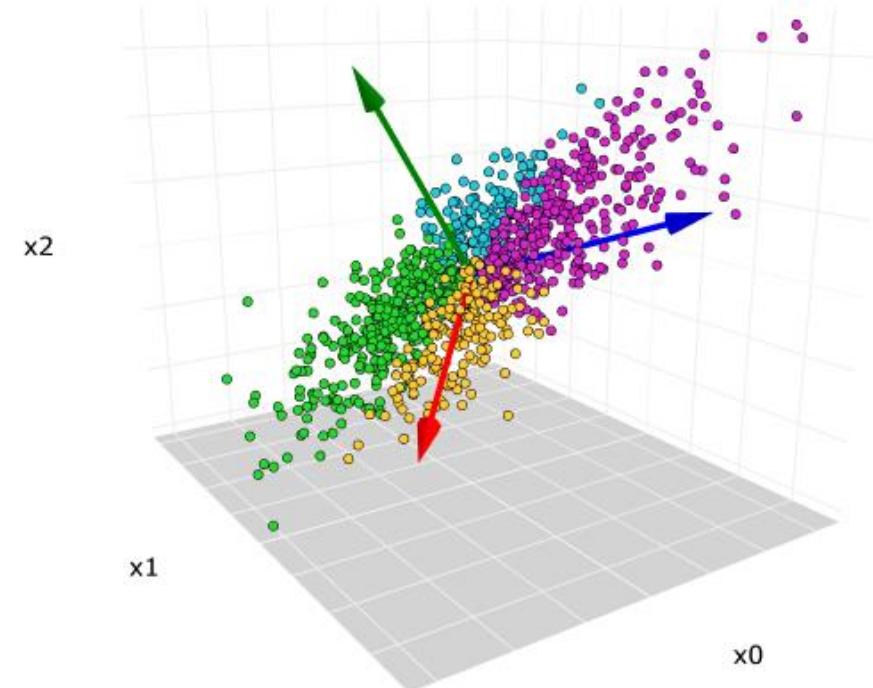
Tanto più il problema ha molte features, tanto più diventa necessario trovare delle tecniche che facciano questa proiezione al posto nostro.



Principal Component Analysis

La **PCA** è una tecnica che viene usata per

- **Data visualization**, ovvero riportare un problema con N dimensioni in 2D / 3D
- **Data compression**, ovvero proiettare i nostri punti in uno spazio con meno dimensioni perdendo un po' di informazione (lossy compression)

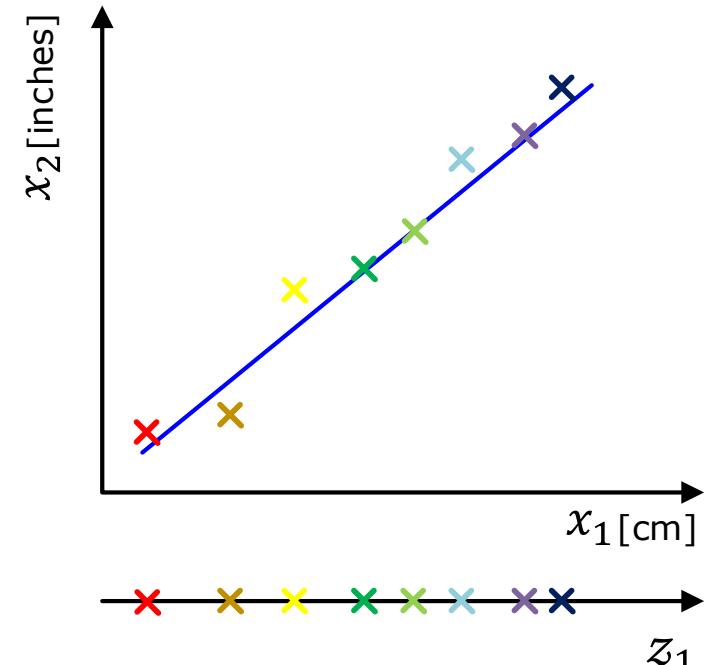


PCA

La PCA viene costruita per **combinazioni lineari delle features** in ingresso e ottiene in uscita delle **variabili scorrelate** tra loro.

L'idea è quella di trovare le features molto correlate tra loro e riassumerle in uno spazio minore con un numero ridotto di variabili.

I valori che si scostano dalla correlazione perfetta vengono approssimati e si perde la loro quota parte di informazione.

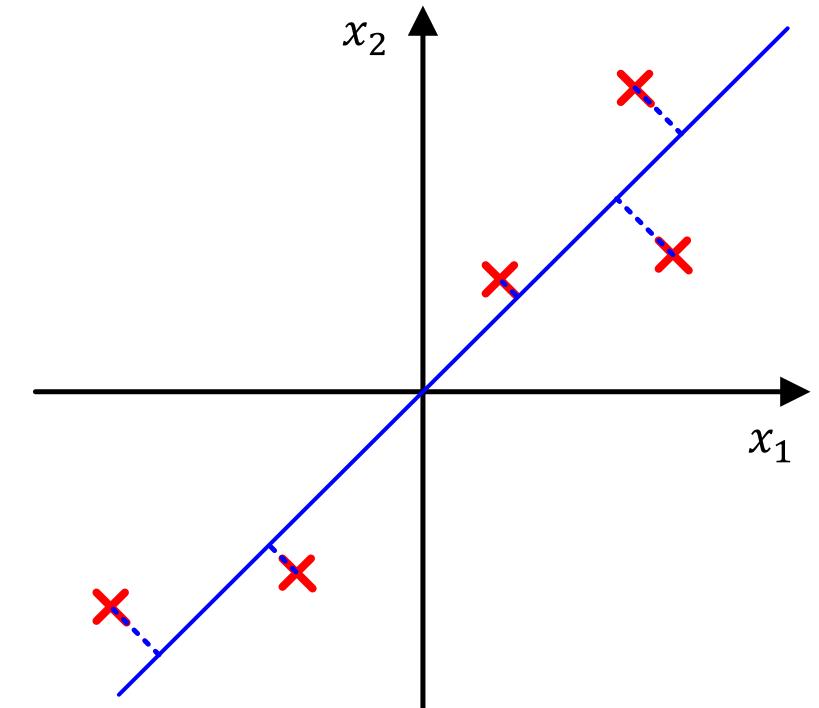


Costruzione

La PCA si ottiene intuitivamente seguendo il seguente processo

- Si identifica la direzione in cui è meglio proiettare i dati, ovvero quella in cui i dati variano maggiormente
- Si trova la forma che minimizza l'errore di proiezione
- Si proiettano i dati e si ri-esegue la procedura

La nuova direzione sarà ortogonale a quella identificata precedentemente. Fortunatamente possiamo calcolare la PCA con un solo calcolo matriciale.



PCA - Implementazione

- Prendiamo le nostre features in colonna X
- Rimuoviamo la media da ogni features e standardizziamo rispetto alla propria deviazione standard, in modo da ottenere ogni features con media zero e deviazione standard 1
- Calcoliamo la **Singolar Value Decomposition** SVD della matrice normalizzata $\tilde{X} \in \mathbb{R}^{N \times d}$

$$\tilde{X} = USV^T$$

$$U = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}_{N \times N}$$

$$S = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}_{N \times d}$$

$$V^T = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}_{d \times d}$$

PCA - Implementazione

Le colonne della matrice U sono una base ortonormale di \mathbb{R}^N

Gli elementi sulla diagonale di S sono i **valori singolari** di \tilde{X}

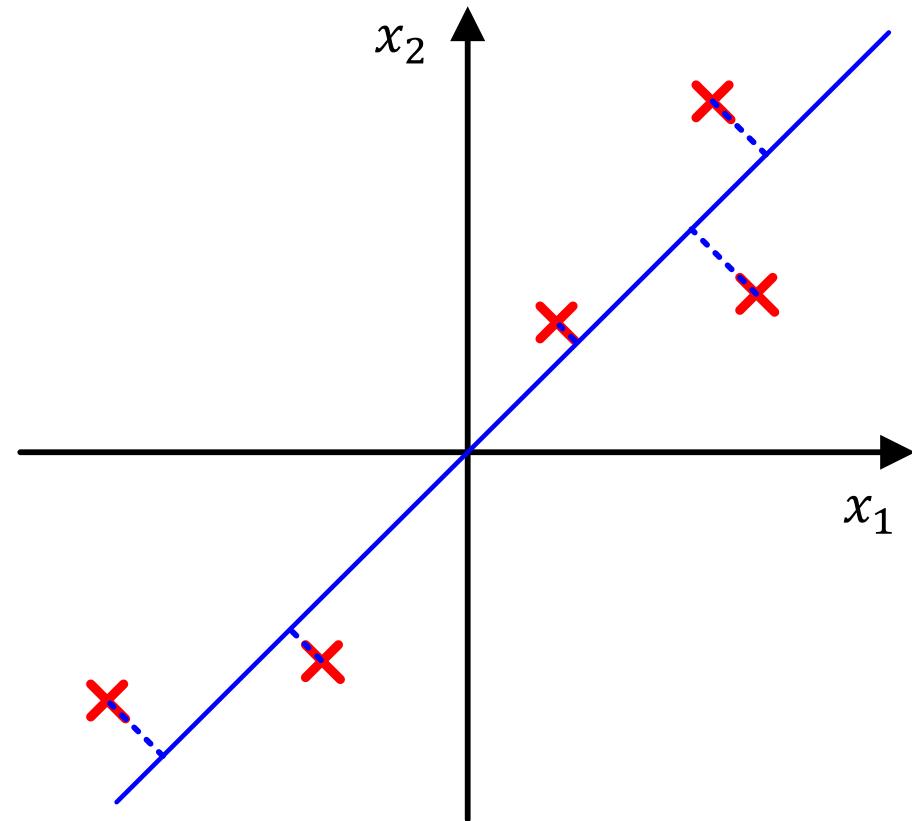
Le colonne della matrice V sono la base ortonormale di \mathbb{R}^d e sono gli autovalori della matrice di covarianza $\Sigma = \frac{1}{N} \tilde{X}^\top \tilde{X}$

Prendendo le prime $q \leq d$ colonne da V, otteniamo la nostra matrice ridotta V_q

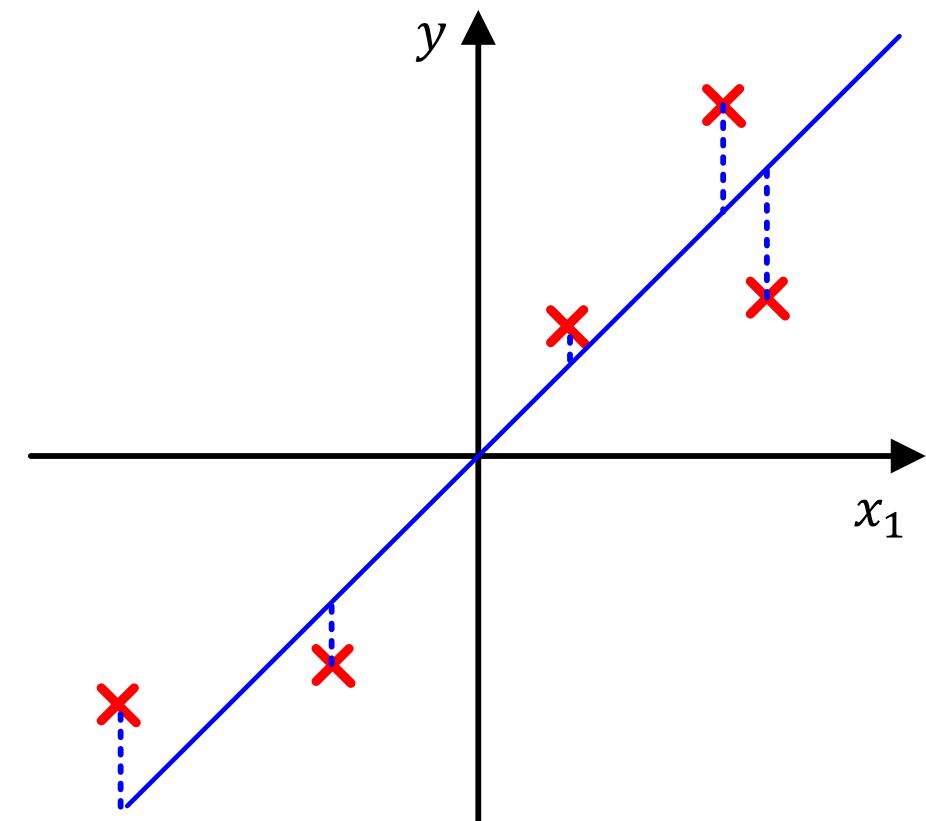
Possiamo calcolarci i nostri nuovi valori proiettati come $Z = \tilde{X}V_q \in \mathbb{R}^{N \times q}$

PCA vs Linear Regression

PCA



Linear regression



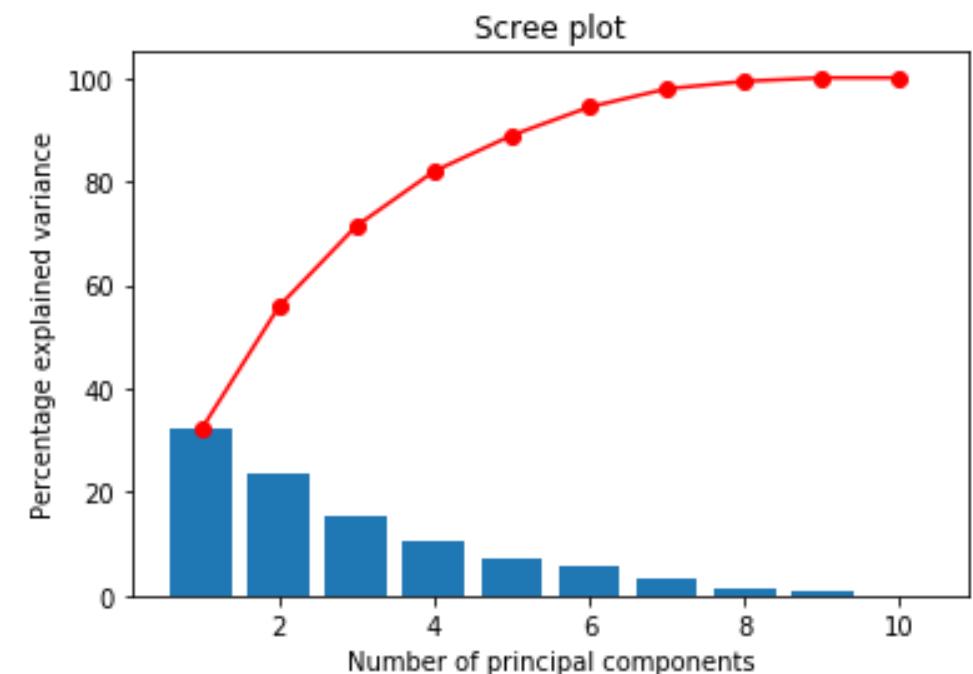
Varianza spiegata

Nel caso della PCA stiamo cercando di trovare le proiezioni che spieghino maggiormente la varianza del dataset.

Di conseguenza ciascuna variabile calcolata «spiega» un pezzo della varianza.

Possiamo usare questa informazione per **scegliere il numero q di componenti** che vogliamo ottenere in funzione della varianza che tolleriamo perdere.

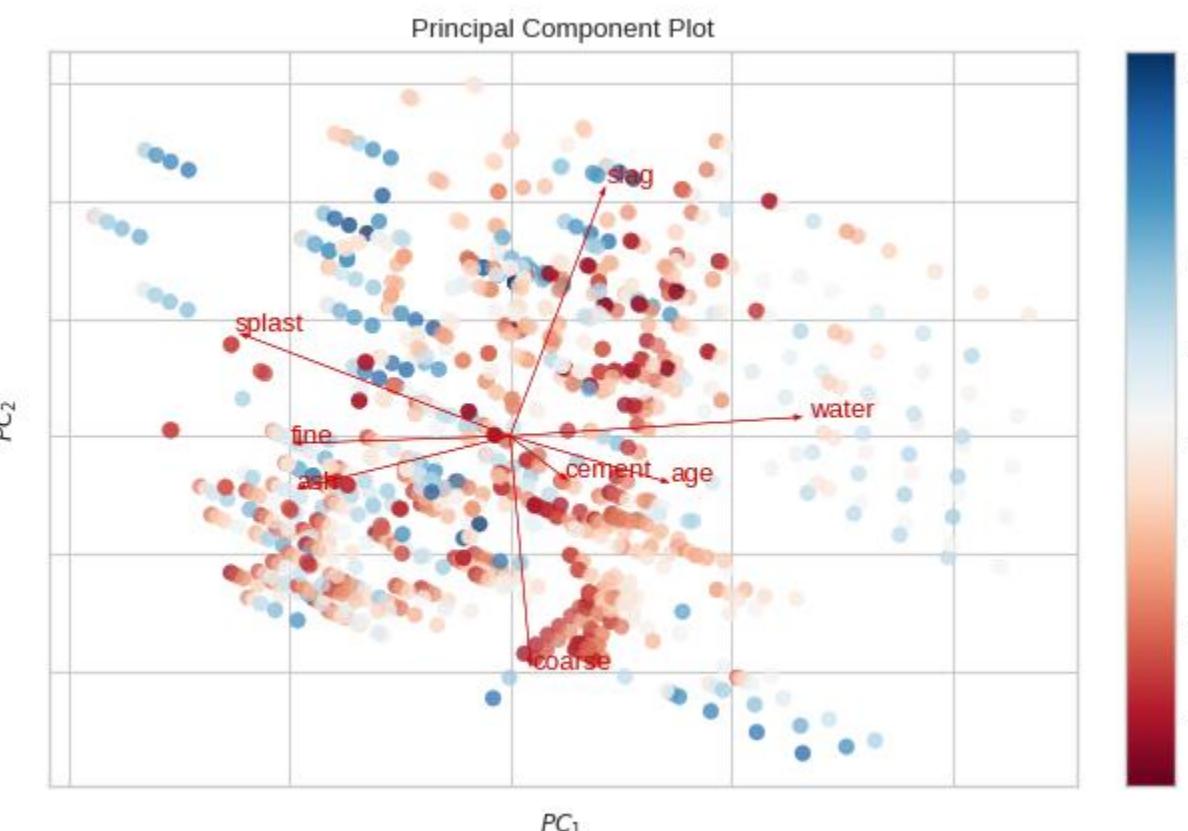
In molti casi ci accorgeremo che aggiungere componenti aggiungerà poco o nulla alla varianza del dataset proiettato.



Assi proiettati

A ritroso possiamo tornare ad interpretare le componenti principali per capire come vengono composte.

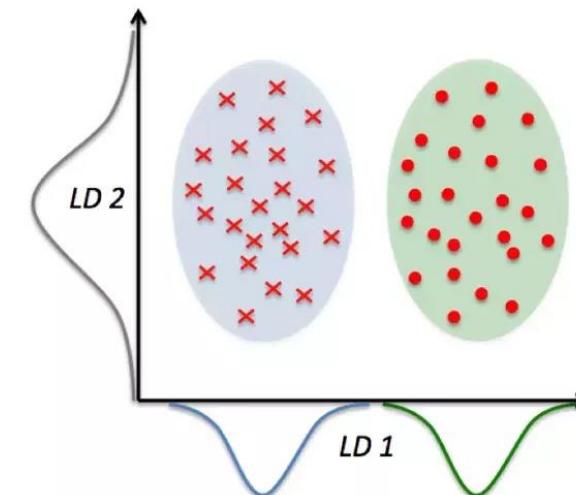
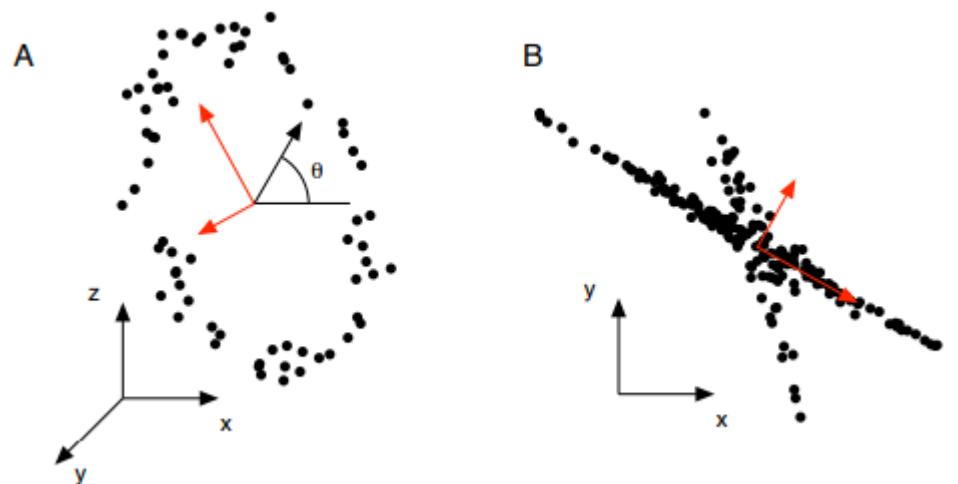
Questo ci consente di interpretare meglio le variabili create e provare a darne una spiegazione.



Svantaggi della PCA

La PCA ha alcuni svantaggi

- Assume che le variabili siano linearmente correlate, in molti casi sappiamo già che non lo sono
- Vengono considerati solamente gli assi a maggior varianza, ci sono alcuni casi in cui questo comporta errori enormi

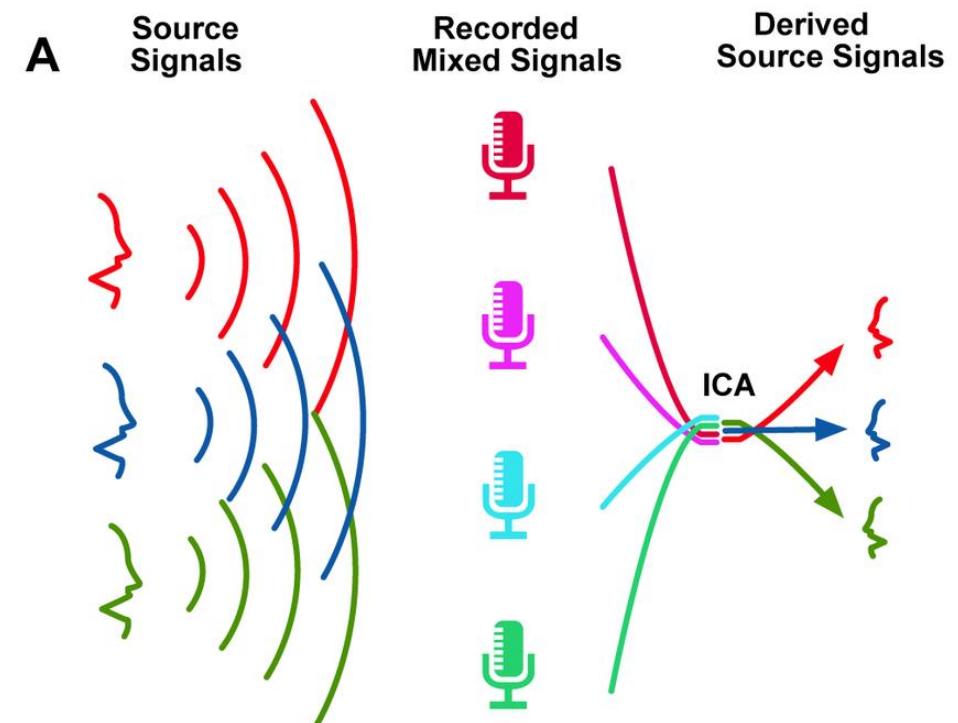


Indipendent Component Analysis

La ICA è una tecnica di unsupervised learning che non serve a ridurre il numero di features, ma a rendere le componenti indipendenti.

Viene spesso usata quando si deve trasformare il dataset per separare fonti di informazioni diverse, ad esempio per riconoscere 3 persone diverse che parlano avendo accesso alle registrazioni di 3 microfoni nella stessa stanza.

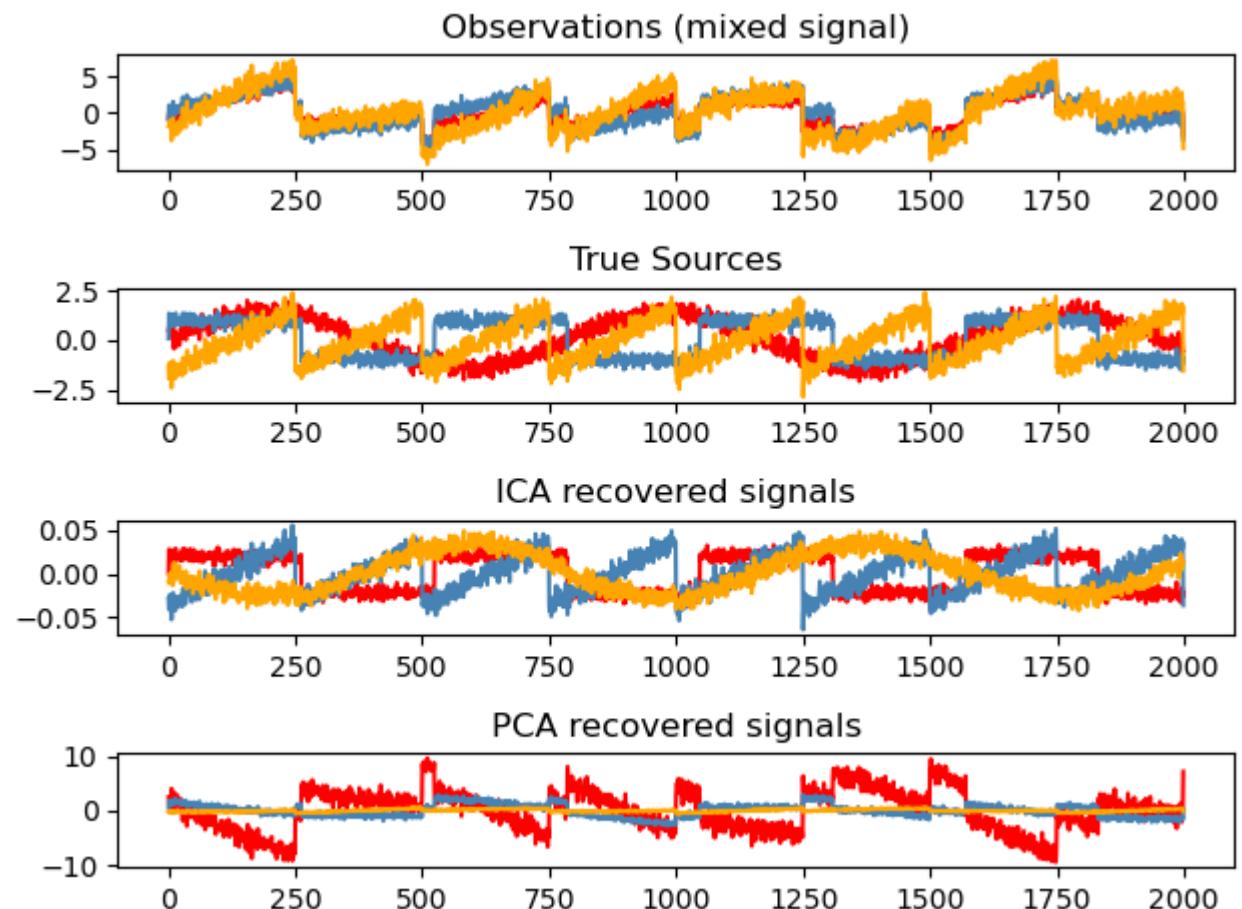
La ICA ci consente di avere delle features in ingresso al modello più robuste e potenzialmente interpretabili, riducendo i termini di correlazione.



PCA vs ICA

PCA e ICA hanno obiettivi diversi e vanno utilizzate correttamente entrambe.

Vediamo la differenza con questo esempio di ricostruzione delle sorgenti.



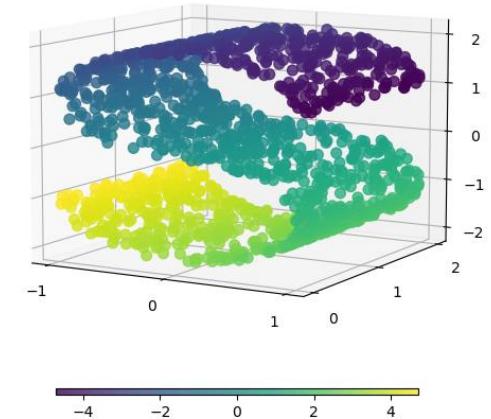
Non linearità

L'assunzione di linearità della PCA è molto comoda perché possiamo calcolare molto velocemente tutte le componenti e la loro importanza (la varianza spiegata).

Di contro sappiamo bene che non tutte le features hanno relazioni lineari, per cui ci serve approfondire tecniche di dimensionality reduction che riescano a gestire bene queste non linearità.

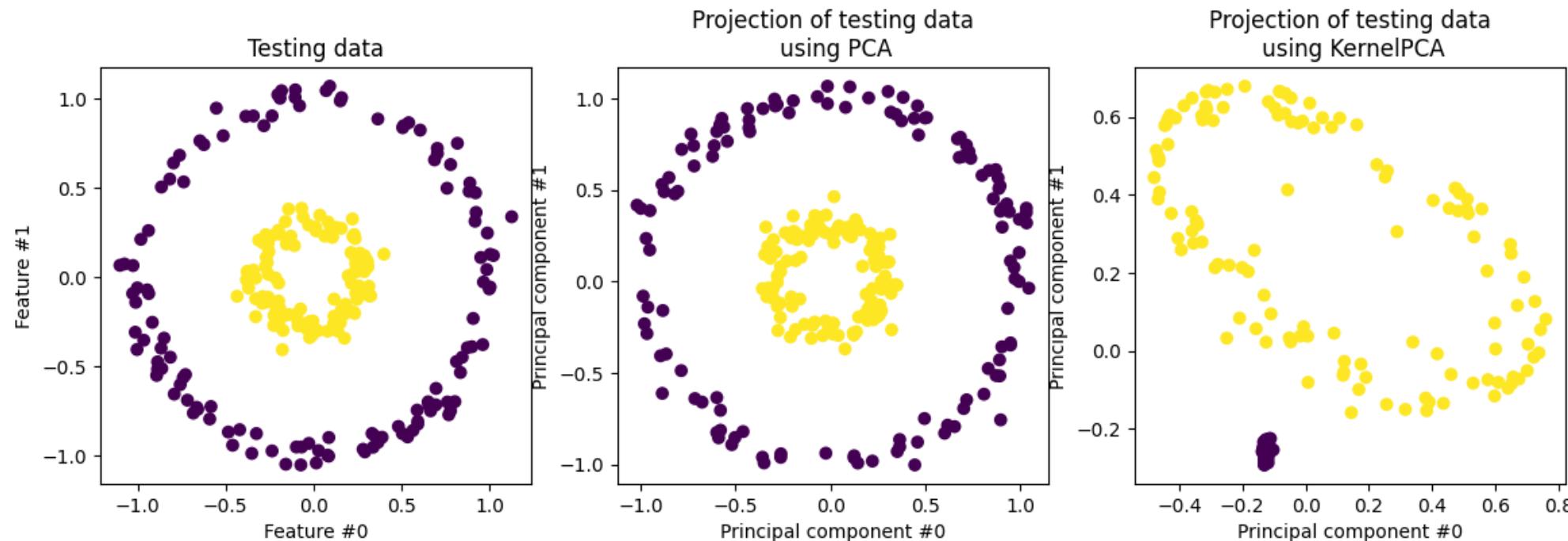
Vedremo una veloce carrellata di queste tecniche, chiamate anche *manifold learning*.

Original S-curve samples



Kernel PCA

Esattamente come per i metodi di classificazione non-parametrici, anche in dimensionality reduction possiamo usare un **Kernel** per trovare uno spazio delle features linearmente separabile per poi applicarvi la PCA.



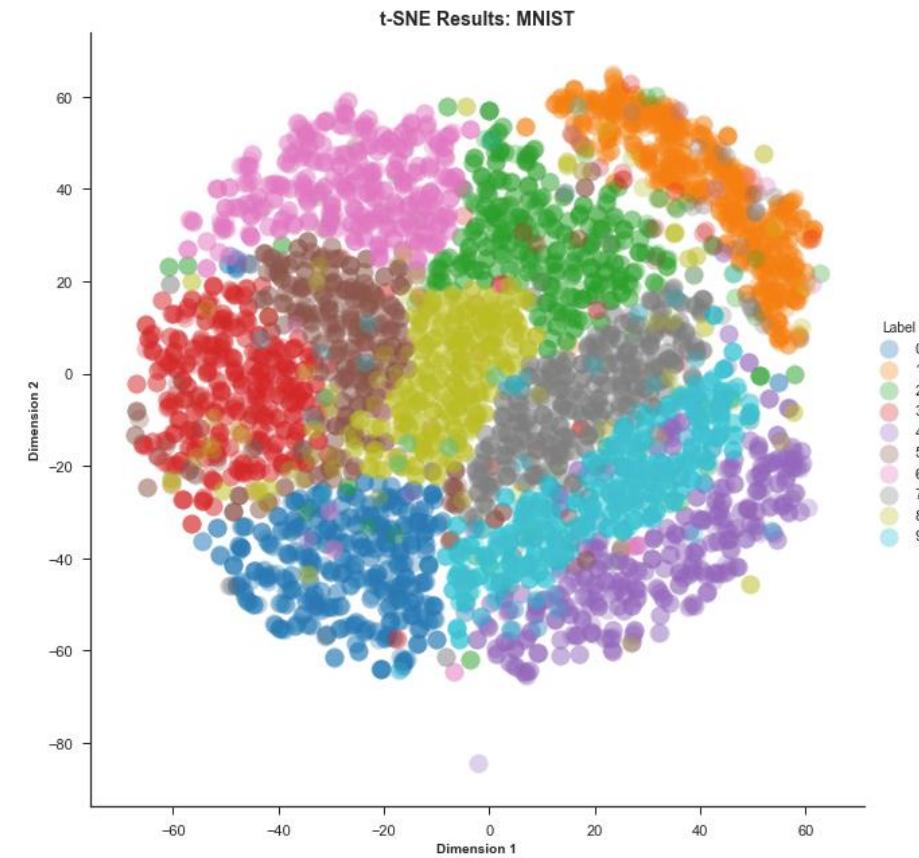
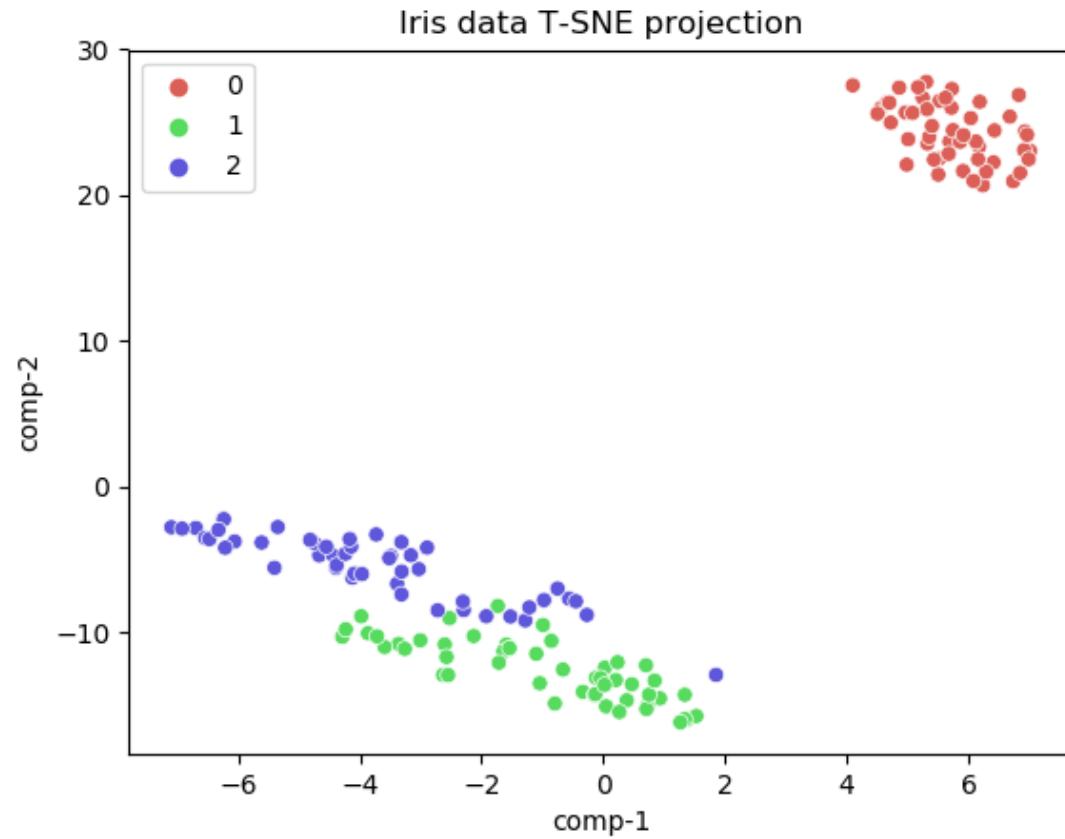
t-SNE

Il **t-distributed stochastic neighbor embedding**, o t-SNE, è una tecnica molto usata per proiettare problemi con un numero di features molto elevato in uno spazio 2D o 3D per visualizzazione.

Si basa sul confronto tra la distanza dei punti in modo che punti vicini nello spazio N dimensionale siano vicini anche nello spazio 2D.

Vengono costruite due distribuzioni di probabilità di vicinanza, una nello spazio N dimensionale, ed una nello spazio ridotto. L'algoritmo iterativamente prova a spostare i punti della seconda in modo che si minimizzi la distanza (KL Distance) tra le due distribuzioni.

t-SNE

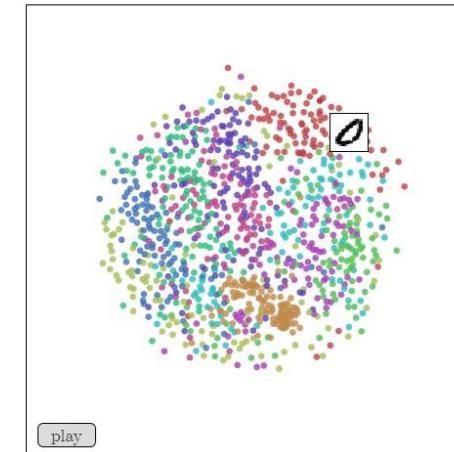


MDS - Multi Dimensional Scaling

MDS è un'altra tecnica utilizzata per fare dimensionality reduction.

Il concetto è simile a quello del t-SNE, si calcolano distanze tra gli elementi in una «dissimilarity matrix» in questo caso. Invece però di stimare due distribuzioni e minimizzarne la differenza in questo caso MDS prova a mantenere inalterate le distanze dei punti ottimizzando una funzione chiamata *Strain* con gli elementi B calcolati a partire dalla dissimilarity matrix D .

$$\text{Strain}_D(x_1, x_2, \dots, x_N) = \left(\frac{\sum_{i,j} (b_{ij} - x_i^T x_j)^2}{\sum_{i,j} b_{ij}^2} \right)^{1/2},$$



Visualizing MNIST with MDS

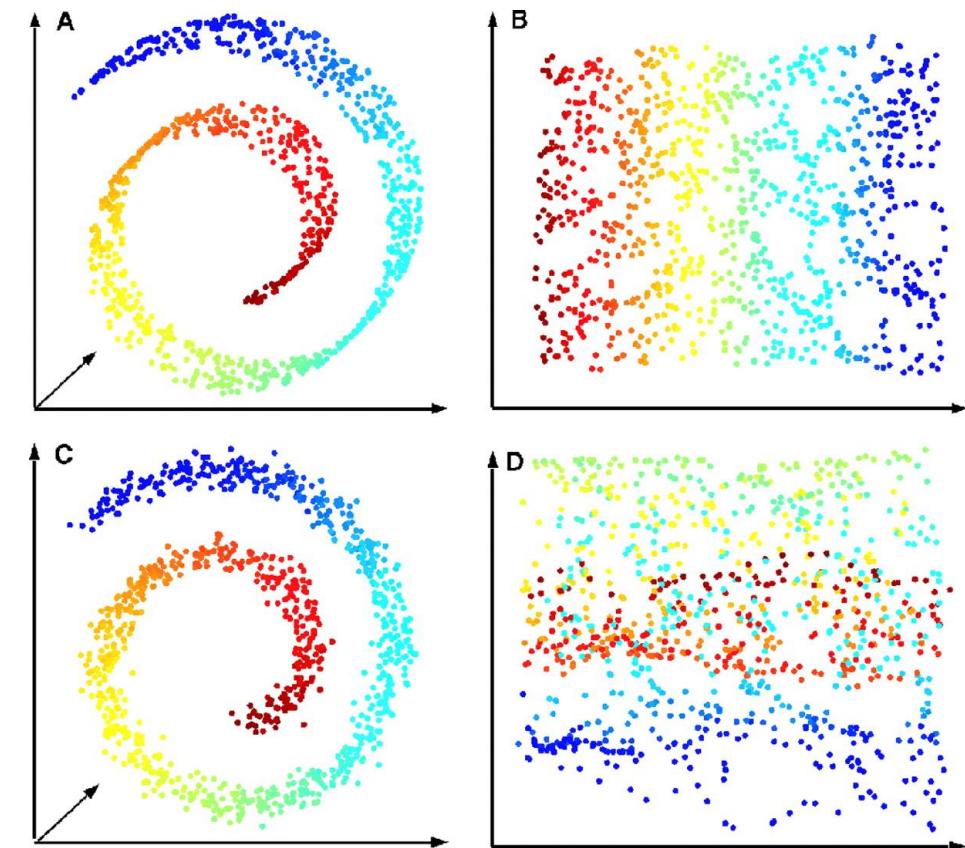
Isomap

Isomap è un altro algoritmo basato sulle distanze tra i punti ma in questo caso lavora a livello locale.

Iterativamente

- determina i punti più vicini ad ogni candidato (es. all'interno di un raggio o i suoi k-NN)
- costruisce un grafo di vicini con gli archi pesati in funzione della distanza
- calcola la minima distanza tra due nodi tramite Dijkstra o altre tecniche per poi calcolarsi il nuovo spazio di proiezione

In questo modo è molto efficace per casi complicati come la spirale, in cui i dati hanno distanze locali molto più significative degli assi di varianza.



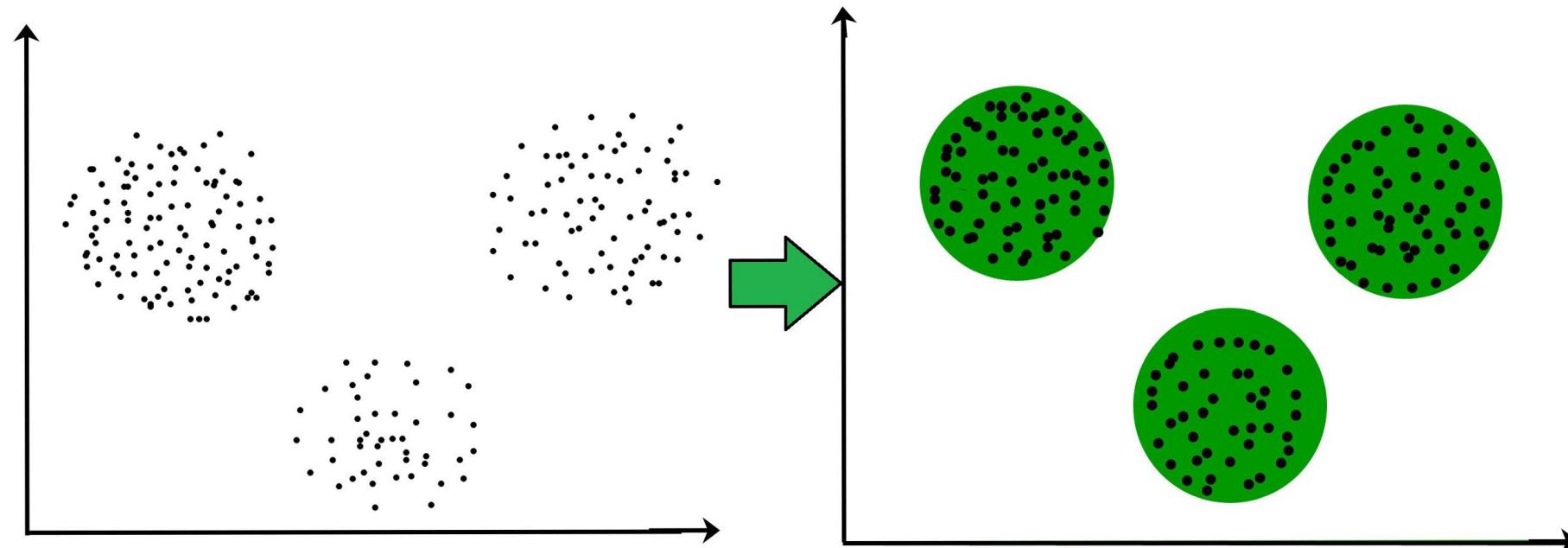
Esercitazione

Proviamo ad eseguire queste tecniche di dimensionality reduction su alcuni problemi noti

https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz_yT9_quAuBqd0Fm0?usp=sharing

Clustering

Il clustering è un ambito dell'unsupervised learning in cui andiamo a trovare «cluster» ovvero gruppi di punti simili tra loro all'interno dei dati senza avere una label.

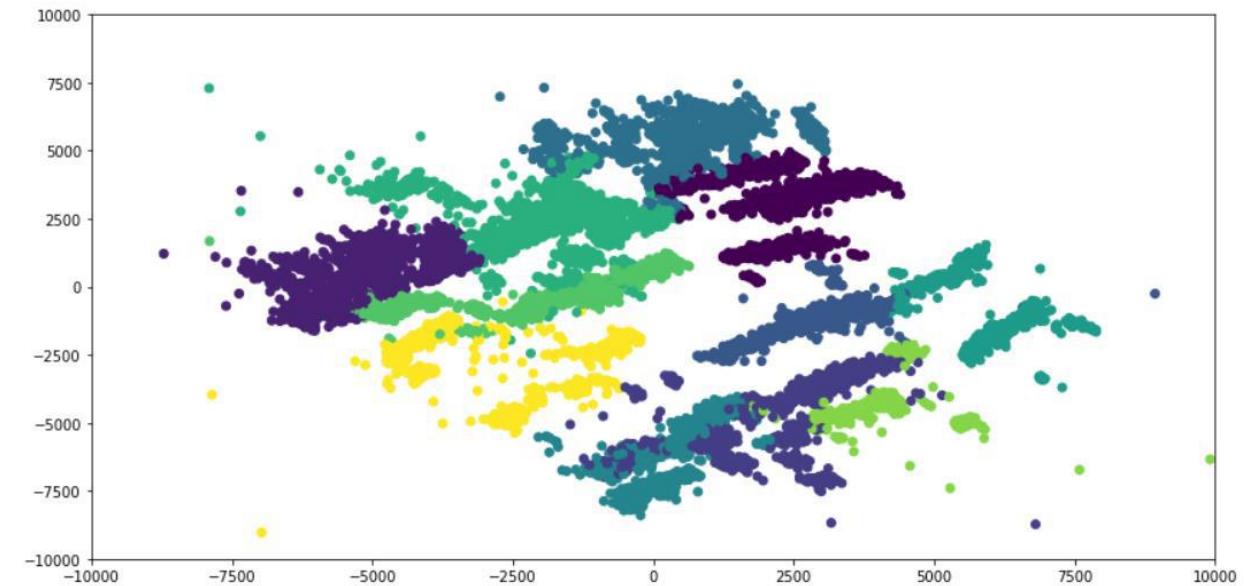


Clustering

Fare clustering ci consente di interpretare pattern presenti nei dati senza una loro esplicitazione.

Questo è molto utile nei casi in cui abbiamo

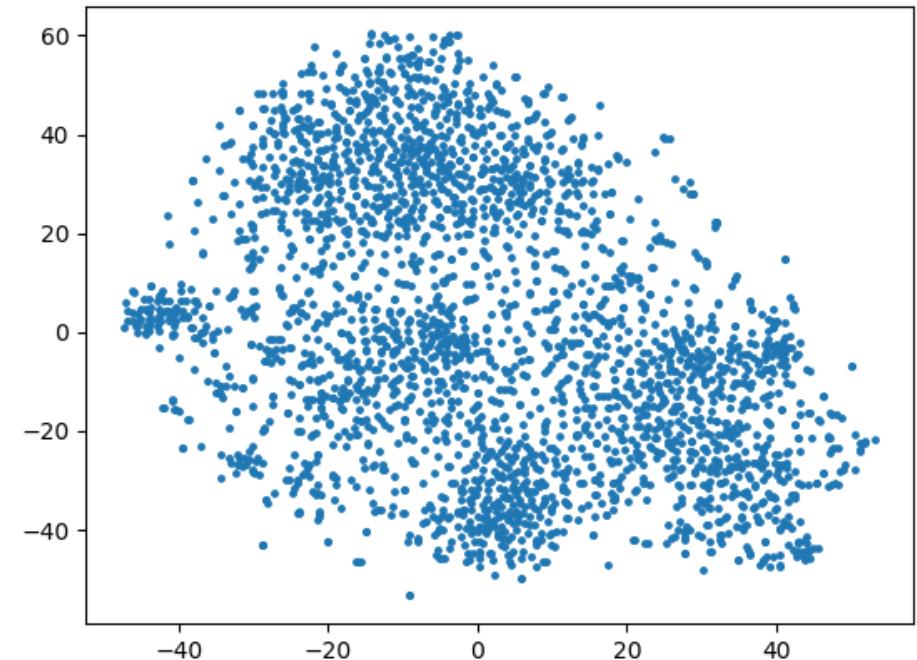
- Gruppi di utenti che si comportano in modo simile
- Posizionamento dei prodotti nel mercato
- Film simili
- ...



Problema

Un cliente, operante nel mondo del customer service, ci fornisce una grossa quantità di richieste di assistenza via chat per un produttore di cialde da caffè.

L'obiettivo è identificare le richieste più frequenti, non sapendo ciascuna richiesta storicizzata a che gruppo appartiene per riportare al cliente le aree su cui intervenire maggiormente per migliorare la qualità del servizio.



Problema

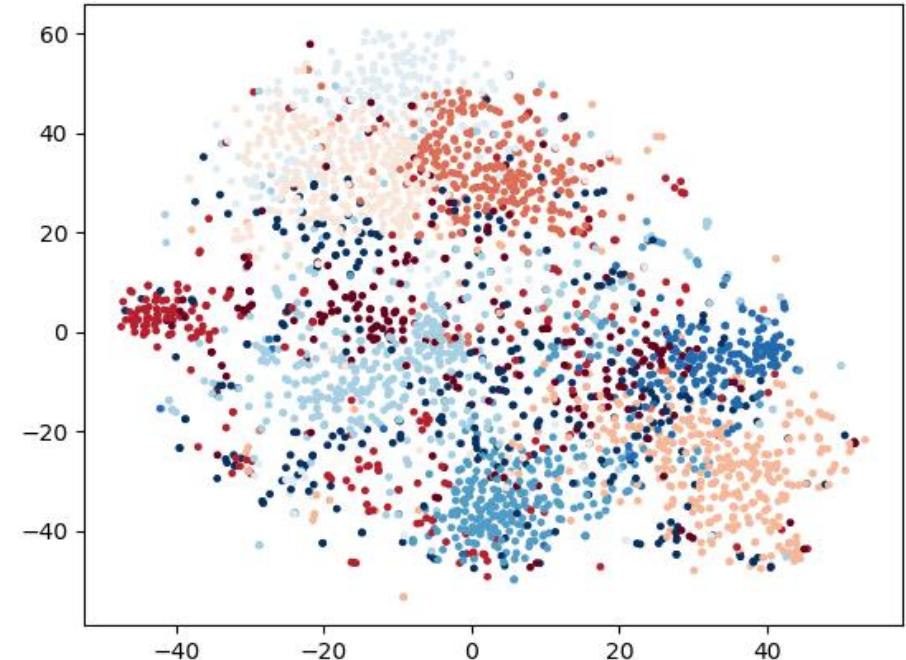
Dati i testi ci serve

- Un modo di codificare le nostre richieste di assistenza
- Un modo di proiettarle in uno spazio

Le tecniche per il testo le vedremo più avanti, assumiamo di avere un metodo che date due frasi ci calcola una distanza semantica.

Possiamo scegliere di calcolarci tutte le distanze tra tutte le frasi o campionarne un certo numero random ed usarle come basi del nostro spazio. A quel punto abbiamo le nostre features, distanze da degli esempi specifici, e possiamo andare a strutturare il nostro problema di clustering.

Il clustering è meglio farlo in uno spazio N dimensionale più grande e poi per comodità visualizzarlo in 2D o 3D.

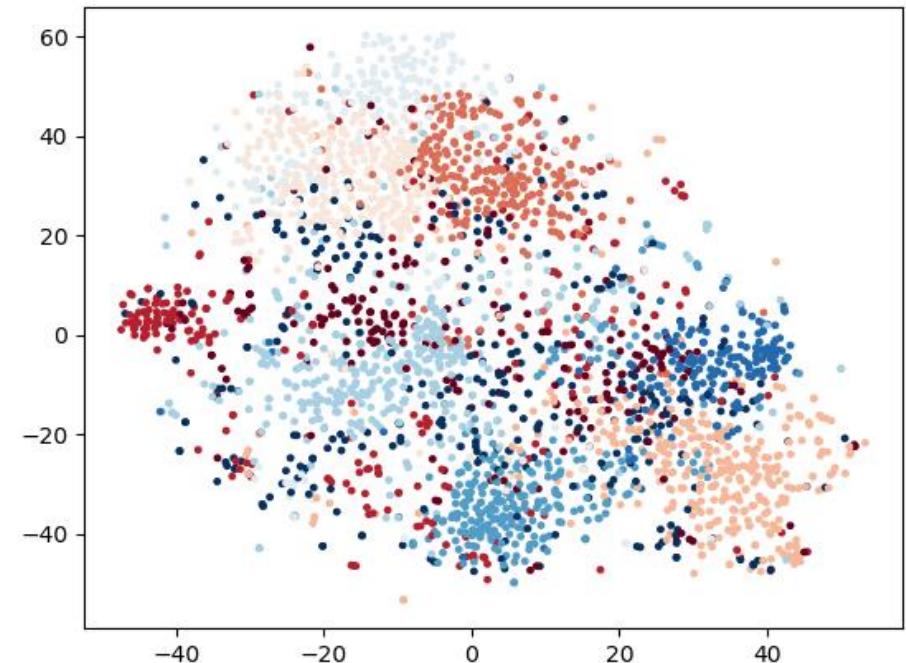


Problema

Fatti questi passaggi non ci resta che tornare ad interpretare i cluster, andando a vedere le frasi che li compongono a cosa fanno riferimento.

Il cluster rosso a sinistra sono tutte richieste per non aver ricevuto una mail di conferma di una promozione.

Il cliente non si era accorto che il server delle mail di promozione si era spento, segnalato il problema hanno risolto una trentina di richieste al mese.



Clustering

Normalmente è molto più facile ottenere dati senza label che labellati, per cui in alcuni casi potremmo fare prima clustering, prendere degli esempi da un cluster per valutare che appartengano tutti alla stessa categoria ed eventualmente etichettare tutto il cluster.

È interessante anche provare a capire perché, o meglio quali sono le caratteristiche che accumunano gli esempi dei gruppi identificati.

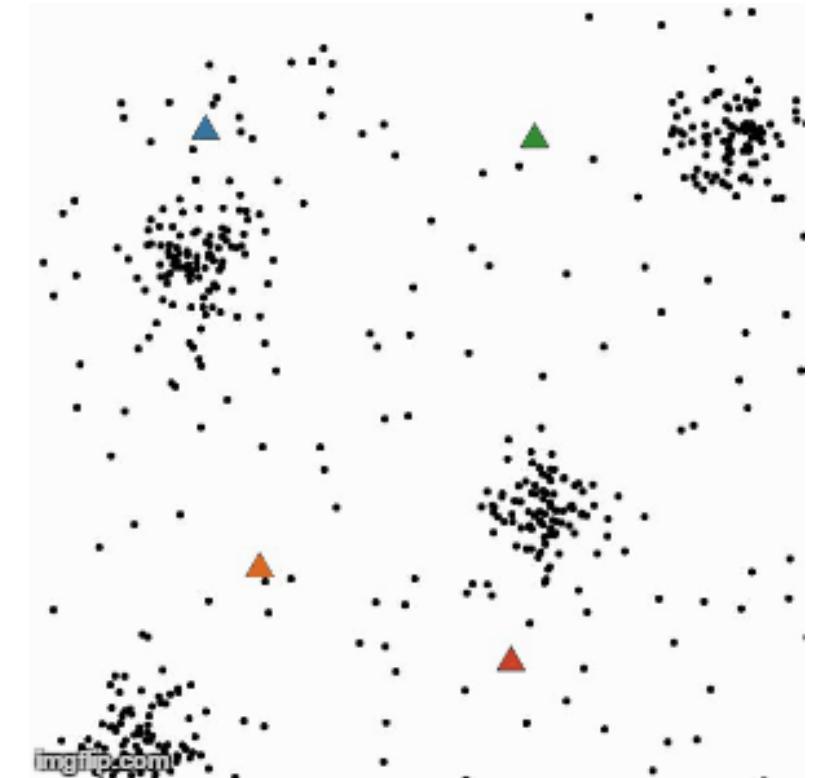
Infine possiamo utilizzare l'informazione del cluster di appartenenza per definire policy, ad esempio avremo gruppi di item che vendono costantemente nel tempo, per cui ha senso fare magazzino, e item che invece hanno un'alta volatilità, per cui ordineremo gli stock solo all'ordine del cliente.

K-means

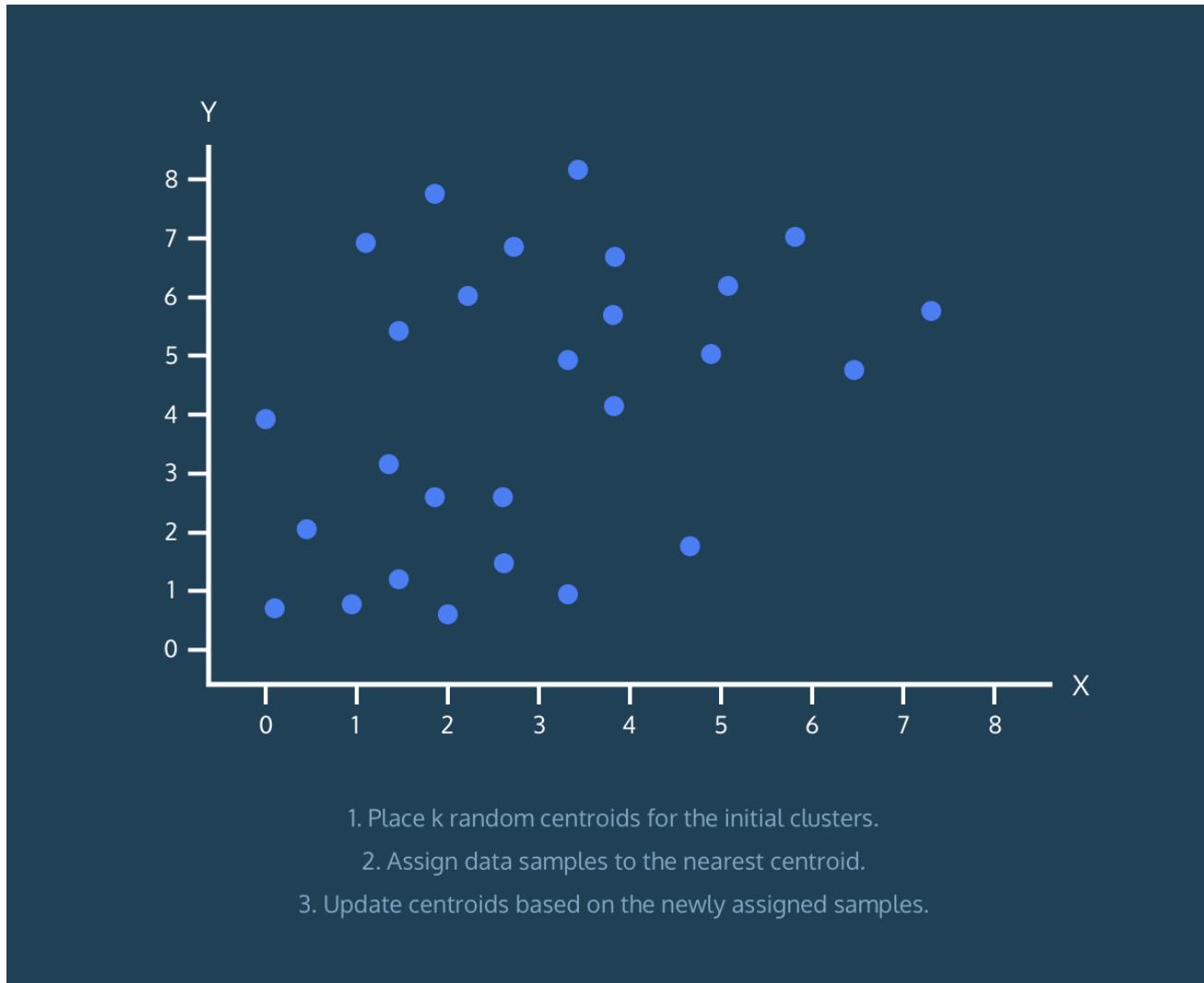
Il K-means è la tecnica più semplice di clustering.

Dobbiamo sapere a prescindere quanti cluster stiamo cercando nei dati (k).

L'obiettivo della tecnica è minimizzare la somma totale dei quadrati delle distanze di ogni suo punto dal proprio centroide.



K-means



K-means

Come abbiamo visto nelle immagini

- Definiamo un numero di cluster
- Lanciamo random K centroidi
- Iterativamente
 - Associamo ogni punto al centroide più vicino
 - Ricalcoliamo la posizione del centroide come media delle distanze dai suoi punti

Quando non muoviamo più nessun punto abbiamo terminato la ricerca.

K-means è un algoritmo greedy che converge ad un minimo locale.

Possiamo eseguire più volte la ricerca dei cluster con centroidi random per migliorarne le performance.

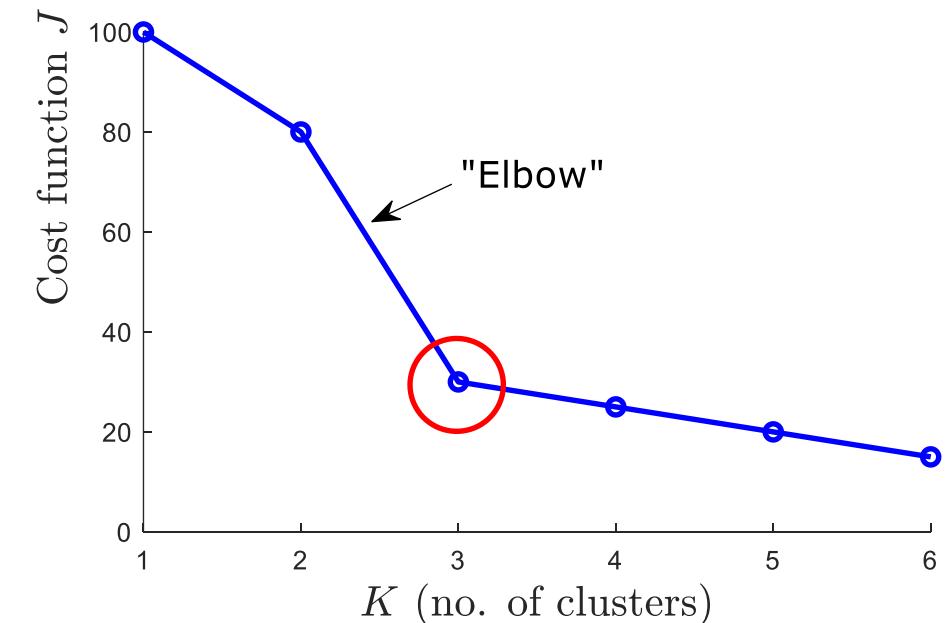
Numero di cluster

Una delle scelte fondamentali è quella del numero di cluster.

Trattando problemi unsupervised non sappiamo a priori quanti gruppi potrebbero esserci, per questo ci possiamo affidare a

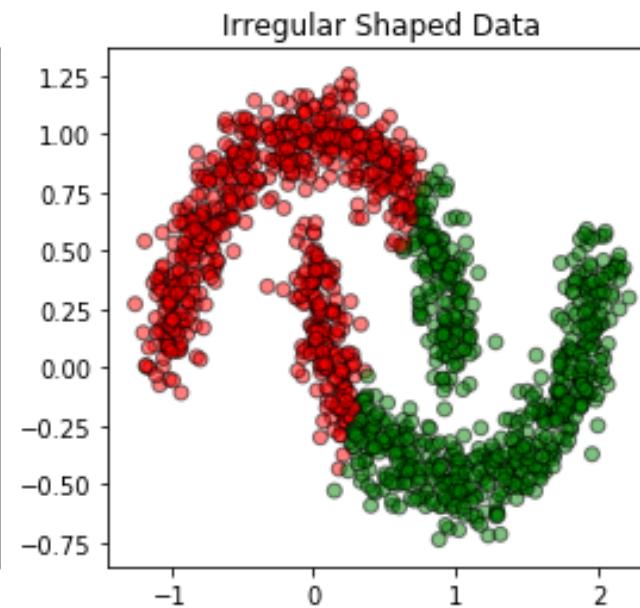
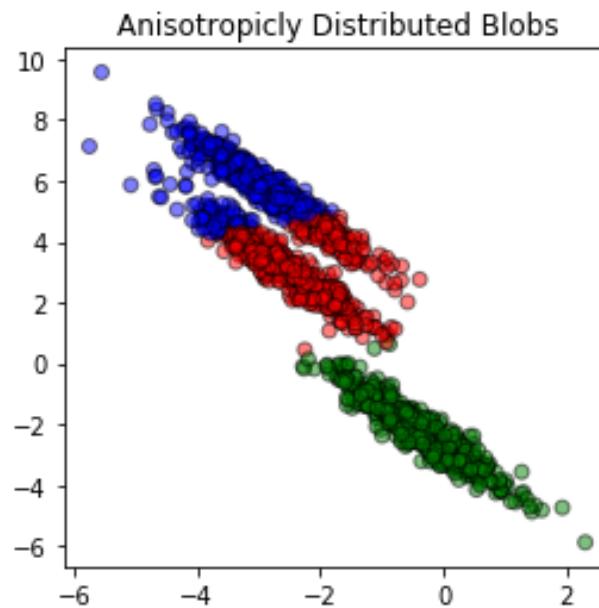
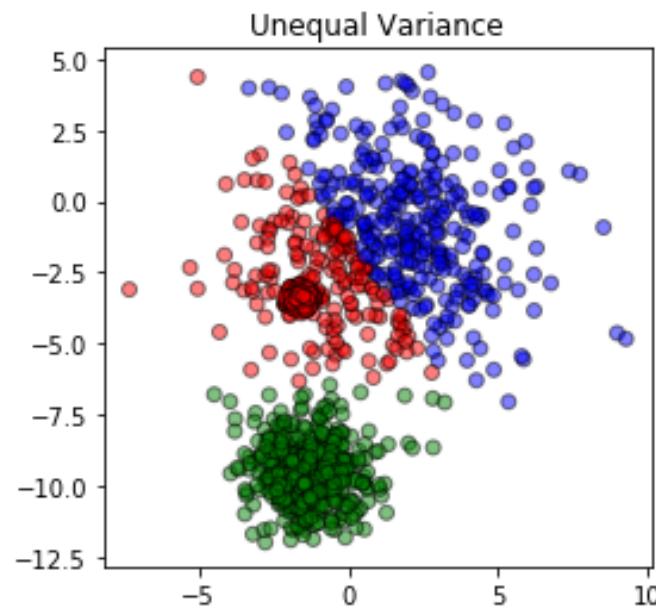
- Conoscenza sui dati e sul problema
- Visualizzazione dei dati (soprattutto nei casi con cardinalità bassa)
- Regola del «Elbow», ovvero osservare la variazione che aggiungere altri cluster comporta sulla divisione del problema.

Un esempio è in quante taglie dividere le dimensioni delle magliette. Possiamo clusterizzare le dimensioni dei nostri clienti e vedere dove c'è una variazione per cui aggiungere una taglia ha meno senso.



Errori del k-means

Il k-means è soggetto a numerosi tipi di errore e tende a dare a ciascun cluster lo stesso numero di punti. Inoltre randomizzando lo stato iniziale è molto soggetto a risultati diversi finendo spesso in minimi locali molto poco espressivi.



Esercitazione

Proviamo ad eseguire la nostra prima tecnica di clustering su un dataset e vedere se estrae qualche pattern utile

https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz_yT9_quAuBqd0Fm0?usp=sharing



08 – Unsupervised Learning

Daniele Gamba

2022/2023

Top Serie TV – Corso ML 2023

Serie prese dalla Top 250 IMDB per numero di recensioni.

Rispondete che proviamo a vedere le serie più simili ai vostri gusti

<https://forms.gle/Mz4cqNSQ2kTge1sw9>

Rank & Title	IMDb Rating	Your Rating
 13. Il trono di spade (2011)	9,1	
 2. Breaking Bad - Reazioni collaterali (2008)	9,4	
 99. Stranger Things (2016)	8,6	
 50. Friends (1994)	8,8	
 20. Sherlock (2010)	9,0	
 5. Chernobyl (2019)	9,3	
 126. Dexter (2006)	8,6	
 28. The Office (2005)	8,9	



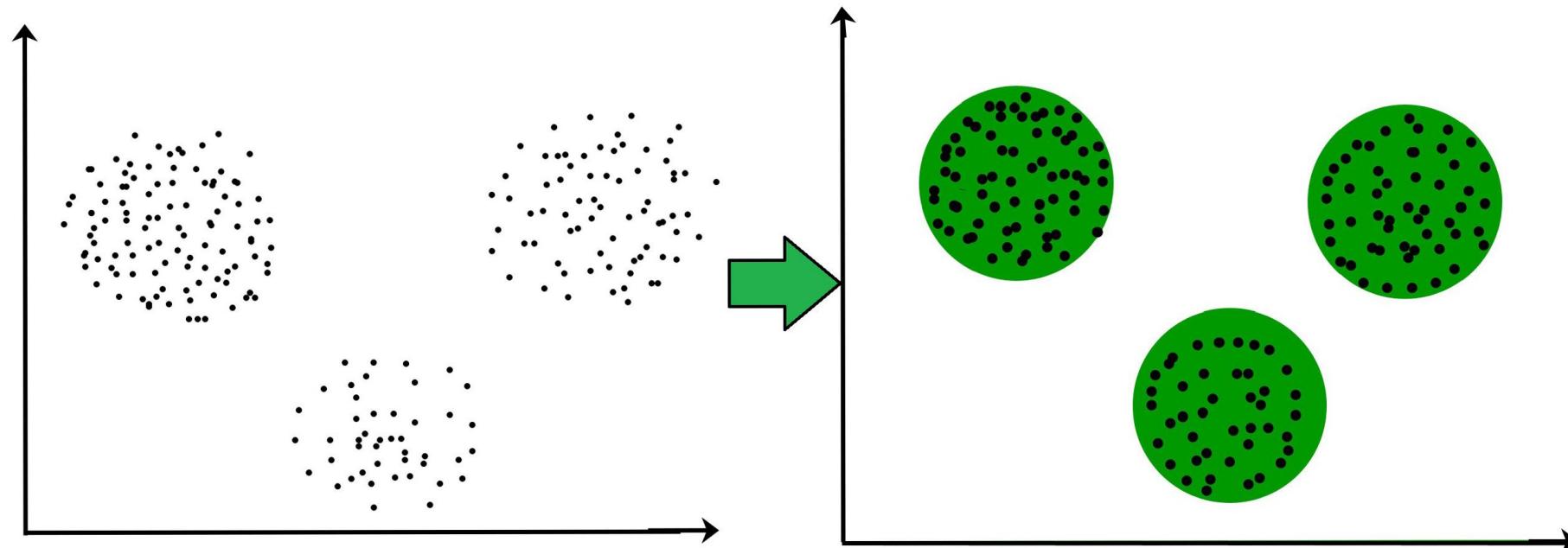
Lezione precedente

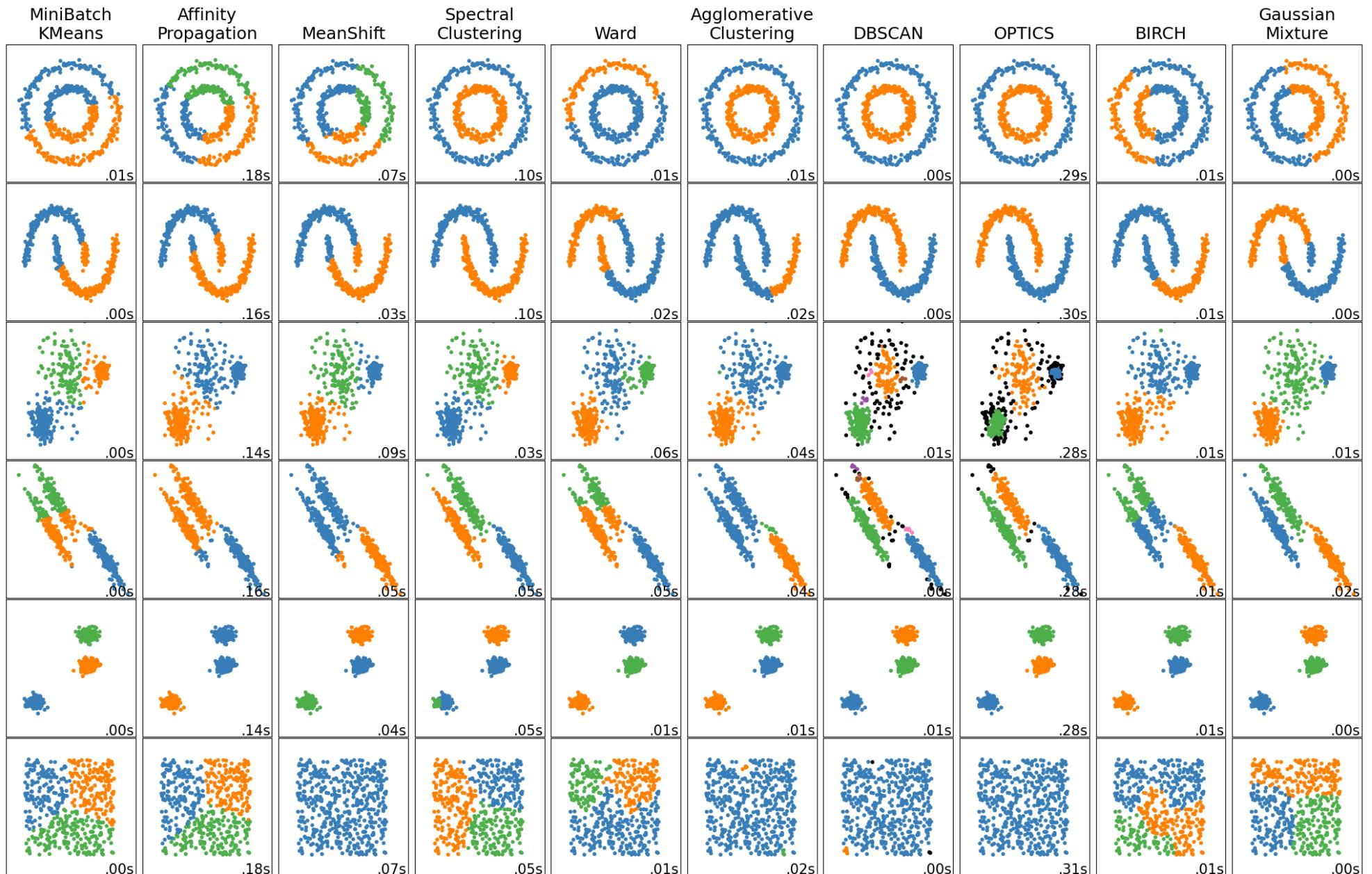
Abbiamo visto

- Dimensionality reduction
- Introduzione al clustering
- k-means

Clustering

Il clustering è un ambito dell'unsupervised learning in cui andiamo a trovare «cluster» ovvero gruppi di punti simili tra loro all'interno dei dati senza avere una label.

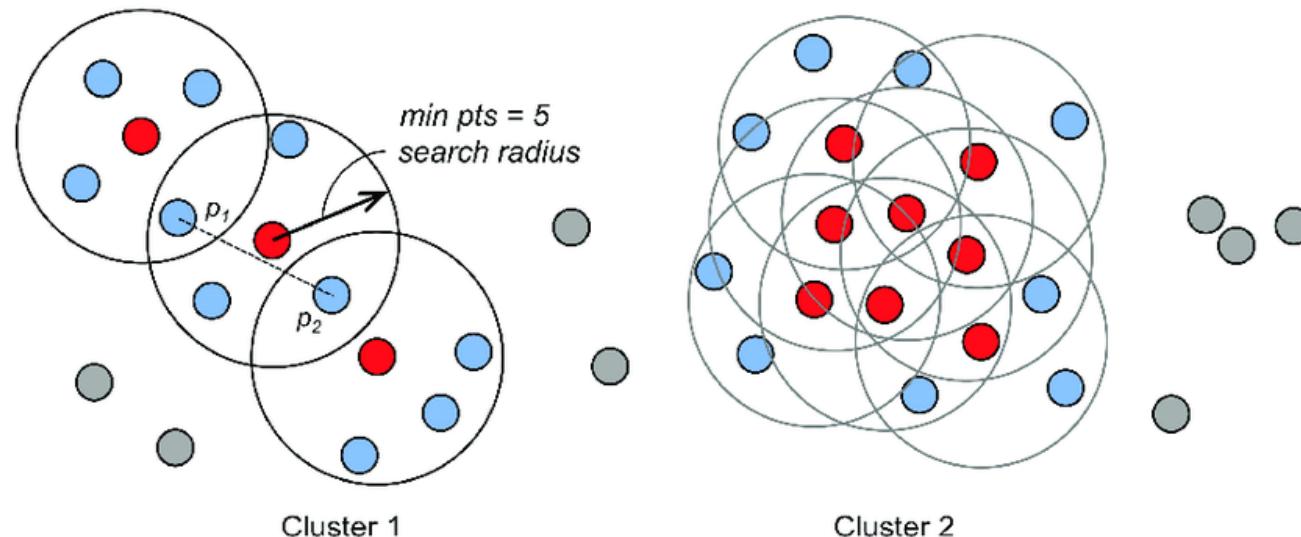




DBSCAN

DBSCAN è una tecnica di clustering che si basa sul concetto di densità.

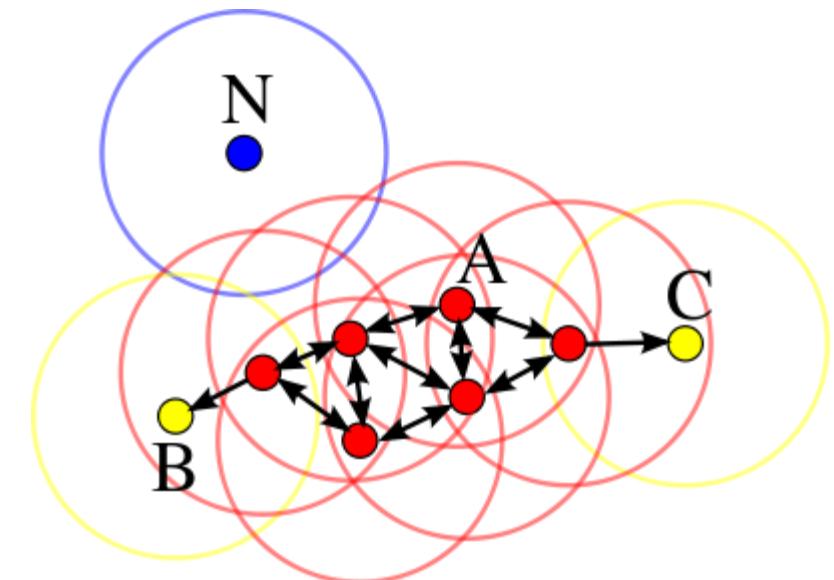
Preso un punto random, espandiamo il cluster a tutti i punti che hanno una distanza minore di un ε fissato.



DBSCAN

Al termine del primo cluster, ci troveremo con N punti appartenenti ad un cluster ed altri all'esterno.

Per ciascun punto valutiamo il numero di altri punti all'interno del suo raggio. Se sono sopra ad una certa soglia lo considereremo parte di un nuovo cluster e inizieremo ad espanderlo, se il punto è sparso con niente o pochi punti nel suo intorno viene classificato come noise.



DBSCAN - Limiti

Il limite più grosso di DBSCAN è che il raggio fisso ne limita fortemente le capacità nel caso in cui ci sono punti molto vicini.

Non riesce quindi ad interpretare la *variazione* di densità fintanto che riesce ad espandere il cluster.

Se i cluster fossero anche solo debolmente connessi tra loro DBSCAN tenderebbe a farli appartenere tutti allo stesso cluster o richiederebbe molto tuning.

OPTICS

Ordering points to identify the clustering structure – OPTICS è un algoritmo che vuole superare i limiti di DBSCAN andando a pesare le variazioni di densità.

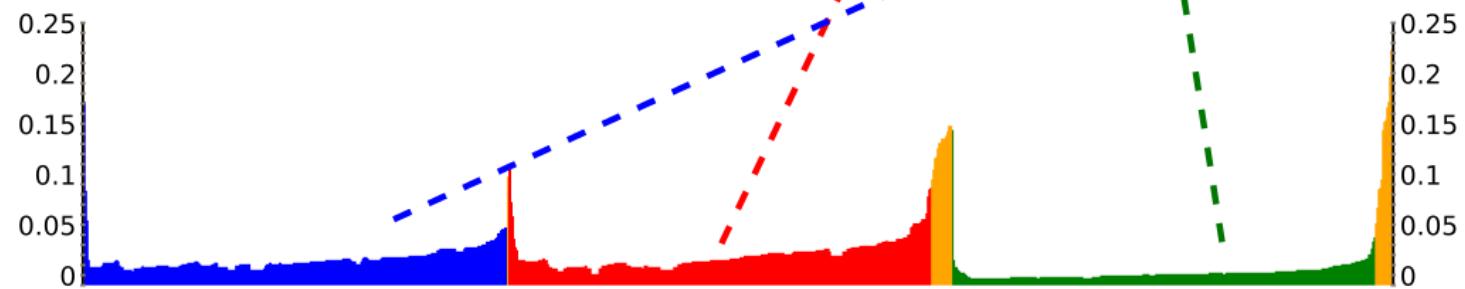
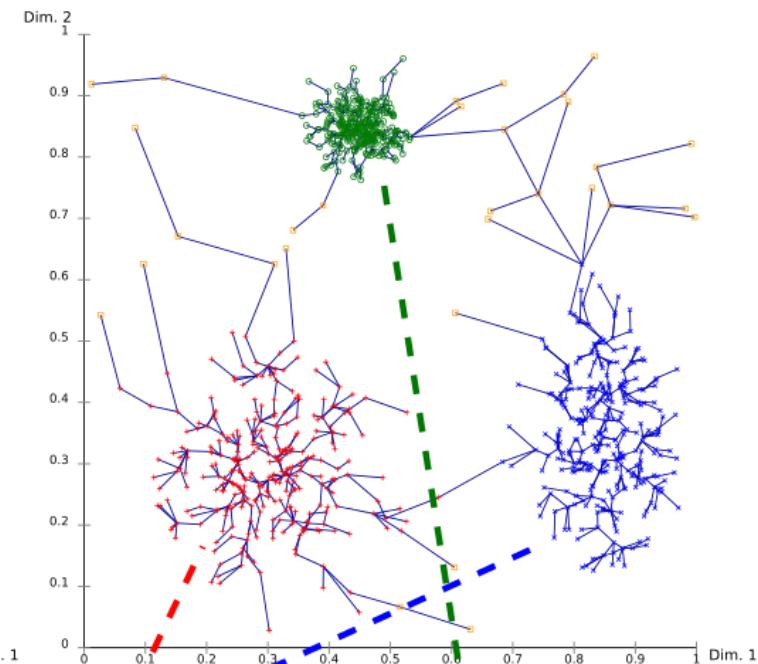
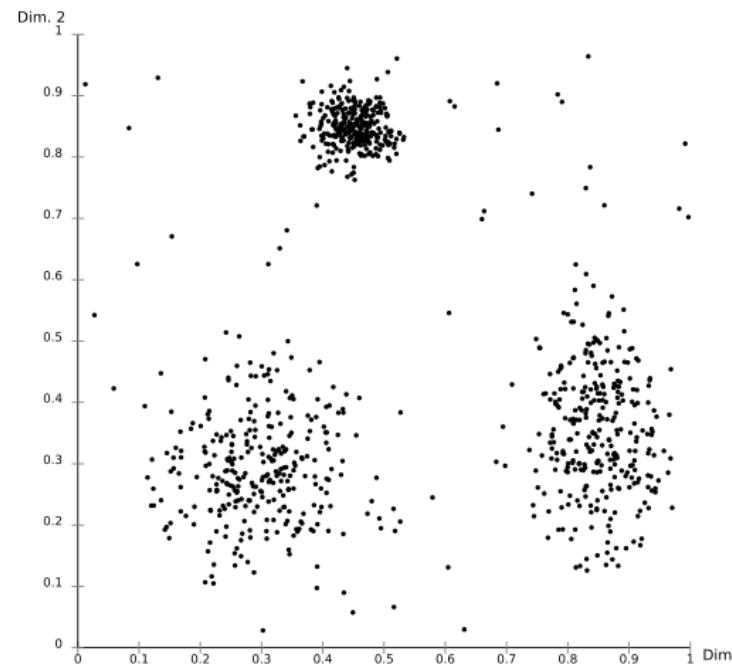
I punti vengono ordinati in modo che i punti più vicini risultino vicini nell'ordinamento.

Anche in questo caso c'è una ε che descrive la massima distanza da considerare ed il numero minimo di punti per formare un cluster.

A ciascun punto è anche assegnato una metrica di «raggiungibilità» dal centro del cluster più vicino.

Andando a misurare la raggiungibilità di ogni punto possiamo identificare i cluster in funzione della loro densità. Le raggiungibilità si distribuiranno come «vallate» nel dendogramma.

OPTICS



Hierarchical clustering

In molti problemi, come quello delle richieste di assistenza, è frequente trovare dati appartenenti ad aree diverse (meccanica, qualità, commerciale, ..) con un dettaglio maggiore per ciascuna di queste (problemi alla pompa, problemi alla resistenza, problemi meccanici, ...).

Questo perché i dati si strutturano in aree tematiche e problematiche singole con una **gerarchia**.

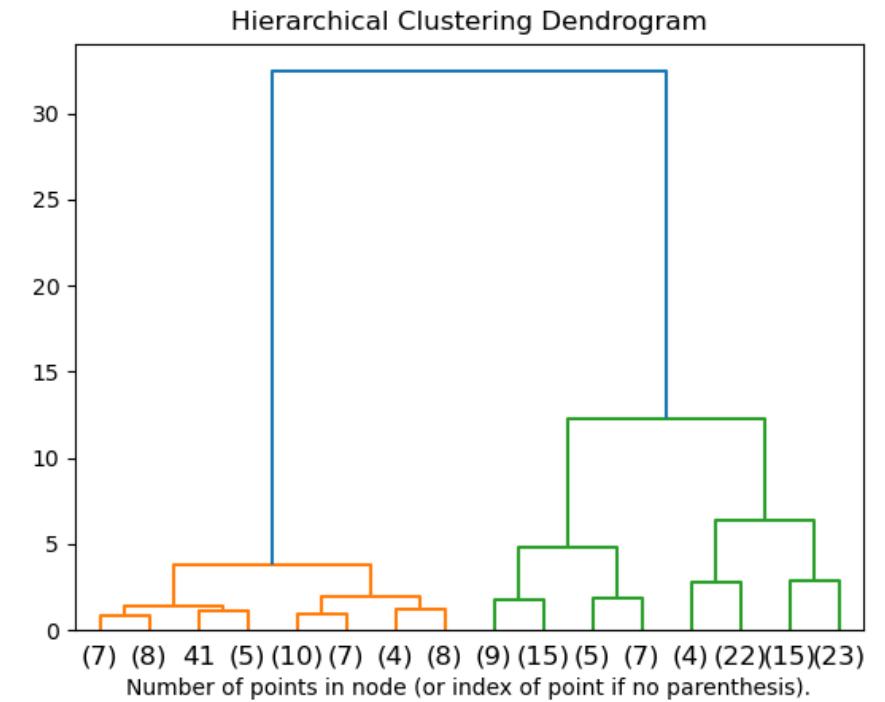
Il clustering gerarchico è una tecnica che ci consente di costruire un albero di relazioni seguendo un approccio

- Bottom-up, andando ad aggregare iterativamente cluster simili
- Top-down, andando a dividere i cluster in sottocluster più densi

Hierarchical clustering

L'hierarchical ha diverse metriche con cui procedere a dividere o ad aggregare i cluster.

Una di queste, chiamata **ward**, minimizza la somma delle distanze al quadrato all'interno di tutti i cluster, cercando di minimizzare la varianza all'interno del cluster.



Esercitazione - Telco

Siamo una compagnia telefonica e vogliamo capire di più come mai i nostri clienti ci abbandonano.

Abbiamo raccolto delle informazioni sui nostri clienti che ci hanno abbandonato nell'ultimo mese tra cui i servizi che avevano attivi, le informazioni demografiche, quanto hanno speso, ecc.

Cerchiamo di capire come si distribuiscono i nostri clienti persi e raggruppiamoli in modo da targettarli con campagne ad-hoc.

https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz_yT9_quAuBqd0Fm0?usp=sharing

Recommender systems

I recommender systems sono algoritmi che provano a suggerire ad un utente un nuovo item nella speranza che gli piaccia.

Gli algoritmi di raccomandazione sono ovunque, Netflix, Amazon, YouTube, Spotify, ecc.

Rappresentano una parte critica del fatturato, sia per suggerire altri acquisti (Amazon, ..) sia per fornire una miglior qualità del servizio (YouTube, Spotify, Netflix).

Alcuni algoritmi di raccomandazione possono esser visti sia come problemi **unsupervised** (trovo utenti più simili a me) che come problemi di **regressione** (quale potrebbe essere il rating che darò a quel film).

How likely are you to recommend Windows 10 to a friend or colleague?

<input checked="" type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
Not at all likely			Extremely likely	

Please explain why you gave this score.

I need you to understand that people don't have conversations where they randomly recommend operating systems to one another

Recommender systems



Movies



Online Videos



Music



Books & products



Softwares



Page Rank



tripadvisor®

Restaurants



Accommodation



Dating



Friends

Netflix Challenge

Dal 2006 al 2009 Netflix ha lanciato una competizione per un algoritmo di previsione dei ratings.

Offriva un premio da 1 milione di dollari a chi avesse migliorato lo stato dell'arte (Cinematch) del 10%. Parteciparono 20.000 teams da 150 paesi.

Il 26 Giugno 2009 il team BellKor's Pragmatic Chaos vince con un miglioramento del 10.05%.

The screenshot shows the Netflix Prize Leaderboard page. At the top, it displays "Leaderboard 10.05%" and a note "Display top 20 leaders." A yellow arrow points to the "% Improvement" column in the table below. The table has columns for Rank, Team Name, Best Score, % Improvement, and Last Submit Time. The top entry is for BellKor's Pragmatic Chaos with a score of 0.8558 and an improvement of 10.05%. The Grand Prize threshold is listed as RMSE <= 0.8563. Other teams listed are PragmaticTheory and BellKor In BigChaos.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	BellKor's Pragmatic Chaos	0.8558	10.05	2009-06-26 18:42:37
Grand Prize - RMSE <= 0.8563				
2	PragmaticTheory	0.8582	9.80	2009-06-25 22:15:51
3	BellKor In BigChaos	0.8590	9.71	2009-05-13 08:14:09

Content Based vs Collaborative Filtering

Due delle tecniche classiche di recommender systems si basano su due approcci differenti

Content Based

Raccomandare item che sono simili a quelli che l'utente ha apprezzato in passato

Collaborative Filtering

Raccomandare item che utenti simili al target hanno apprezzato

Il secondo approccio ha preso maggior piede in quanto

- Item che ho già apprezzato potrebbero non servirmi più (es. ho già acquistato su Amazon il telefono che mi piaceva, non ne comprerò immediatamente un altro)
- Utenti simili potrebbero aver esplorato nuovi item di cui non conoscevo l'esistenza

Collaborative filtering

Dietro al collaborative filtering c'è l'idea che l'informazione sui gusti degli utenti sia diffusa.

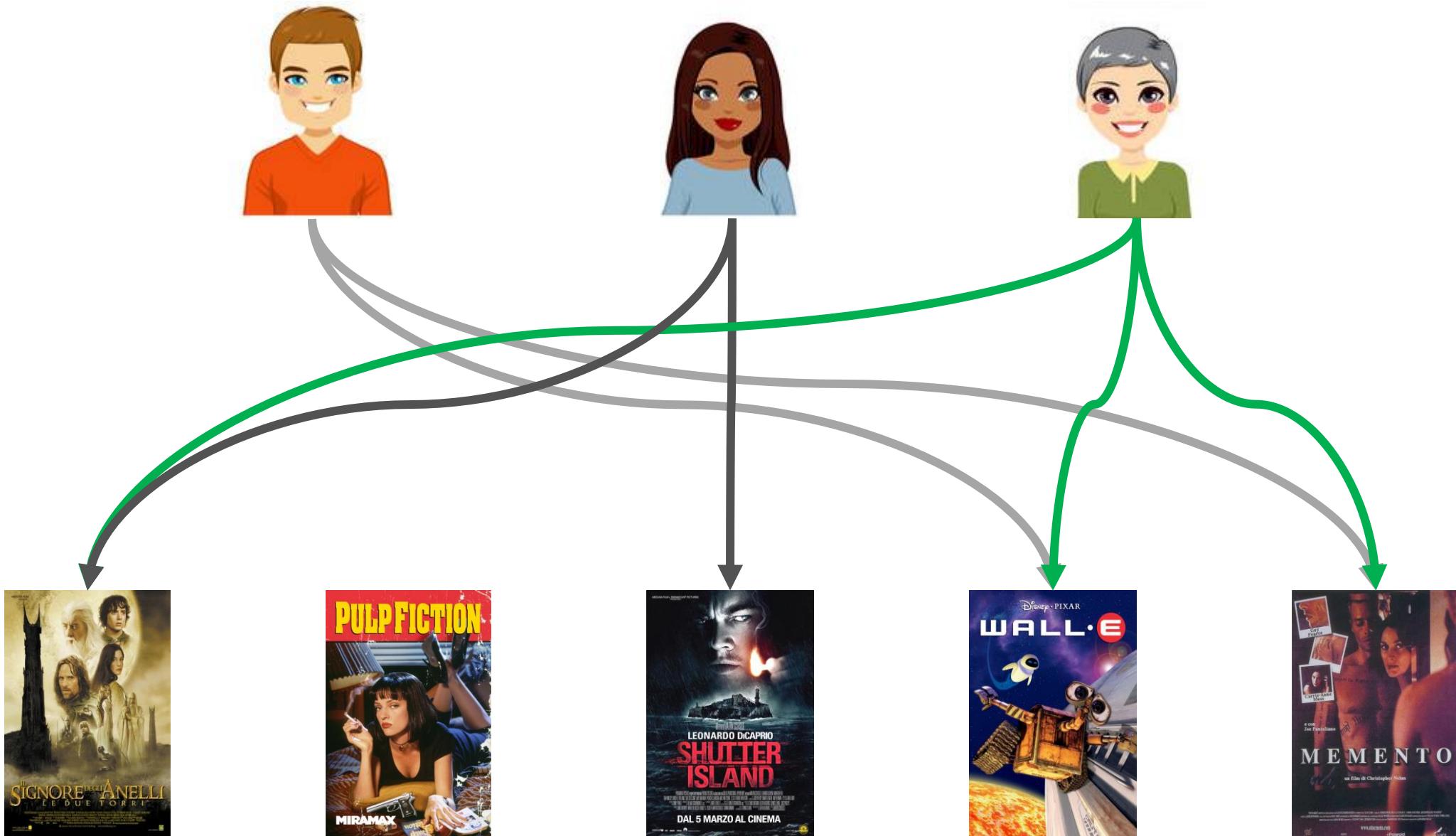
Utenti diversi, che esplorano maggiormente, tenderanno ad esplorare e valutare diversamente i diversi item, consentiranno di esplorare «lo spazio dei gusti» anche dei nuovi utenti meno temerari.

Utenti simili, che condividono gli stessi gusti, consentiranno di riproporre agli altri nuovi item.

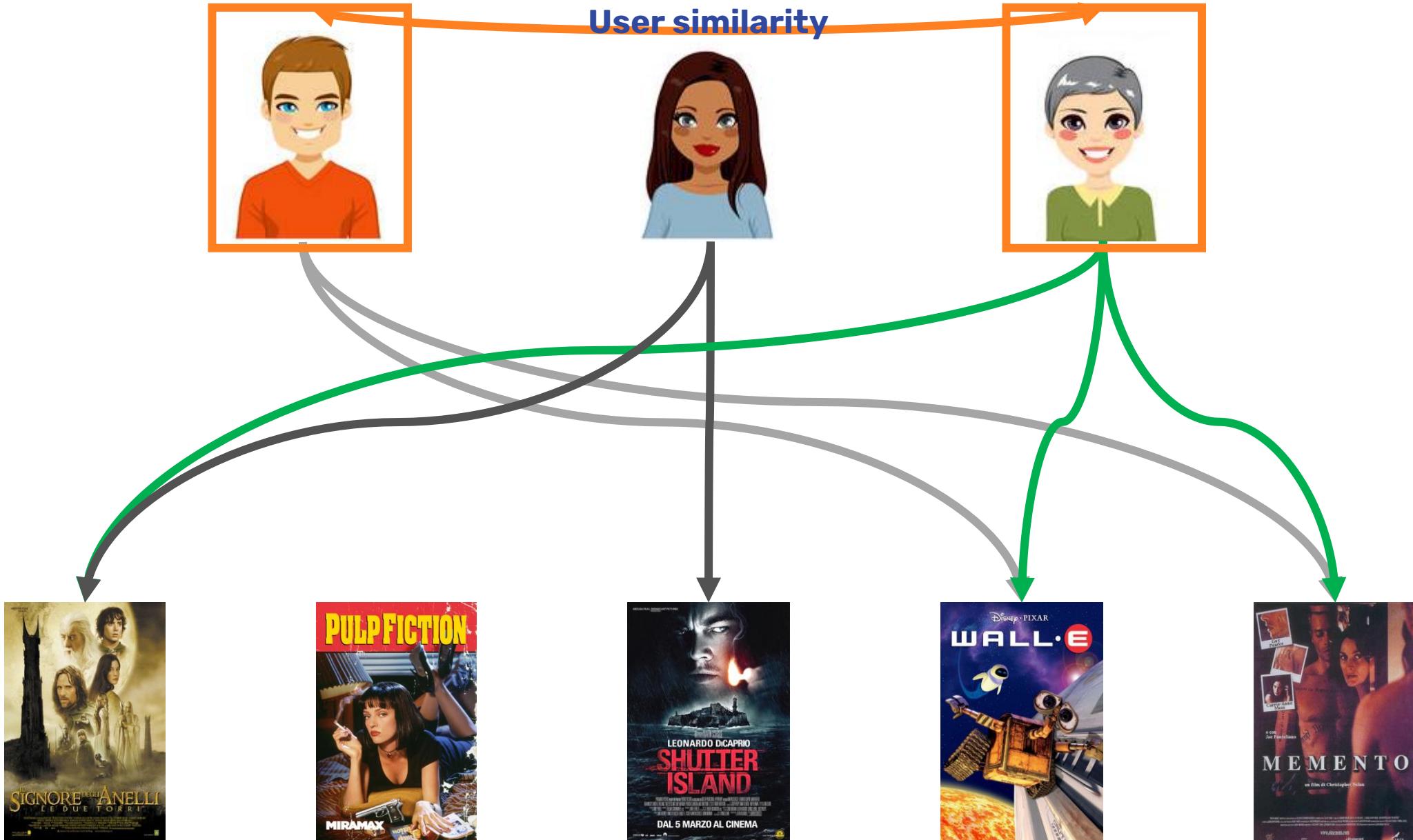
Ci sono due approcci identificabili

- Item-based
- User-based

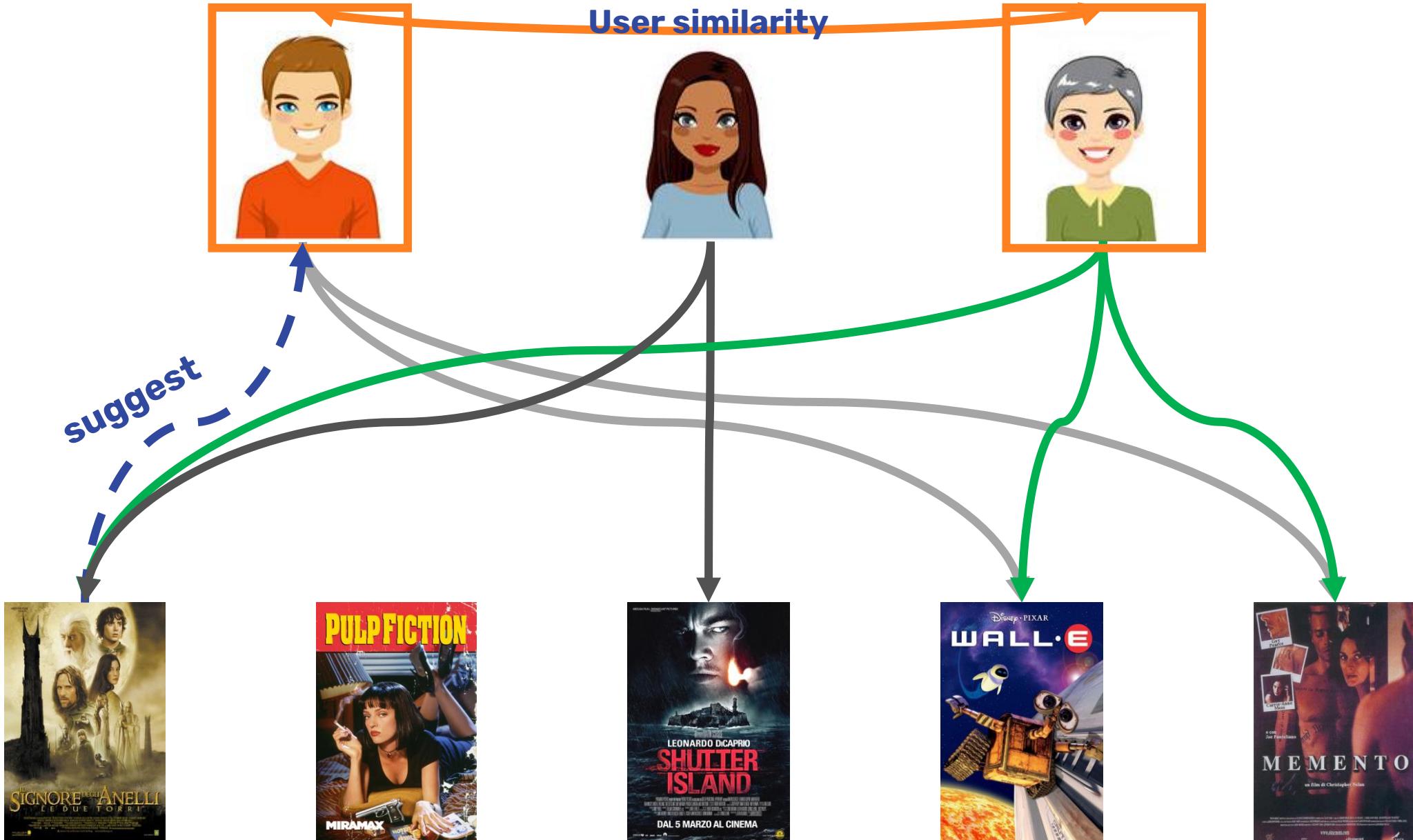
Collaborative Filtering – User based



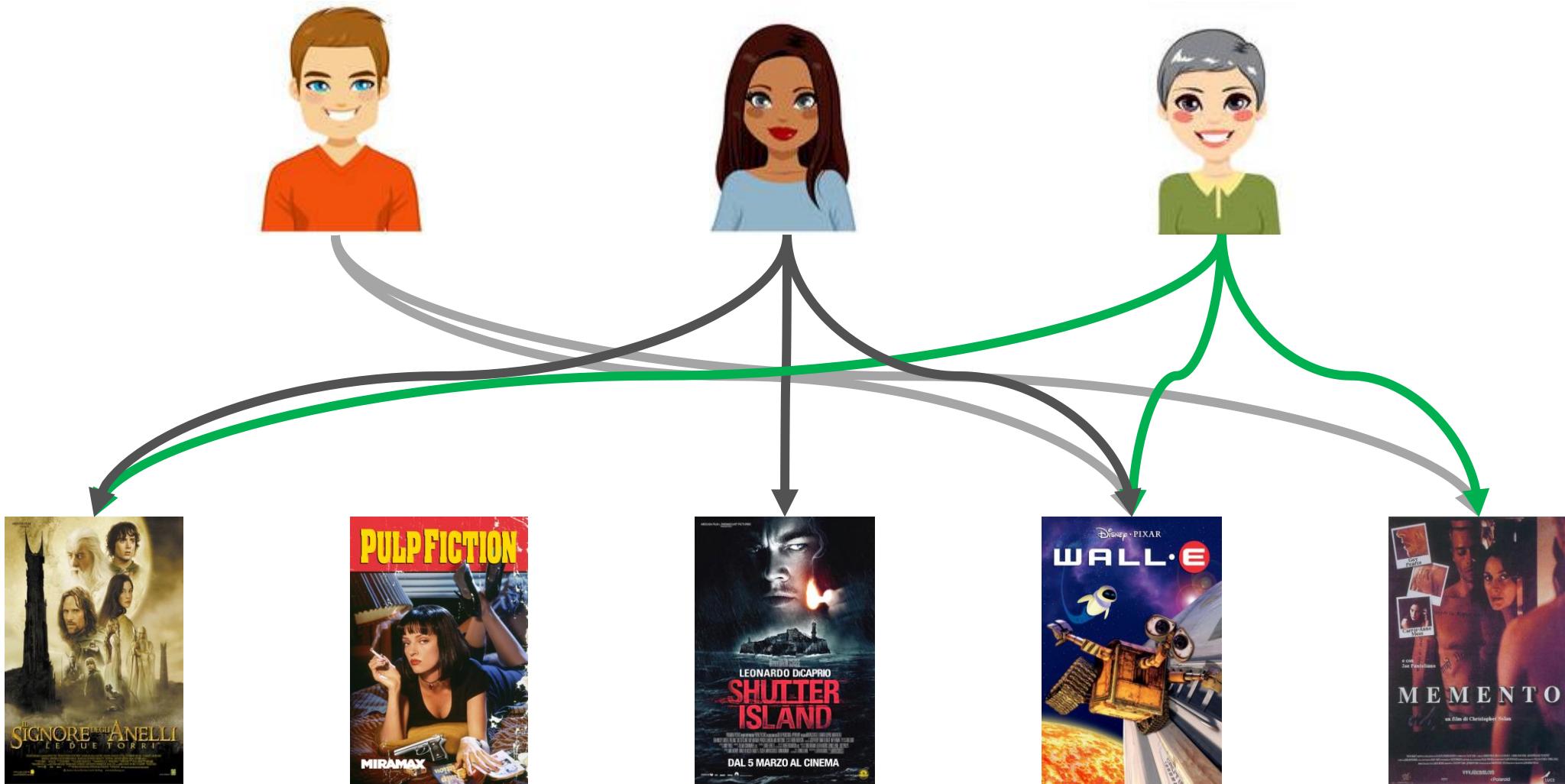
Collaborative Filtering – User based



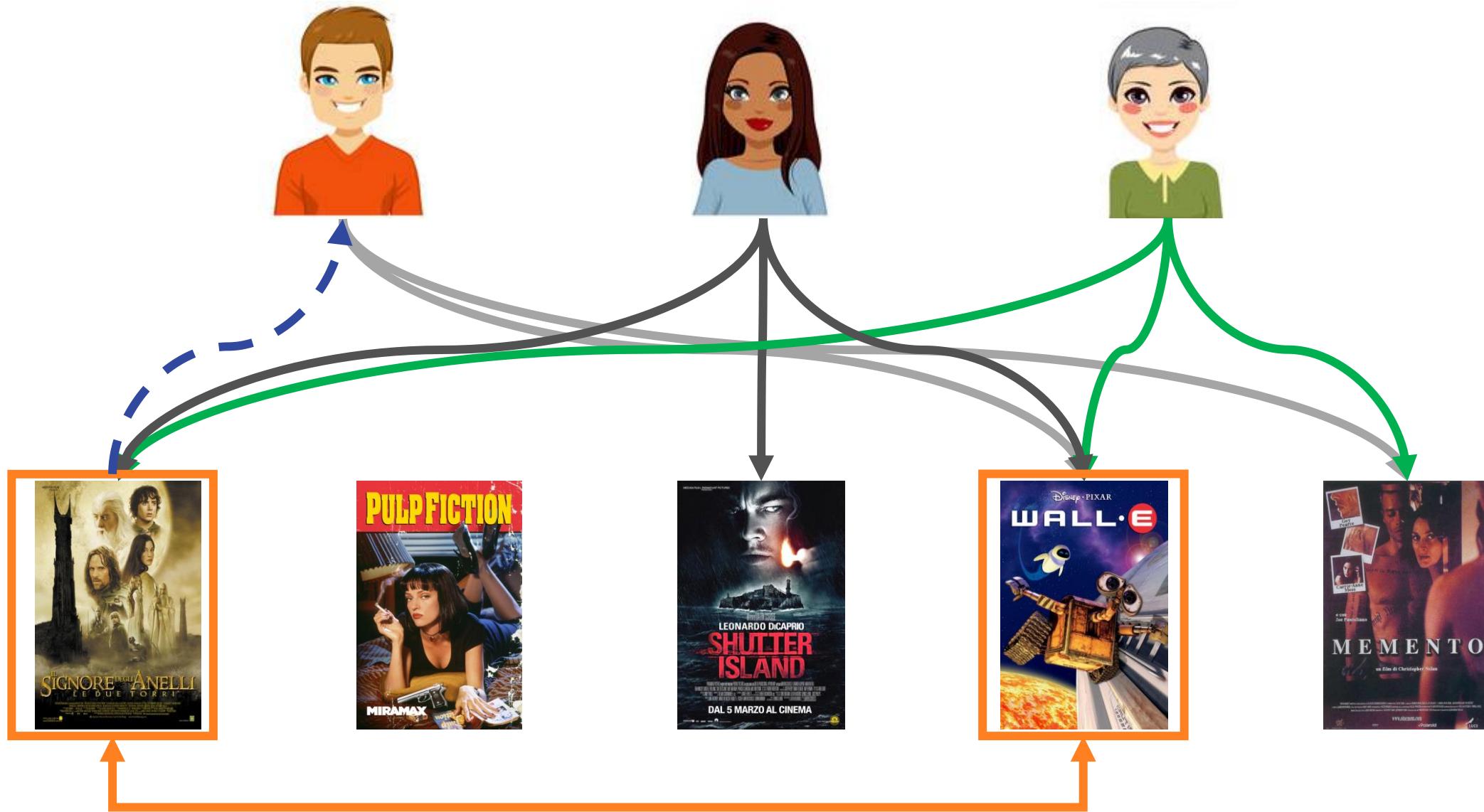
Collaborative Filtering – User based



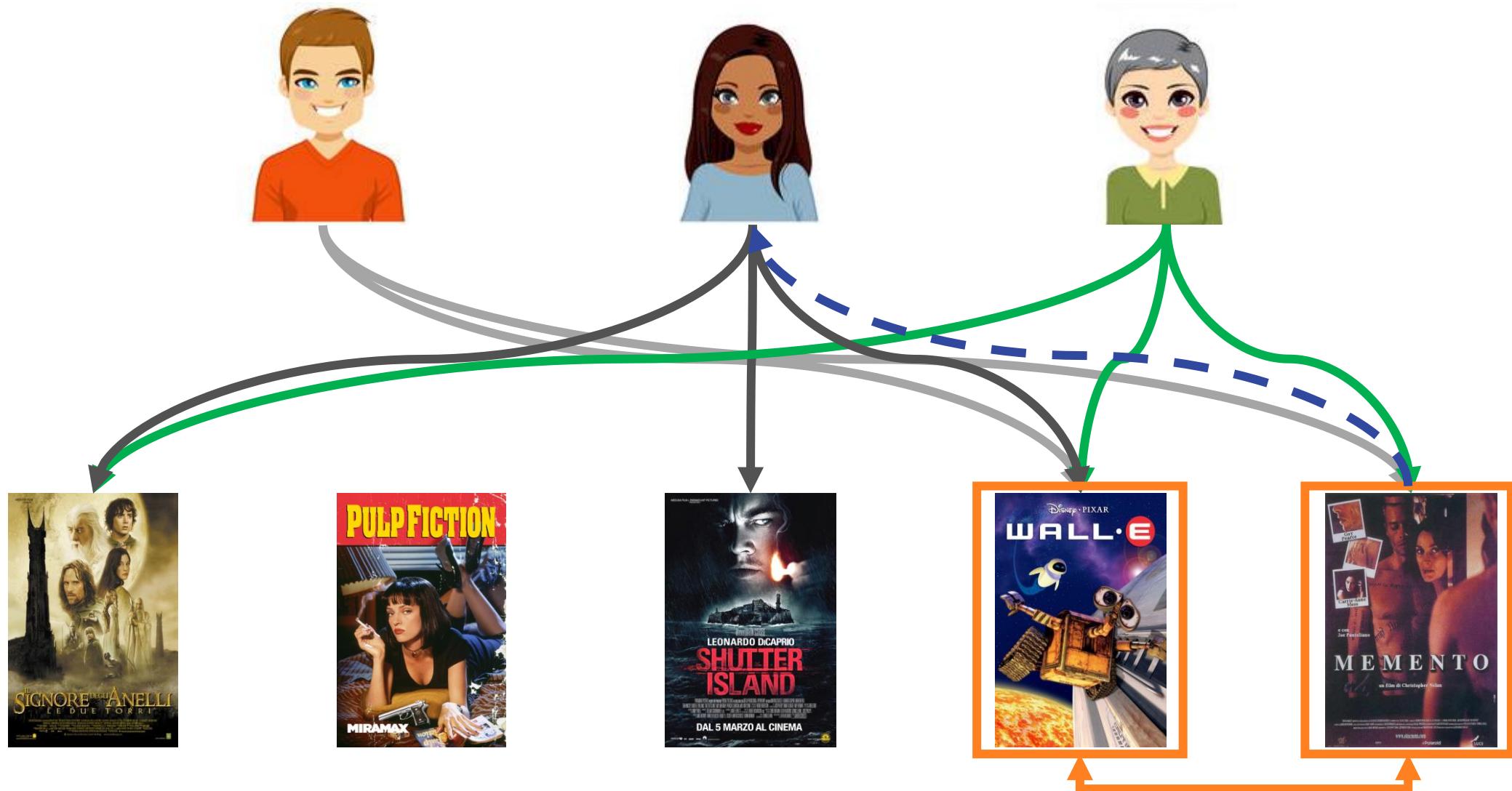
Collaborative Filtering – User based



Collaborative Filtering – User based



Collaborative Filtering – User based



User based

Come visto nell'esempio, in questo caso riproponremo agli utenti degli item selezionandoli tra quelli apprezzati da un campionamento degli altri utenti con simili.

È una tecnica che possiamo vedere come

- Parametrica, se cerchiamo di definire i gusti di una tipologia di persone
- Non-parametrica, se confrontiamo direttamente i gusti di altri utenti

Come metriche di distanza possiamo usare

- Euclidean Distance
- Pearson's Correlation
- Cosine Similarity
- Jaccard Distance

Item Based

A differenza dell'user-based, item based prevede il confronto tra gli item.

Andremo a calcolare una metrica di distanza tra tutti gli item in modo da trovare i film simili tra loro.

Suggeriremo all'utente quelli simili ai film a cui ha dato voti alti.

Proviamo nel dataset anche a calcolarci la similarità tra le serie sulla base dei rating.

Top Serie TV – Corso ML 2023

Serie prese dalla Top 250 IMDB per numero di recensioni.

Rispondete che proviamo a vedere le serie più simili ai vostri gusti

<https://forms.gle/Mz4cqNSQ2kTge1sw9>

Rank & Title	IMDb Rating	Your Rating
 13. Il trono di spade (2011)	9,1	
 2. Breaking Bad - Reazioni collaterali (2008)	9,4	
 99. Stranger Things (2016)	8,6	
 50. Friends (1994)	8,8	
 20. Sherlock (2010)	9,0	
 5. Chernobyl (2019)	9,3	
 126. Dexter (2006)	8,6	
 28. The Office (2005)	8,9	



Esercitazione – Collaborative filtering

Chi avrà i gusti più simili al prof?

<https://colab.research.google.com/drive/1mfTGtubZSfwIk4jhQcAGXnffnFUK1HZY?usp=sharing>

Content-based filtering

I criteri di filtraggio content-based si basano su caratteristiche dell'item stesso.

Possiamo sfruttare features o caratteristiche come la categoria, il regista, l'attore, ecc. per poi fornire suggerimenti basati su questi.

- Directed by **Christopher Nolan**  **Suggest The Prestige**
- Inception is a **psycho thriller**  **Suggest Shutter Island**
- The main actor is **Di Caprio**  **Suggest Titanic**

Recap

Abbiamo finito di vedere una parte tradizionale del Machine Learning.

Procederemo nella seconda parte del corso a riaffrontare i problemi di regressione, classificazione, dimensionality reduction, anomaly detection, etc. tramite deep learning.



09 – Neural Networks

Daniele Gamba

2022/2023

Lezione precedente

Abbiamo finito di vedere una carrellata di tecniche classiche di Machine Learning per problemi di

- Supervised
 - Regression
 - Classification
- Unsupervised
 - Dimensionality reduction
 - Recommendation

Neural networks

Nel 1943 era stata ipotizzata la possibilità di descrivere matematicamente il funzionamento di un neurone per risolvere problemi di logica. Il fatto di poter connettere tra loro più unità semplici, neuroni, che in qualche modo emulassero il funzionamento del cervello, a creare reti neurali non è nuovo.

L'idea passò però velocemente di moda e venne ripresa solamente negli anni '80, sotto la nuova formulazione di «*connectionism*», ovvero lo studio delle connessioni.

Negli anni 90 però furono inventate altre tecniche di Machine Learning, tra cui SVM, per cui le reti neurali ripassarono in secondo piano.

Infine, negli ultimi anni, le reti neurali sono tornate di moda e sembrano la strada a risolvere buona parte dei problemi tipici del machine learning.

Neural networks

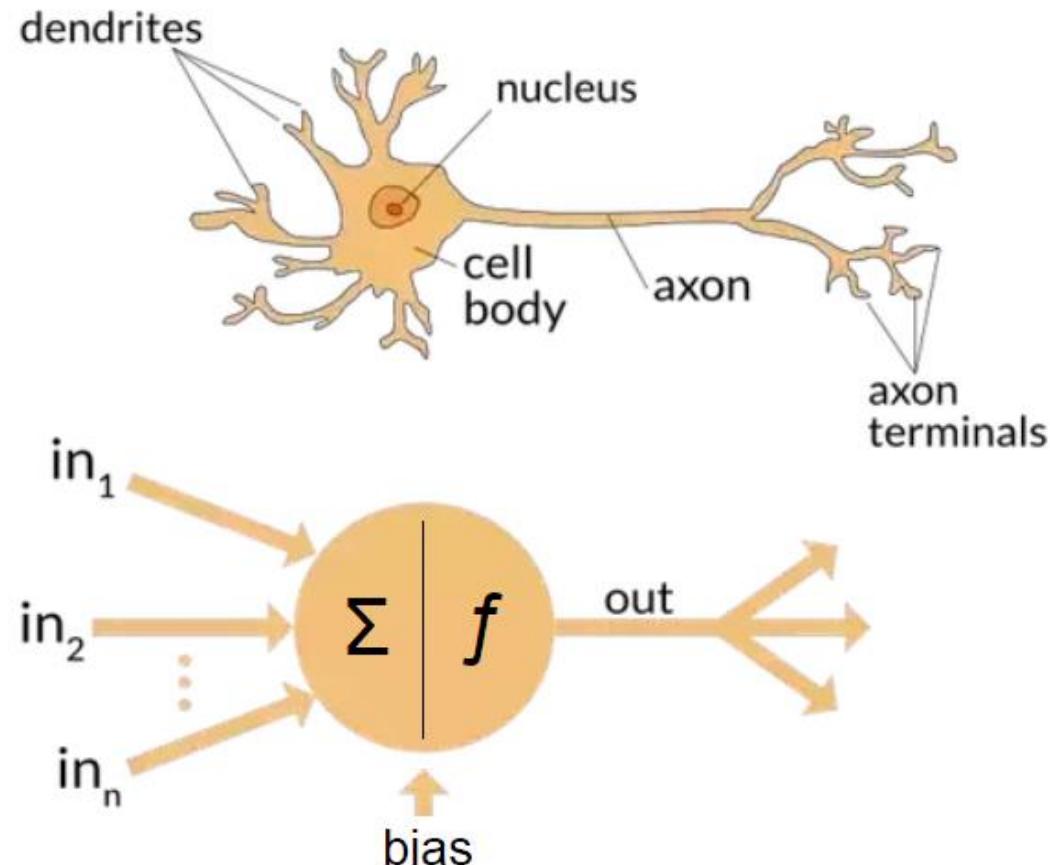
Le reti neurali sono **approssimatori universali**.

A patto di connettere tra loro abbastanza neuroni e avere abbastanza dati a disposizione, provare a possiamo risolvere qualsiasi problema.

Negli ultimi anni si sono affermate per via di

- Quantità di dati sempre più grandi, da diverse fonti, ed economici da raccogliere
- Potenza di calcolo sempre più grande (soprattutto per le GPU)
- Potenza di calcolo sempre più economica (grazie al cloud)

Neuron



Neural network

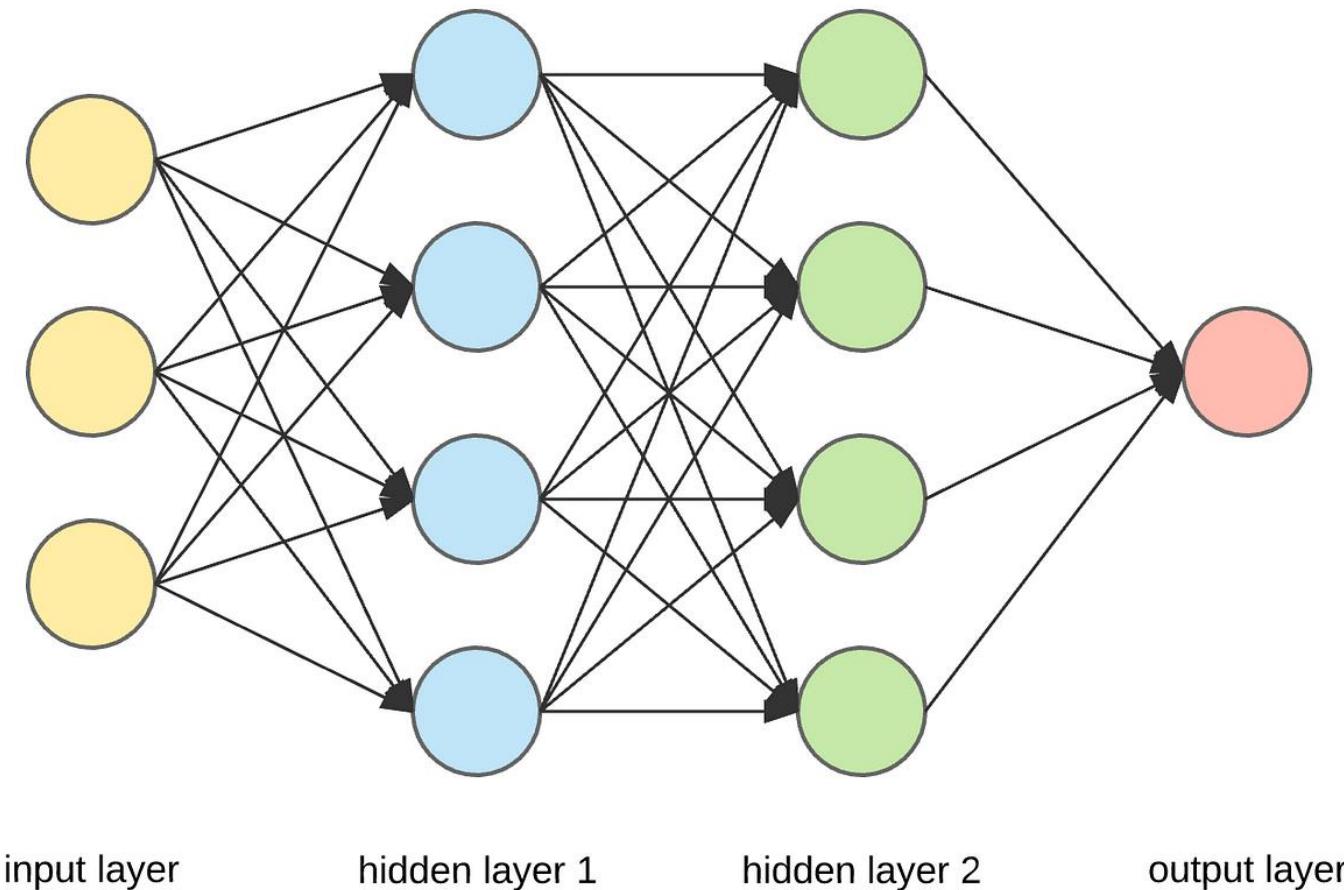
Connettendo l'uscita di un gruppo di neuroni con l'ingresso di altri neuroni otteniamo una **rete neurale**.

Ogni gruppo di neuroni sullo stesso livello è chiamato **layer**.

Il primo layer in ingresso è chiamato input layer ed è connesso con i nostri dati, x .

Le uscite dell'ultimo layer sono gli output del modello e corrispondono alle \hat{y} .

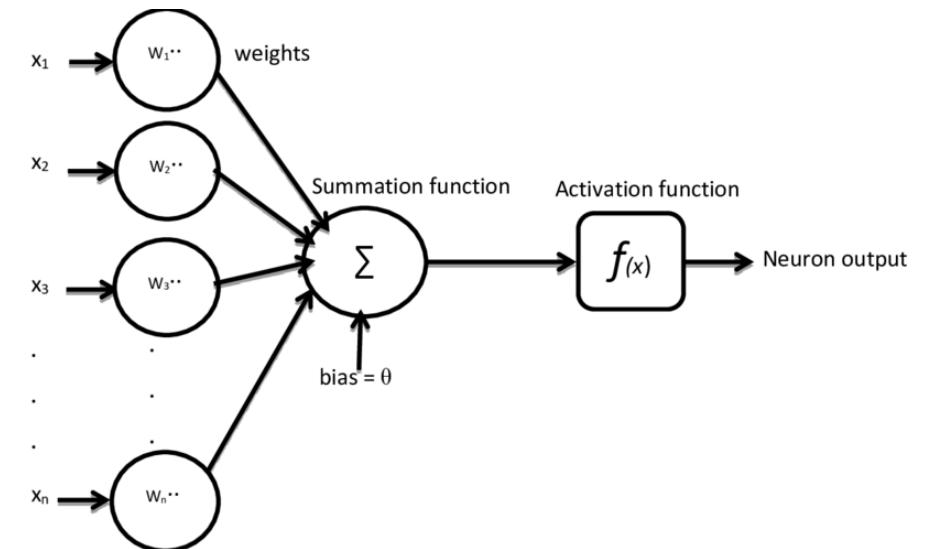
Neural network



Activation function

Ogni neurone

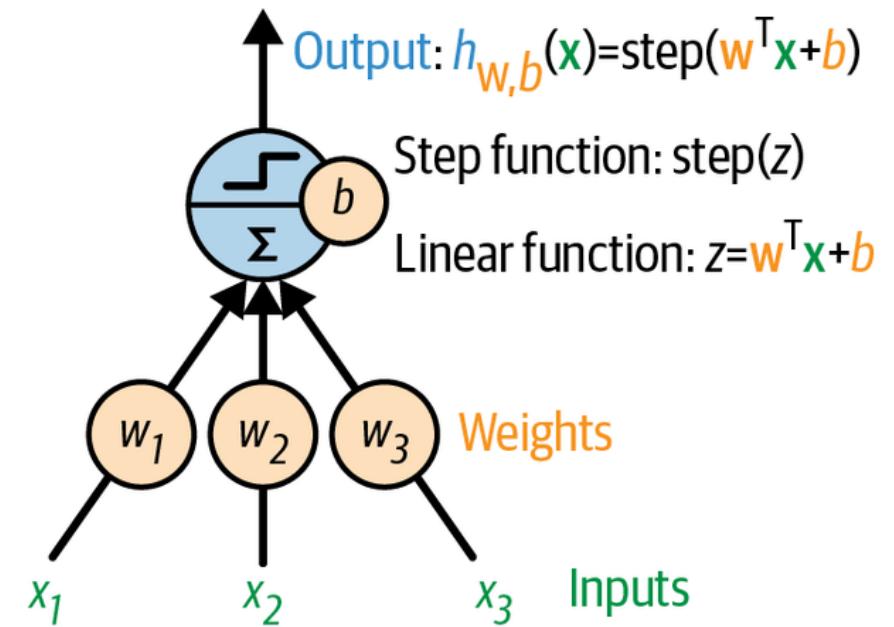
- Accetterà in ingresso degli input
- Li peserà per un weight w
- Sommerà tutti i risultati
- Applicherà una **funzione di attivazione**
- Condividerà il suo output



Activation functions

Nelle parti di ML applicata a regressione e classificazione abbiamo già visto alcune funzioni di attivazione. La più famosa nell'ambito delle reti neurali è il perceptron di Rosenblatt.

La funzione di attivazione corrispondeva alla funzione segno. Viene prodotto come output +1 o -1 semplicemente se il risultato della sommatoria interna è maggiore o minore di zero.



Activation functions

Altre funzioni di attivazione che abbiamo visto e che vengono molto usate sono

- La funzione lineare
- Il sigmoide
- La tangente iperbolica
- ReLU
- ...

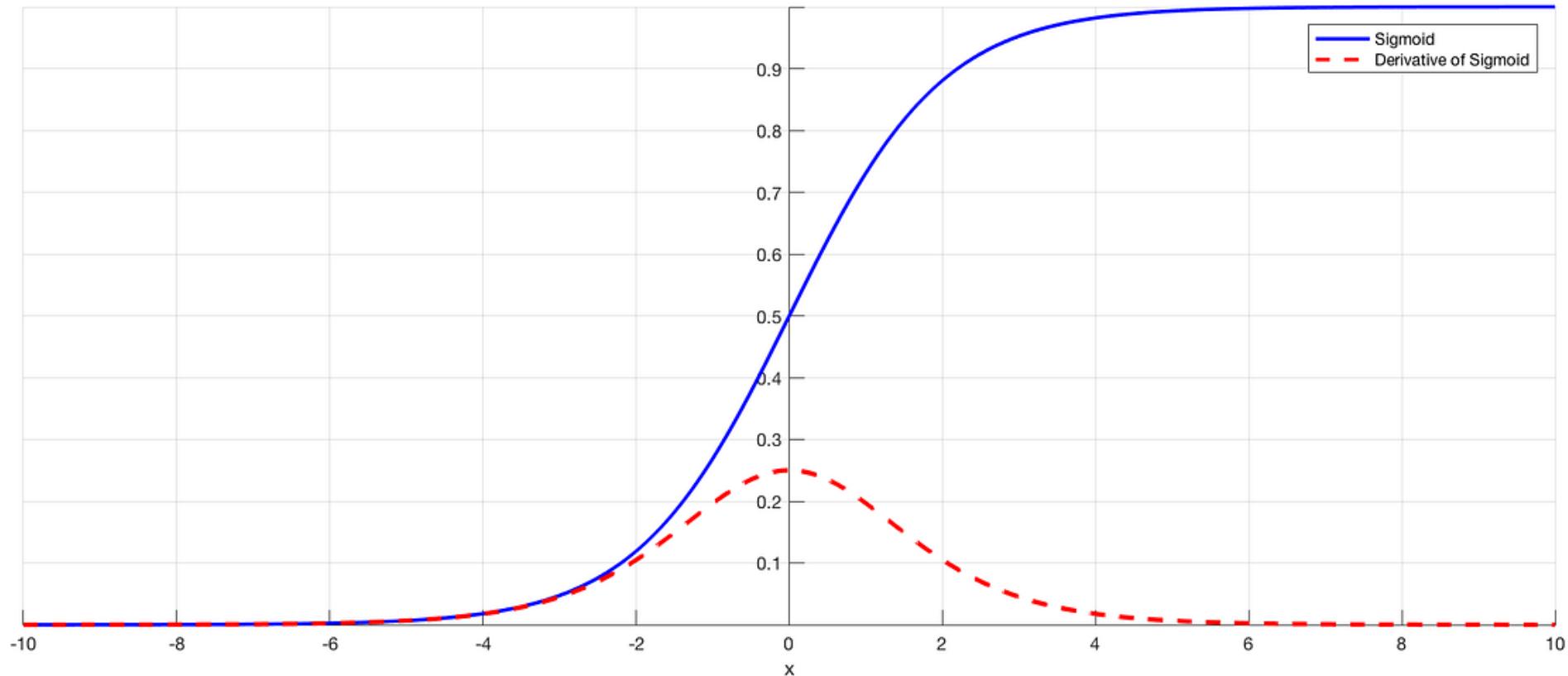
Caratteristiche della funzione di attivazione

La funzione di attivazione trasforma i dati in input pesati per generare un output.

Nelle reti neurali vogliamo che il nostro algoritmo sia in grado di affrontare le profonde non-linearità presenti nei dati, per cui cercheremo funzioni di attivazione non lineari che se applicate, un layer dopo l'altro, ci consentono di approssimare funzioni anche molto complesse.

Un aspetto da tener presente è che la nostra funzione di attivazione deve essere derivabile e che la sua derivata favorisca in qualche modo l'apprendimento della rete, cioè che la sua derivata non sia mai nulla.

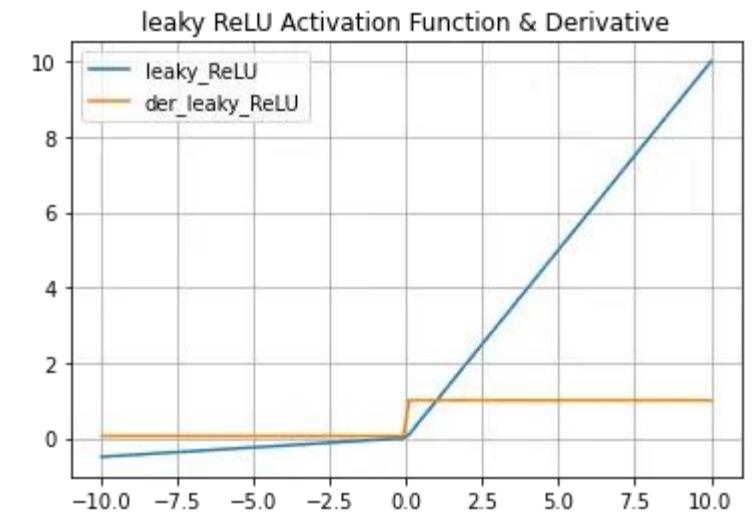
Derivata del sigmoide



Derivata

Sigmoide, ReLU, e molte altre funzioni di attivazione soffrono del fatto che se ci troviamo molto lontani dall'area di non-linearità (cioè valori molto grandi o molto negativi nel secondo caso), la derivata non ci aiuta a correggere il valore dei pesi.

Funzioni come la Leaky-ReLU hanno una derivata piccola ma non nulla in tutto il loro spazio.



Linear activation

La funzione di attivazione lineare non è nient'altro che la combinazione lineare degli ingressi. Utilizzare più layer consecutivi di attivazioni lineari non aggiunge nessuna informazione al modello, è immediatamente approssimabile con un layer solo.

ES: scriviamo l'output di una rete neurale di 2 layer con attivazioni lineari.

In alcuni casi è comunque utile utilizzare un'attivazione lineare.

Esempio: l'ultimo layer di una rete neurale per un problema di regressione di cui non siamo sicuri degli estremi superiori o inferiori e normalizzando l'uscita non riusciremmo a coprire tutti i valori possibili in produzione.

Neural network

Proviamo ora a giocare con una rete neurale già implementata per problemi di regressione e classificazione, vediamo come impattano il diverso numero di neuroni, di layer e di funzioni di attivazione

<https://playground.tensorflow.org/>

Multi-Layer Perceptron - MLP

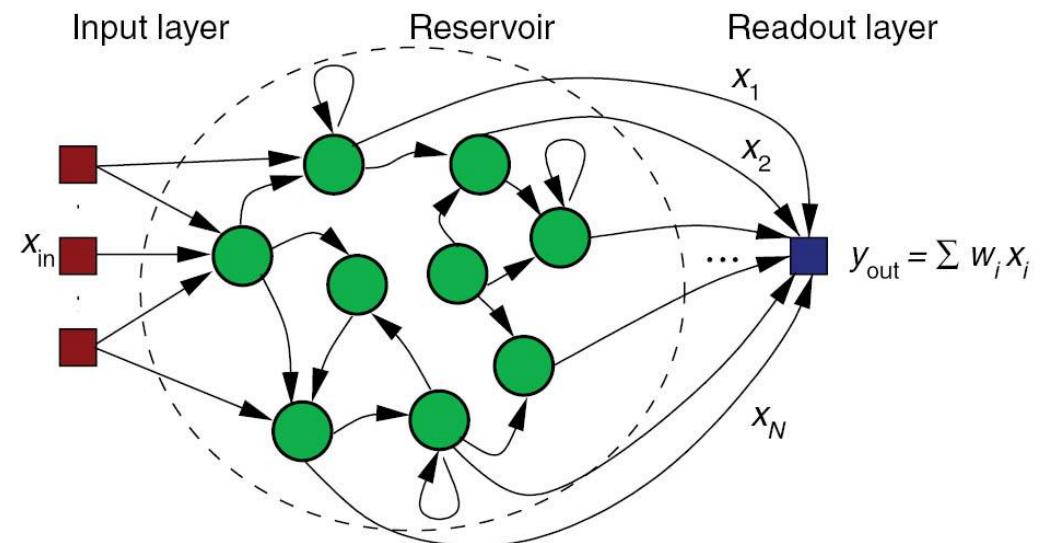
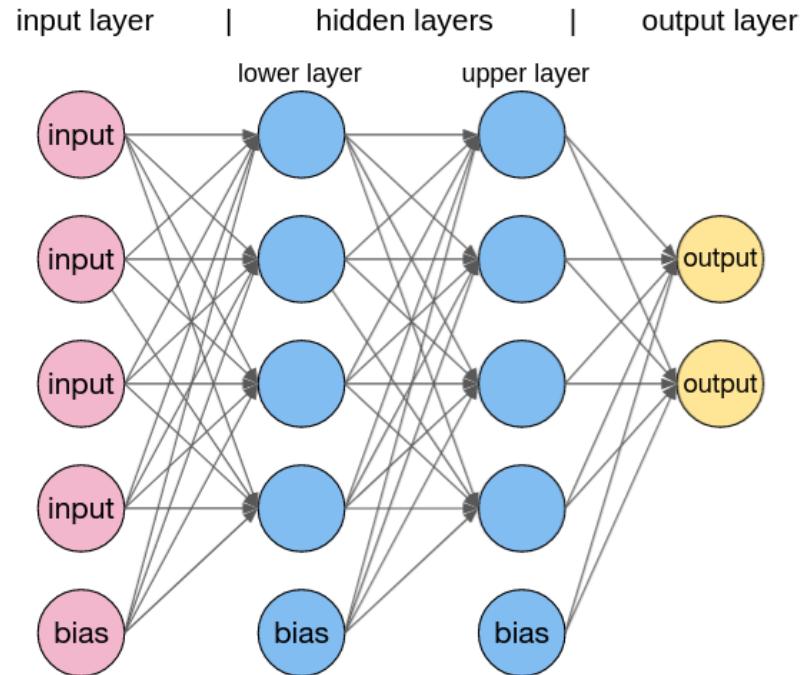
Il Multi-Layer Perceptron, o MLP, è la tipologia di rete neurale che abbiamo visto fino ad adesso. Tutti i neuroni del layer precedente sono connessi con il layer successivo.

Questa formulazione è molto comoda perché ci consente

- di definire con semplicità i nostri layer
- di affrontare problemi molto non lineari aggiungendo semplicemente altri layer
- di parallelizzare i conti di ciascun layer in quanto ogni nodo è indipendente

Di contro, il numero di parametri w all'interno del modello cresce molto rapidamente.

MLP vs rest



Nelle reservoir networks non c'è distinzione tra un layer e l'altro, per cui per elaborare un neurone dobbiamo elaborare tutta la sequenza.

Apprendimento

Fin'ora abbiamo visto come definire una rete neurale, ma com'è che apprende?

Il processo di apprendimento di una rete neurale si chiama back-propagation e si compone di questi passi

- Viene preso un batch di dati (es. 16 dati)
- Viene inserito nell'input layer e passato attraverso la rete fino a produrre gli output (forward pass)
- I risultati intermedi vengono salvati
- Viene misurato l'errore di predizione tramite la **loss functions**
- Viene calcolato quanto il layer precedente contribuisce all'errore
- Viene calcolato l'aggiustamento medio per ciascun neurone e riapplicato tramite gradient descent
- Ricalcolando i valori che avrebbero dovuto ottenere le attivazioni del layer precedente, ripetiamo per ogni layer fino all'input

Apprendimento

Andando un po' più nel dettaglio, partendo dalle nostre predizioni e il nostro errore di predizione

$$y_k = \sum_i w_{ki} x_i \quad E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

Il gradiente rispetto al peso è

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

Ma ogni neurone è una attivazione (wx) passata per una funzione di attivazione (h), quindi ciascun output è

$$a_j = \sum_i w_{ji} z_i \quad z_j = h(a_j).$$

Apprendimento

Calcolando che dovremo avere la derivata della funzione composta per modificare i nostri pesi, sfruttiamo la chain rule e definiamo δ_j

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \quad \delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

avremo

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \quad \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$$

e dopo un paio di passaggi manuali

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

con δ_j pari a l'errore da applicare al nostro peso

Attenzione

Ci sono alcuni punti a cui fare attenzione quando si addestra una rete neurale

- Tutti i pesi vanno **inizializzati random**, altrimenti non imparerebbe niente. Nel caso due pesi fossero esattamente identici all'interno dello stesso layer il loro contributo sarebbe identico e dimezzato.
- Si addestra a **batch** di dati, campionati random, secondo lo **stochastic gradient descent** altrimenti ciascun esempio di training varierebbe ogni neurone. Una volta terminati tutti i batch del training set si è terminata un'epoca e si riinizia con la successiva.

Rimangono valide tutte le considerazioni fatte per le altre tecniche di machine learning

- Scalare o normalizzare le features in ingresso aiuta enormemente
- Si soffre facilmente di overfitting per via dell'alto numero di pesi, per cui possiamo usare tecniche di regolarizzazione
- Il dataset è meglio che sia bilanciato opportunamente
- Il learning rate impatta molto e spesso va variato durante l'apprendimento.

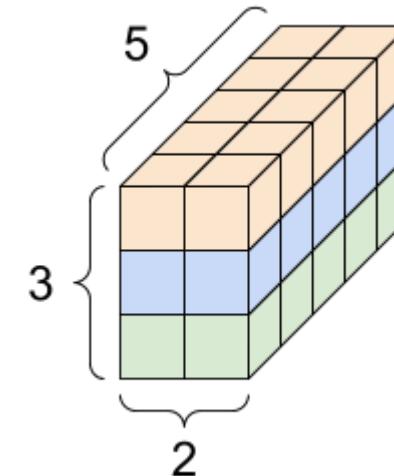
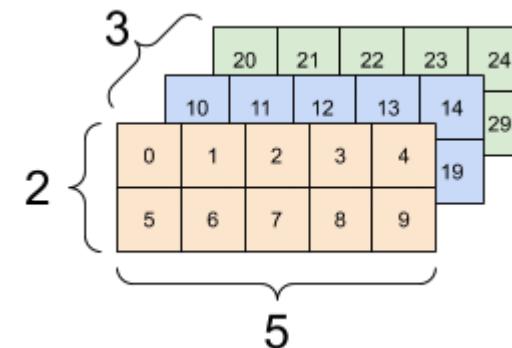
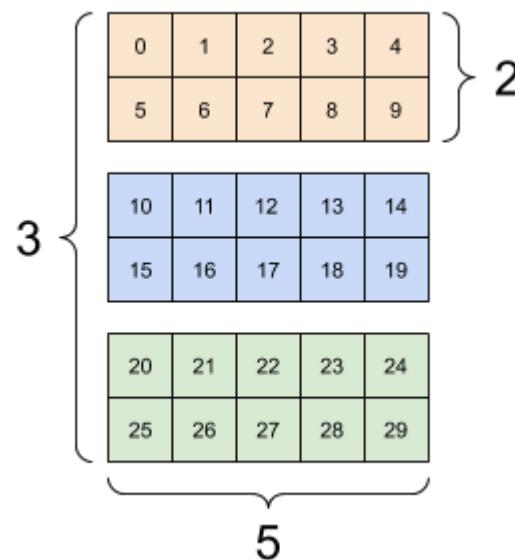
Introduzione a Tensorflow

Per costruire ed addestrare reti neurali si utilizzano framework più flessibili ed a basso livello rispetto a Scikit-Learn. I due framework principali sono **Tensorflow** (di Google) e **PyTorch** (ex. Facebook, ora Linux Foundation). Nel corso vedremo prevalentemente Tensorflow 2, eventualmente con qualche rimando a PyTorch, i concetti di fondo sono identici, cambia leggermente l'implementazione.

Con questi framework andiamo a costruire noi l'algoritmo di Machine Learning invece di trovarlo già implementato.

Tensori

Iniziamo dal concetto di tensore, ovvero una rappresentazione in memoria di un oggetto N dimensionale, immutabile



<https://www.tensorflow.org/guide/tensor?hl=it>

Definizione di un modello

Il nostro algoritmo sarà composto da tre parti

- Una struttura della rete
- Una loss function da ottimizzare
- Un ottimizzatore che applica la loss per variare i pesi

Struttura della rete

Nel nostro MLP la struttura della rete è definita da

- Il numero di layer
- Quanti neuroni avrà ogni layer
- Quale funzione di attivazione scelgiamo per ogni layer
- Eventuali layer funzione (Flatten, DropOut, ecc. che vedremo più avanti)
- Eventuali regolarizzazioni
- A cosa è connesso ciascun layer

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 3072)	0
dense_6 (Dense)	(None, 200)	614600
dense_7 (Dense)	(None, 150)	30150
dense_8 (Dense)	(None, 10)	1510
=====		
Total params: 646,260		
Trainable params: 646,260		
Non-trainable params: 0		

Loss function

Con la Loss function andiamo a definire la funzione di costo che dovremo ottimizzare, tanto più è grande tanto più saremo lontani dall'ottimo.

Possiamo sfruttare un set di funzioni di costo standard come

- MeanSquaredError, MeanAbsoluteError, MAPE, ecc. per problemi di regressione
- BinaryCrossentropy, CategoricalCrossentropy, CosineSimilarity, ecc. per problemi di classificazione
- e numerose altre

Oppure possiamo costruirci una funzione di loss custom, come ad esempio una combinazione di queste per parti diverse del nostro algoritmo.

Metrics

Oltre alla funzione di loss possiamo far calcolare al modello in automatico delle metriche che possono darci delle informazioni sul nostro modello.

La differenza sta nel fatto che non ottimizzeremo rispetto a queste metriche, sono semplici misure.

Ad esempio, nei problemi di classificazione

- AUC
- Accuracy
- Precision, recall

Ottimizzatori

Nel momento in cui andiamo a fare back-propagation dobbiamo decidere come applicare il gradiente. Negli anni si sono sviluppati diversi ottimizzatori che aggiornano i pesi in modo leggermente diverso

Stochastic Gradient Descent

È quello tradizionale che abbiamo già visto anche nel machine learning tradizionale. Al learning rate viene sommato un «momentum» che tiene conto della velocità con cui viene sceso il gradiente. Abbasserà il learning rate con il rallentare dell'ottimizzazione.

RMSProp

Simile al SGD, calcola una media mobile dei quadrati dei gradienti che usa per dividere, sotto radice, il learning rate. È un altro modo di vedere la velocità con cui si sta scendendo il gradiente.

Adagrad

È uguale a RMSProp come impostazione ma si salva una media mobile per ogni direzione. In questo modo parametri diversi vengono aggiornati con learning rate pesati diversamente, rendendolo più robusto a variazioni di «scala» tra le features.

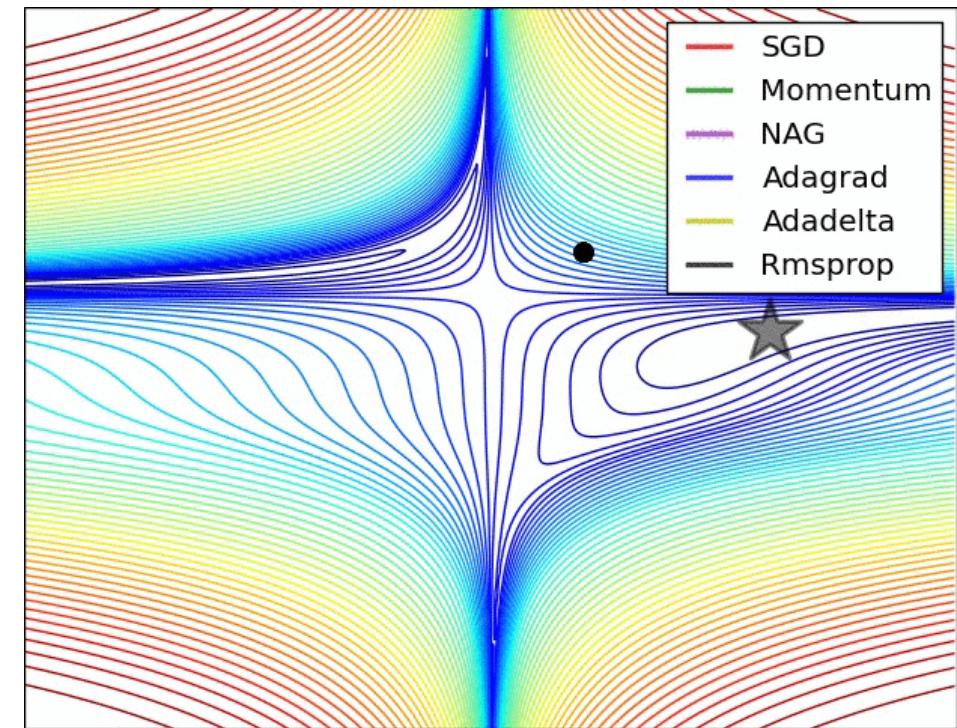
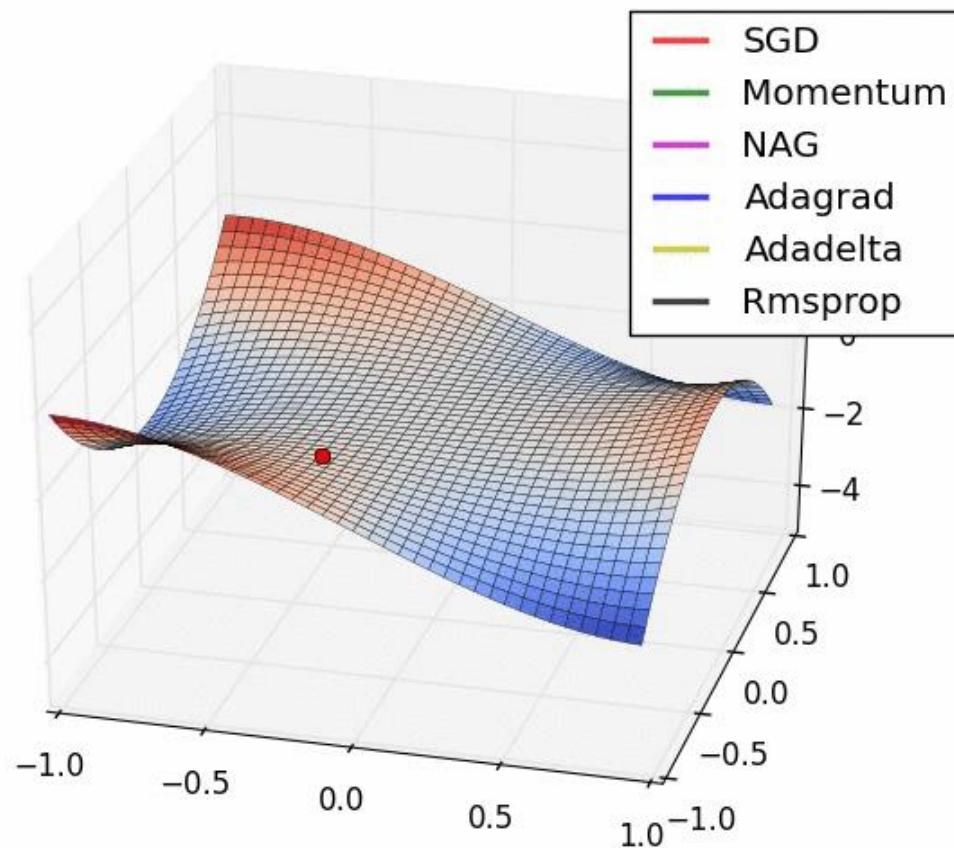
Ottimizzatori

Adam

È uno dei più usati e aggiunge all'impostazione di RMSProp e Adagrad un cumulativo della storia dei gradienti da usare come il momentum dell'SGD. In generale si parte usando questo per poi variare i diversi parametri.

Tutti questi ottimizzatori hanno a loro volta dei parametri da cercare, in primis il learning rate e poi tutti gli altri. È critica la scelta dell'ottimizzatore e dei suoi iperparametri perché determina la velocità con cui la rete apprende. Tanto più abbiamo strutture grandi, tanto più è determinante.

Ottimizzatori



Esercizio

Applichiamo un ottimizzatore ad una variabile per ottimizzarla e proviamo a costruiamo un MLP sia con Sklearn MLPRegressor che con Tensorflow per vederne le differenze.

<https://colab.research.google.com/drive/1IEK-KNBY5qx-CthQag5yIMliron-4Yie?usp=sharing>



10 – Neural Networks

Daniele Gamba

2022/2023

Lezione precedente

Abbiamo iniziato a vedere

- Cos'è una rete neurale
- Concetto di neurone e layer di neuroni
- Funzioni di attivazione
- Ottimizzatori
- Loss e metriche

Exe

Costruiamo un MLP, compiliamolo ed addestriamolo a prevedere i prezzi delle case

https://colab.research.google.com/drive/1BemOYYUABKNIBKKoy7y_Hd6VAJ9ttpjj?usp=sharing

MLP

Abbiamo visto la scorsa lezione che l'MLP è una serie di layer, tutti connessi tra loro, con uno o più ingressi e una o più uscite.

Questo ci consente di definire come sono connessi i neuroni **densi** tra loro ma possiamo liberamente applicare il concetto per ricreare diversi blocchi utili.

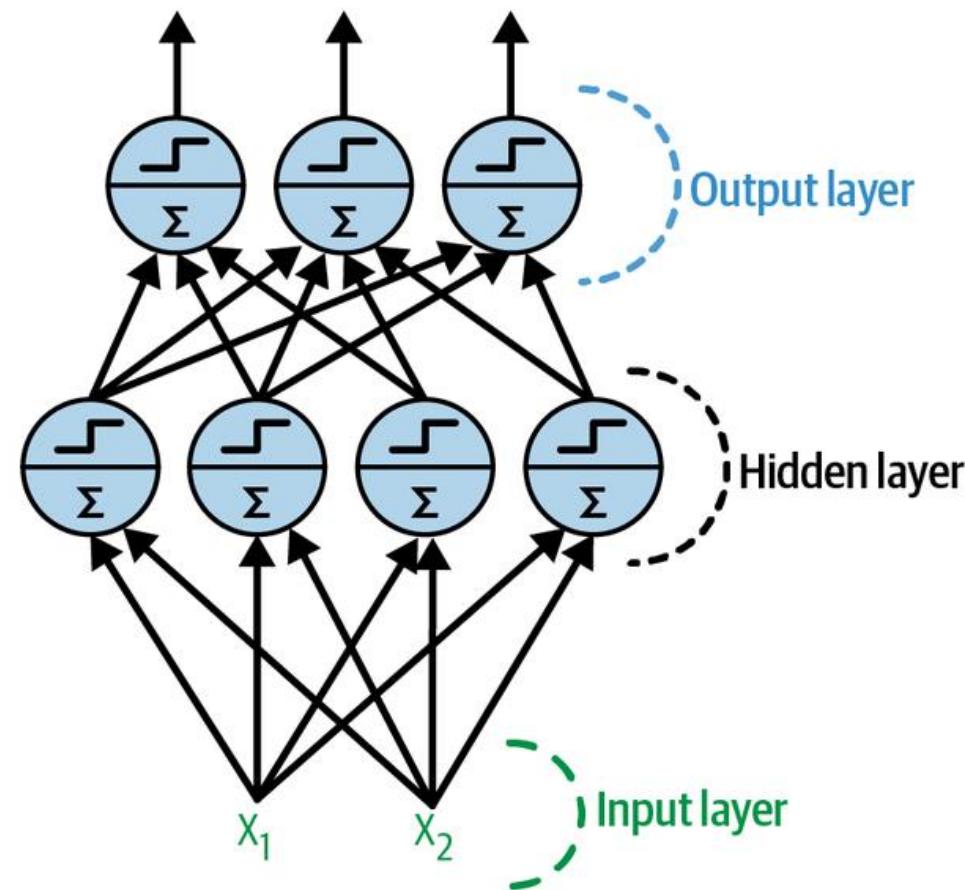
Sequential model

La base è un modello **sequenziale**, ovvero in cui tutti i layer sono connessi uno di seguito all'altro.

Per definire questo tipo di modelli possiamo appoggiarci al Sequential model di Keras in cui dobbiamo semplicemente specificare quali layer vogliamo e sarà la classe ad implementarlo.

```
norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
model = tf.keras.Sequential([
    norm_layer,
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(1)
])
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(loss="mse", optimizer=optimizer, metrics=["RootMeanSquaredError"])
norm_layer.adapt(X_train)
history = model.fit(X_train, y_train, epochs=20,
                     validation_data=(X_valid, y_valid))
mse_test, rmse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3]
y_pred = model.predict(X_new)
```

Sequential model

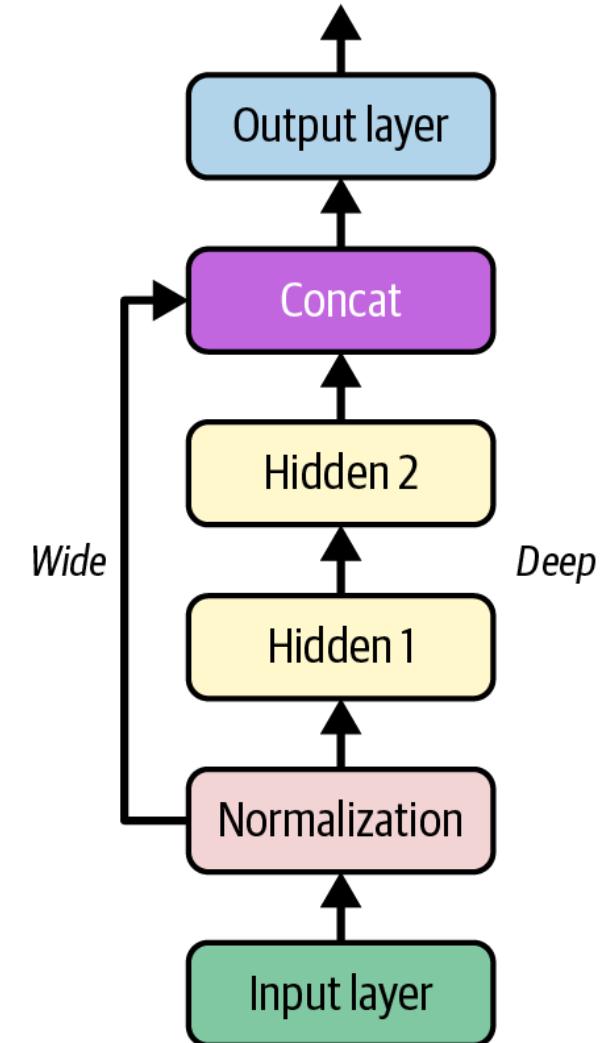


Wide and deep

Un'altra struttura che possiamo realizzare è la **Wide and Deep**.

In questo caso il modello passa l'ingresso sia attraverso una deep neural network che direttamente verso l'uscita.

In questo modo il nostro output layer potrà sfruttare informazioni che arrivano direttamente dall'output sia informazioni più «profonde» che arrivano attraverso la struttura della rete.



Wide and deep

In questo caso la rete non è più interamente sequenziale perché l'input è passato anche direttamente verso l'uscita.

Per questo dobbiamo specificare noi i layer e successivamente specificare come questi si connettono tra loro.

layer

```
normalization_layer = tf.keras.layers.Normalization()  
hidden_layer1 = tf.keras.layers.Dense(30, activation="relu")  
hidden_layer2 = tf.keras.layers.Dense(30, activation="relu")  
concat_layer = tf.keras.layers.concatenate()  
output_layer = tf.keras.layers.Dense(1)
```

connessioni

```
input_ = tf.keras.layers.Input(shape=X_train.shape[1:])  
normalized = normalization_layer(input_)  
hidden1 = hidden_layer1(normalized)  
hidden2 = hidden_layer2(hidden1)  
concat = concat_layer([normalized, hidden2])  
output = output_layer(concat)
```

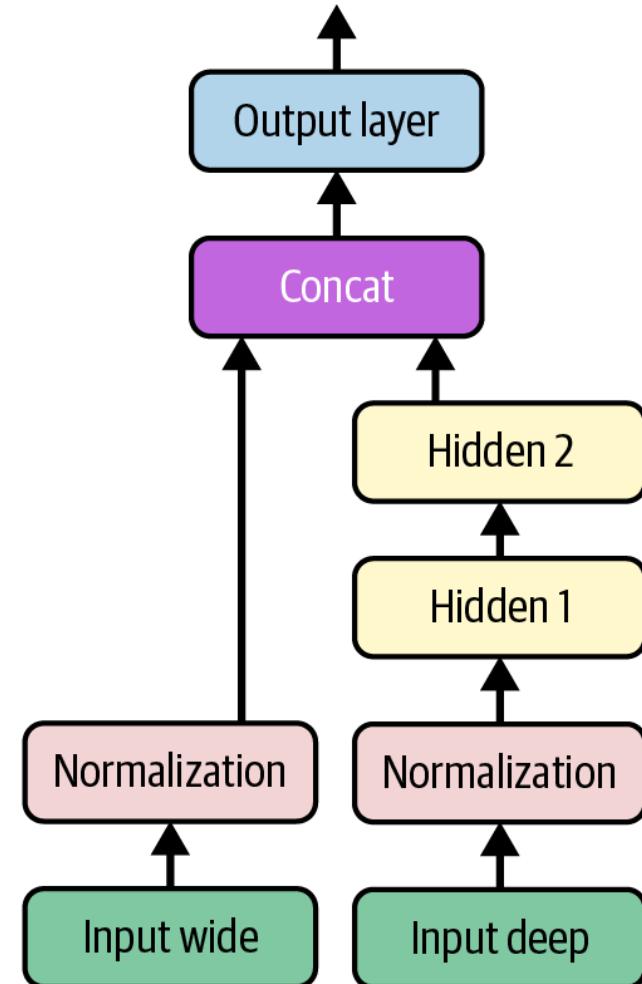
modello

```
model = tf.keras.Model(inputs=[input_], outputs=[output])
```

Input multipli

Mettiamo caso che abbiamo una del prezzo di una casa che viene da un modello generale e che vogliamo raffinare con le nostre features più grezze. In questo caso avremmo a disposizione due input diversi che possiamo elaborare con due flussi diversi nella rete.

Possiamo tranquillamente definire due input diversi e costruire così il nostro modello.



Input multipli

Nel caso di input multipli dovremo prestare attenzione quando creiamo il modello di indicare in input entrambi i layer e di concatenarli.

È importante anche in fase di inferenza andare a passare i dati nell'ordine giusto così come ci siamo immaginati il loro percorso.

inferenza

modello

```
input_wide = tf.keras.layers.Input(shape=[5]) # features 0 to 4  
input_deep = tf.keras.layers.Input(shape=[6]) # features 2 to 7  
norm_layer_wide = tf.keras.layers.Normalization()  
norm_layer_deep = tf.keras.layers.Normalization()  
norm_wide = norm_layer_wide(input_wide)  
norm_deep = norm_layer_deep(input_deep)  
hidden1 = tf.keras.layers.Dense(30, activation="relu")(norm_deep)  
hidden2 = tf.keras.layers.Dense(30, activation="relu")(hidden1)  
concat = tf.keras.layers.concatenate([norm_wide, hidden2])  
output = tf.keras.layers.Dense(1)(concat)  
model = tf.keras.Model(inputs=[input_wide, input_deep], outputs=[output])
```

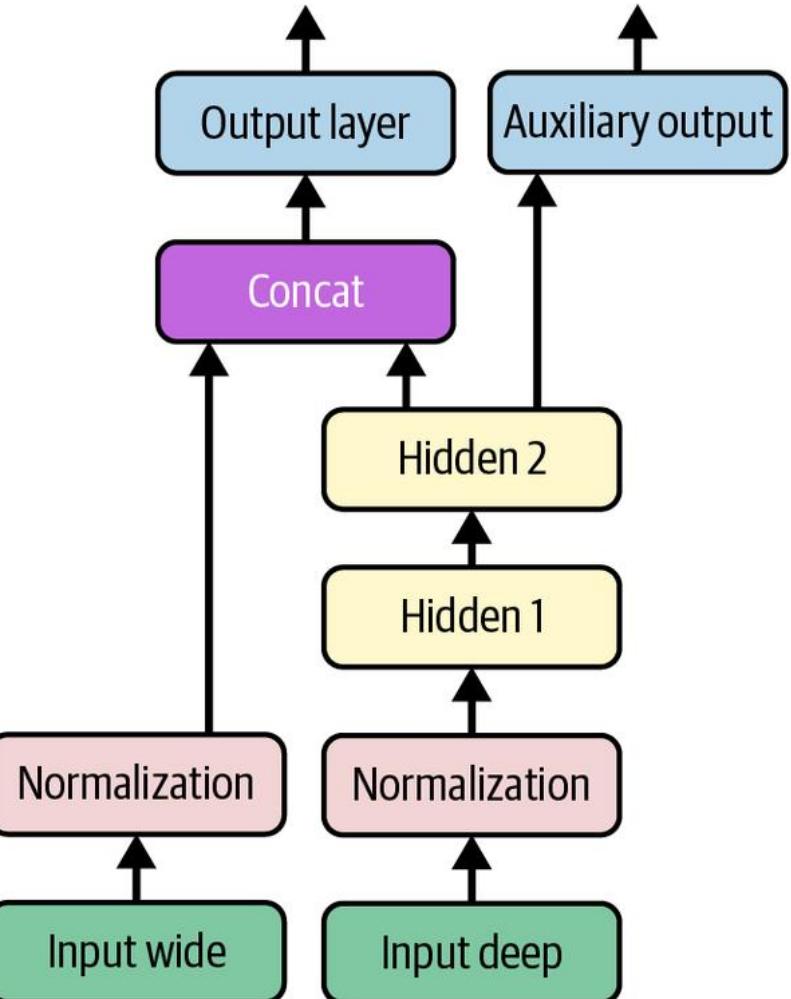
```
X_train_wide, X_train_deep = X_train[:, :5], X_train[:, 2:]  
X_valid_wide, X_valid_deep = X_valid[:, :5], X_valid[:, 2:]  
X_test_wide, X_test_deep = X_test[:, :5], X_test[:, 2:]  
X_new_wide, X_new_deep = X_test_wide[:3], X_test_deep[:3]
```

```
norm_layer_wide.adapt(X_train_wide)  
norm_layer_deep.adapt(X_train_deep)  
history = model.fit((X_train_wide, X_train_deep), y_train, epochs=20,  
                    validation_data=((X_valid_wide, X_valid_deep), y_valid))  
mse_test = model.evaluate((X_test_wide, X_test_deep), y_test)  
y_pred = model.predict((X_new_wide, X_new_deep))
```

Output multipli

Chiaramente così come possiamo avere input multipli possiamo specificare diversi layer come output del modello.

Mettiamo che vogliamo non solo la predizione ma anche le correzioni che il nostro modello predice. Possiamo specificare un secondo output layer e aggiungerlo al modello.



Output multipli

Nel caso di output multipli dovremo specificare **una funzione di loss** per ogni output e di conseguenza un target rispetto al quale fare ottimizzazione.

Così come per gli input dovremo ricordarci in train ed inferenza di passare tutti i dati nel giusto ordine.

modello |

```
[...] # Same as above, up to the main output layer  
output = tf.keras.layers.Dense(1)(concat)  
aux_output = tf.keras.layers.Dense(1)(hidden2)  
model = tf.keras.Model(inputs=[input_wide, input_deep],  
outputs=[output, aux_output])
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)  
model.compile(loss=["mse", "mse"], loss_weights=(0.9, 0.1), optimizer=optimizer,  
metrics=["RootMeanSquaredError"])
```

Output layer

Abbiamo visto come l'output layer rappresenta la nostra predizione \hat{y}

Nel caso di problemi di **regressione** / forecast di una singola variabile possiamo usare un **layer lineare** in uscita per prevedere direttamente il valore o usare altre funzioni di attivazione eventualmente riscalando i dati.

Nel caso di **classificazioni single class** possiamo usare un neurone singolo con una funzione di attivazione **sigmoidale** in modo da rappresentarne la probabilità tra 0 ed 1.

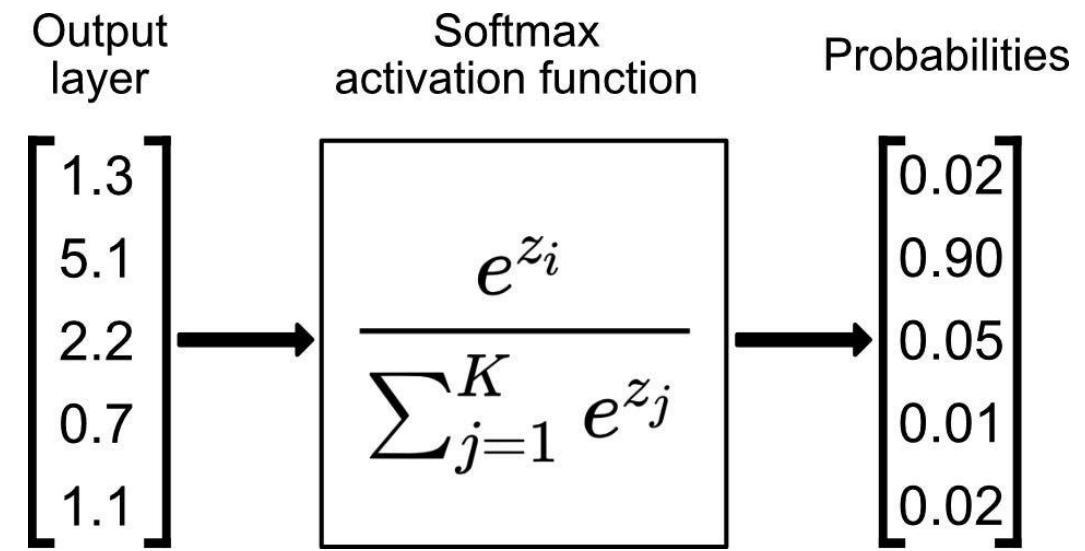
Nel caso di problemi **multiclass classification** dobbiamo distinguere

- Single label, ovvero il nostro esempio può appartenere solamente ad una classe (es. cane, gatto, coniglio, ..)
- Multi label, ovvero il nostro esempio può appartenere a più classi (sano, in funzione, consumo corretto, ..)

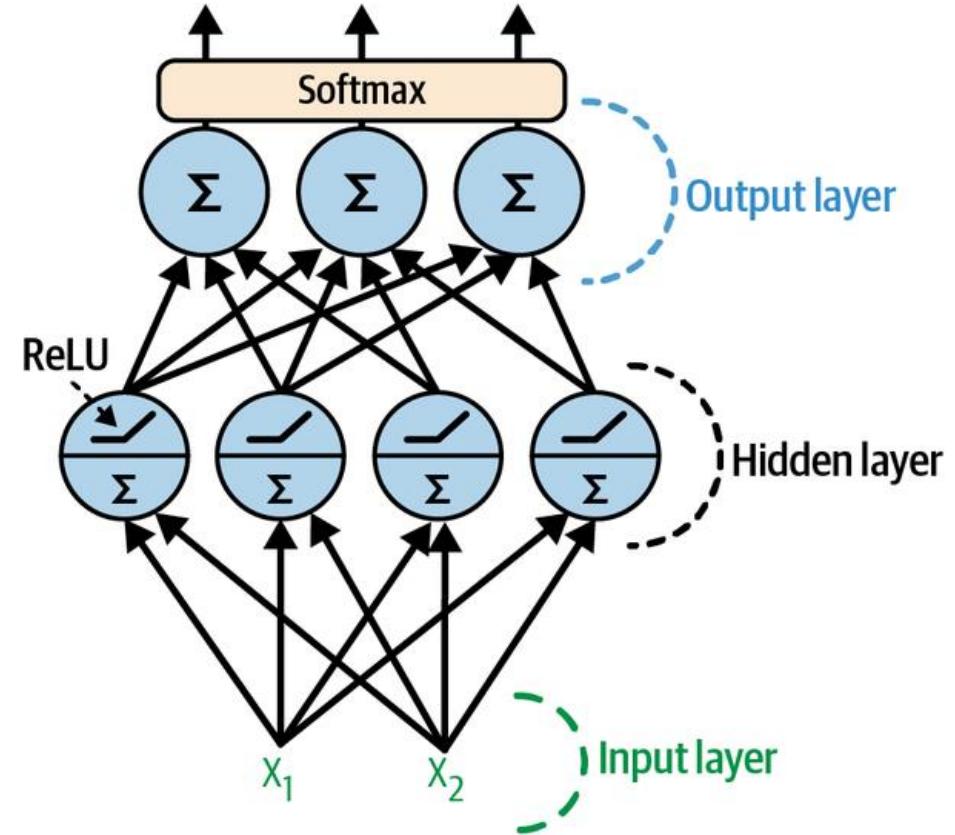
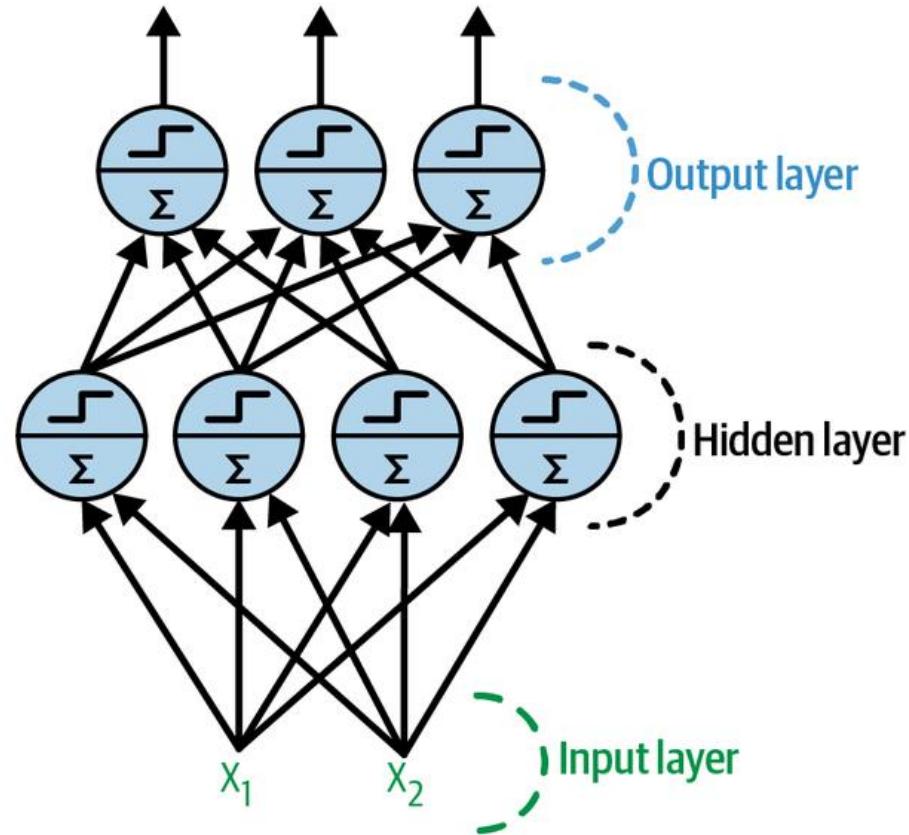
Output layer

Nel caso single-label a fronte di N classi possiamo utilizzare la softmax che ci consente di scegliere come classe in uscita quella con il valore più alto, riscalando le probabilità in funzione dei valori relativi.

Nel caso multi-label ogni neurone avrà invece la sua funzione di attivazione indipendente.



Softmax vs Independent



Vanishing gradient

Un altro dei problemi che si hanno nell'addestrare le reti è chiamato «scomparsa del gradiente».

Quando usiamo funzioni di attivazione come sigmoide o tangente iperbolica la derivata tende ad essere molto piccola come ci allontaniamo dal centro. Durante la fase di apprendimento, la chain rule andrà a moltiplicare tra loro le derivate parziali di ogni parametro, comportando quindi il prodotto tra loro di valori molto piccoli.

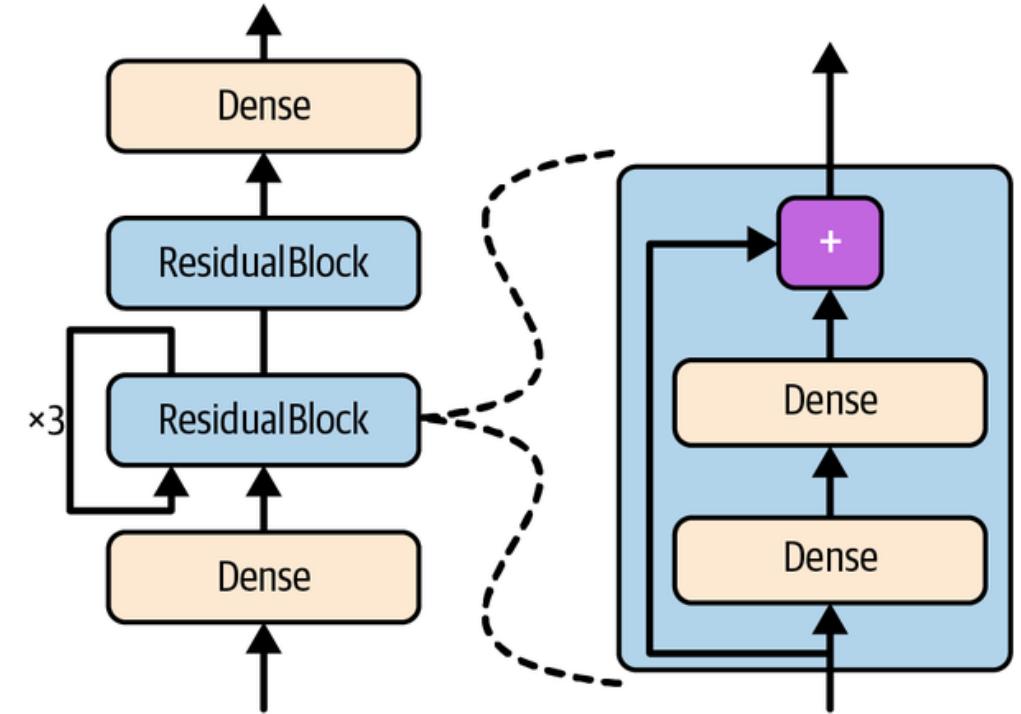
Di conseguenza nelle reti profonde si ha la scomparsa del gradiente, per cui i primi layer non riescono ad apprendere. Per questo si possono cambiare funzioni di attivazione o sfruttare delle «skip-connection» come nella wide and deep network, per portare il gradiente direttamente sui layer più profondi della rete.

Residual block

Il **residual block** è un modulo della rete che possiamo sfruttare nei casi di vanishing gradient.

Si basa sul concetto di **skip-connection**, ovvero andare a connettere direttamente l'input con l'uscita del blocco, tenendo in mezzo le features più deep. La skip connection porta il residuo ad una profondità maggiore nella rete, per questo si chiamano anche **ResNet**.

In molti casi si usa concatenare più residual block di fila.



Custom Layer

In Tensorflow possiamo definire un layer custom come il residual block, andando ad estendere la classe Layer. In questo modo possiamo evitare di scriverci a mano tutti i layer, andando ad applicare ad esempio N layer densi in automatico.

Ci basterà inizializzare il nostro ResidualBlock come uno qualunque degli altri layer.

```
class ResidualBlock(tf.keras.layers.Layer):
    def __init__(self, n_layers, n_neurons, **kwargs):
        super().__init__(**kwargs)
        self.hidden = [tf.keras.layers.Dense(n_neurons, activation="relu",
                                            kernel_initializer="he_normal")
                      for _ in range(n_layers)]
    def call(self, inputs):
        Z = inputs
        for layer in self.hidden:
            Z = layer(Z)
        return inputs + Z
```

Recap

Abbiamo visto come si struttura un MLP

- Sequenziale e non
- Con input e output multipli
- Con diversi layer di uscita
- Con Residual Blocks

Training di una rete

Abbiamo visto come l'addestramento prevede chiamare il .fit del nostro modello e il nostro ottimizzatore si prenderà in carico di applicare il gradiente ai nostri parametri per migliorarli.

Nel caso in cui iniziamo a definire delle strutture custom e delle funzioni di loss custom, potremmo aver bisogno di personalizzare anche il training della nostra rete.

Per far ciò Tensorflow ci mette a disposizione Autodiff per calcolare in automatico i gradienti da applicare in funzione di come vengono usate le nostre variabili.

Gradient Tape

All'interno del «with tf.GradientTape() ..» il «nastro adesivo» si incolla tutte le operazioni che riguardano le variabili.

Alla fine chiamando .gradient() ci vengono calcolate tutte le derivate parziali.

w1, w2 = tf.Variable(5.), tf.Variable(3.)

with tf.GradientTape() **as** tape:

z = f(w1, w2)

gradients = tape.gradient(z, [w1, w2])

Custom training loop

Ora che sappiamo come calcolarci i gradienti possiamo costruirci un training loop custom completo.

Dopo aver calcolato i gradienti li passiamo al nostro ottimizzatore che li applicherà, con le sue logiche, ai nostri pesi.

Tutto ciò è normalmente astratto dal .fit del modello ma vedremo che per algoritmi più complessi ci servirà scriverlo a mano.

```
for epoch in range(1, n_epochs + 1):
    print("Epoch {}/{}".format(epoch, n_epochs))
    for step in range(1, n_steps + 1):
        X_batch, y_batch = random_batch(X_train_scaled, y_train)
        with tf.GradientTape() as tape:
            y_pred = model(X_batch, training=True)
            main_loss = tf.reduce_mean(loss_fn(y_batch, y_pred))
            loss = tf.add_n([main_loss] + model.losses)

        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
        mean_loss(loss)

        for metric in metrics:
            metric(y_batch, y_pred)

        print_status_bar(step, n_steps, mean_loss, metrics)
```

Pre-training

Una volta definita la struttura del nostro modello possiamo generalmente eseguire un primo train e poi un fine tuning.

Il primo train ha l'obiettivo di verificare la capacità del modello di convergere, in questa fase si valuta il numero di layer, le funzioni di attivazione, la struttura del modello e la scelta della loss. Il pre-train a volte viene eseguito anche su un set di dati più ampio e potenzialmente sporco con un learning rate più alto.

Nel pre-train possiamo anche utilizzare dei metodi di ottimizzazione tipo **GridSearch** per tunare alcuni iperparametri come il numero di layer. Fare **Hyperparameter tuning** nel caso delle reti neurali però è estremamente più oneroso perché non abbiamo più dei *weak-learner* ma veri e propri oggetti complessi.

Fine tuning

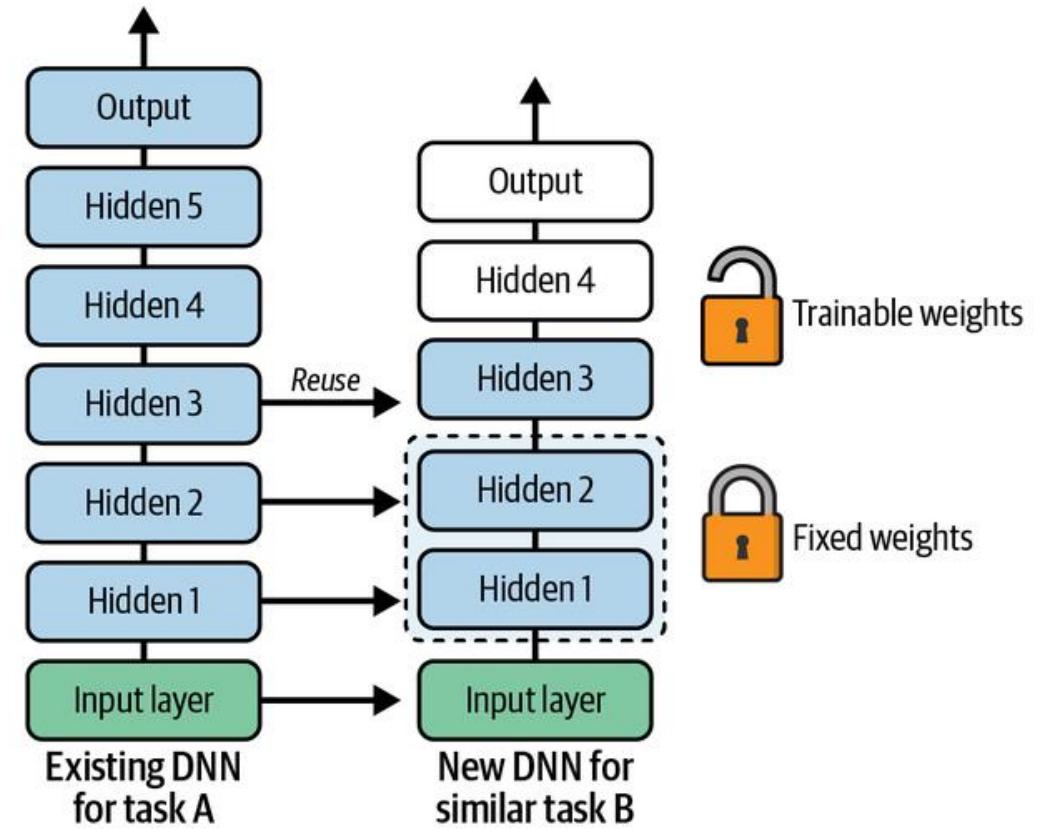
Il fine tuning consiste invece nell'andare a perfezionare il modello in modo che funzioni al meglio sul nostro task.

Per fare ciò possiamo

- Lavorare su un set di dati di maggior qualità
- Andare a cambiare il learning rate
- Aggiungere o modificare i parametri di regolarizzazione
- Cambiare batch size
- Fissare il valore di alcuni layer
- ...

Fixed weight

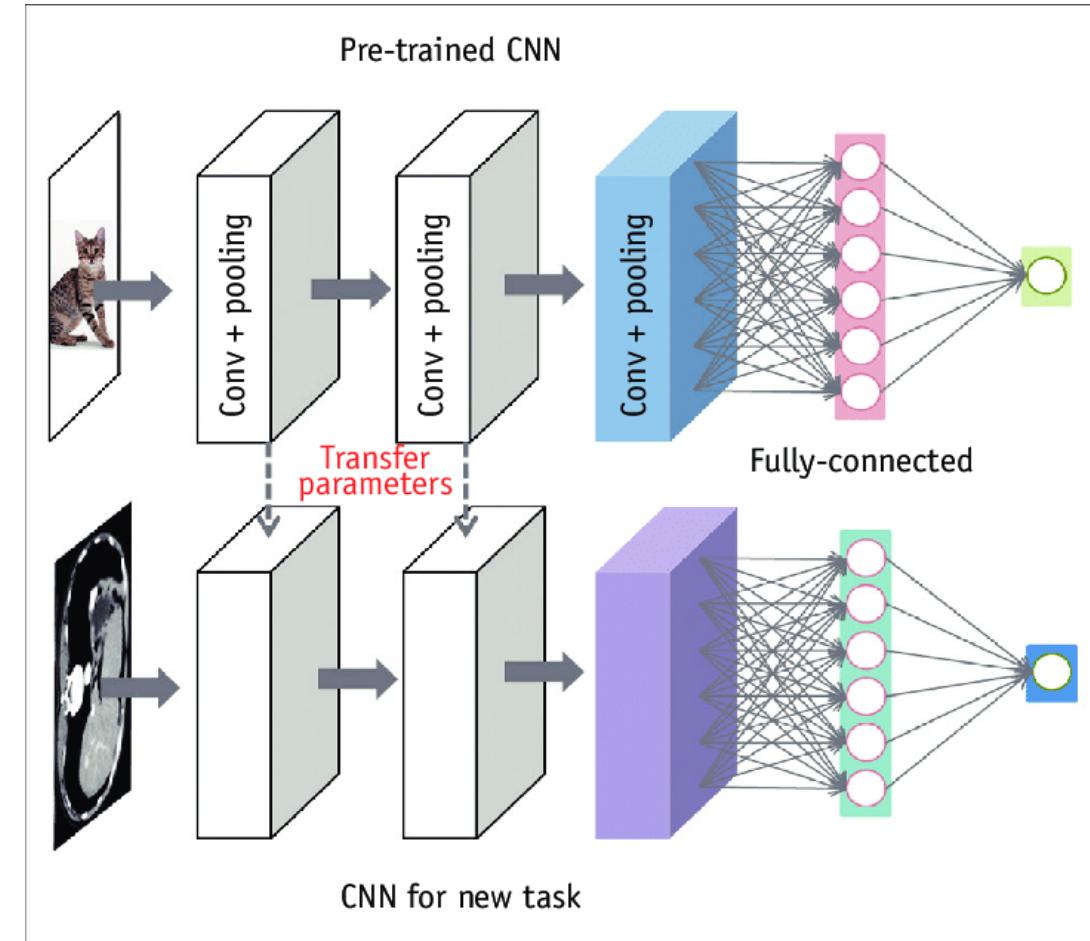
Nel caso avessimo già addestrato un modello su un task simile, oppure abbiamo fatto un pre-train soddisfacente e volessimo mantenerne inalterate le features di «basso livello» che questo estrae, possiamo fissare i pesi di alcuni layer.



Transfer learning – supervised pre-train

Il concetto di utilizzare una rete, o un pezzo di rete, pre-addestrata per risolvere un task leggermente diverso si chiama Transfer Learning.

Grazie al transfer learning possiamo pre-addestrare una rete ad esempio sul COCO dataset a classificare immagini tra 80 classi e ri-sfruttare quella conoscenza, le features che estrae per la sua classificazione, per addestrare un nostro classificatore.



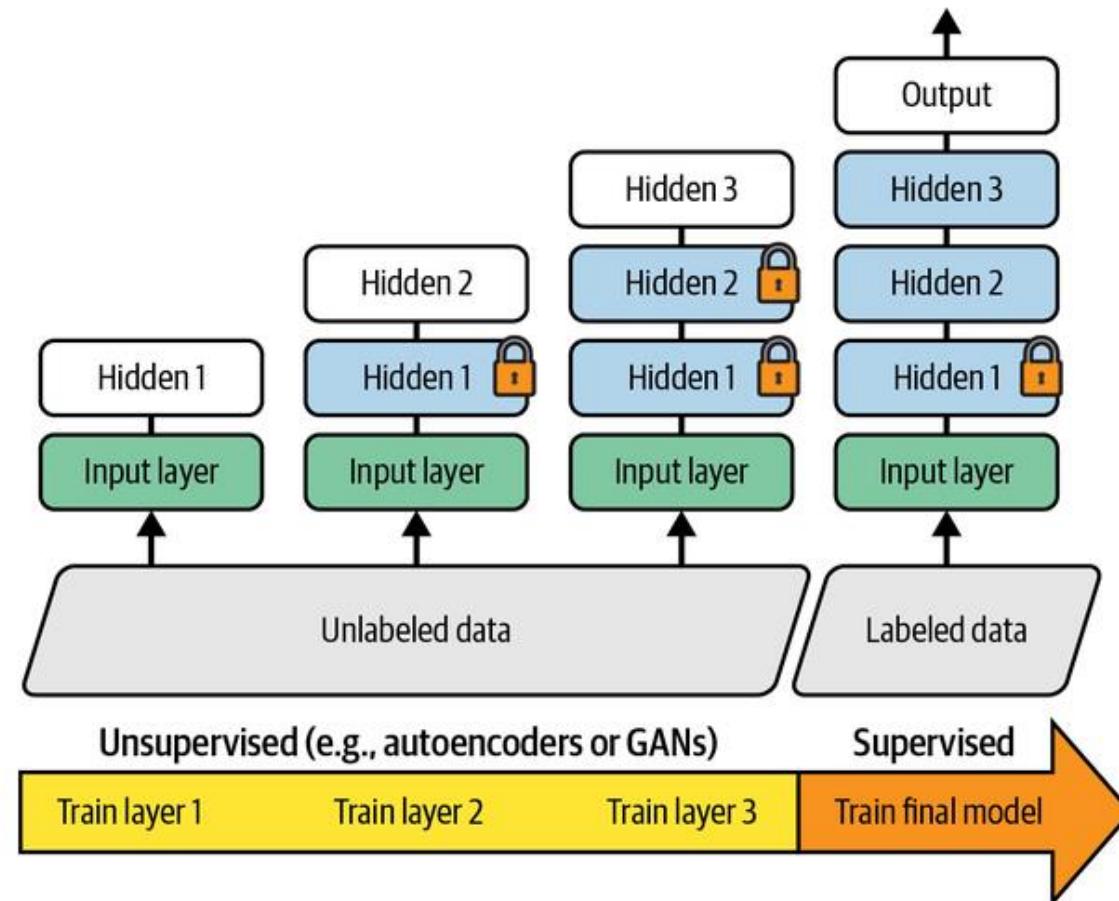
Unsupervised pre-train

Nel caso dell'unsupervised pre-train useremo delle tecniche di unsupervised learning per addestrare i nostri layer ad estrarre features significative sullo stesso nostro dataset ma con un task diverso, ad esempio

- Comprimere le informazioni e decomprimerle (Autoencoder)
- Generare dati verosimili (Generative Adversarial Networks)
- Ricostruire parti di dati (self-supervised)
- Togliere il rumore dai dati (de-noising)

Questi task diversi gettano le basi per estrarre comunque features significative nei primi layer della rete.

Unsupervised pre-train



Overfitting

Nel caso delle reti neurali abbiamo spesso un numero veramente molto elevato di parametri, per cui il classico rapporto di 10-20 esempi per parametro è spesso non raggiungibile.

Pre addestrare e tunare la rete bloccando alcuni layer, o con un learning rate molto più basso, sono solo due delle diverse tecniche che possiamo applicare per ridurre l'overfitting.

Come abbiamo già visto possiamo aggiungere della regolarizzazione (che entrerà nella loss del modello) ai layer per tenere i pesi con valori assoluti bassi.

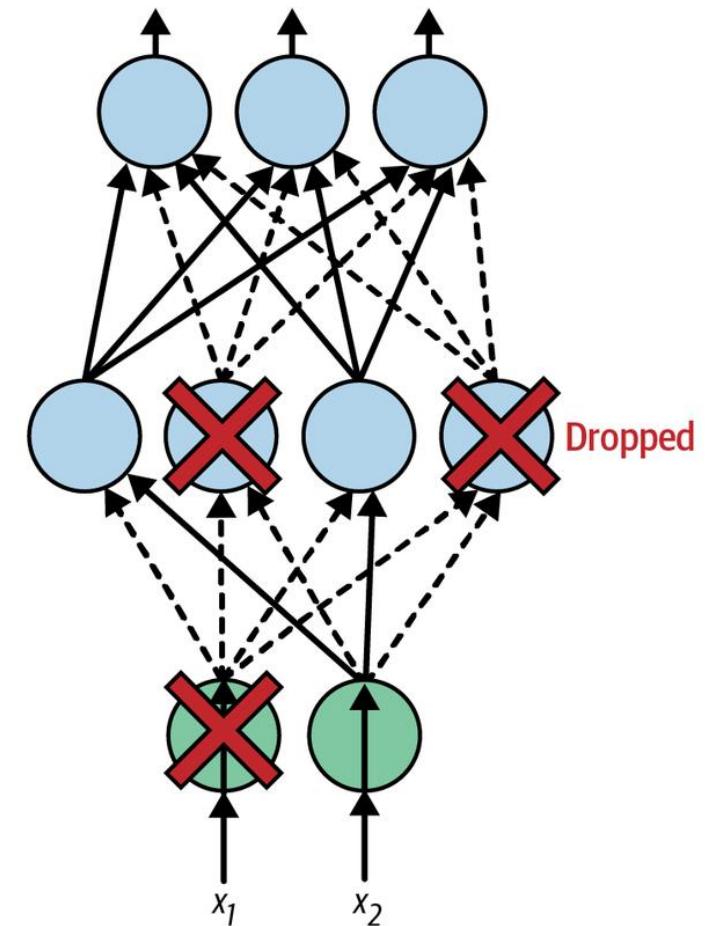
Possiamo anche aggiungere un layer apposta per ridurre il rischio di overfitting.

Dropout

Il layer di **dropout** è un layer che in fase di training «cancella» alcuni neuroni random, mettendo a zero il loro output.

In questo modo il modello è costretto a distribuire la conoscenza tra più neuroni non avendo la possibilità di sfruttare sempre le stesse features per prendere la decisione. Per ogni layer specificheremo un *drop-rate* percentuale di numero di neuroni a zero.

"Would a company perform better if its employees were told to toss a coin every morning to decide whether or not to go to work? The company would be forced to adapt its organization; it could not rely on any single person to work the coffee machine or perform any other critical tasks, so this expertise would have to be spread across several people."

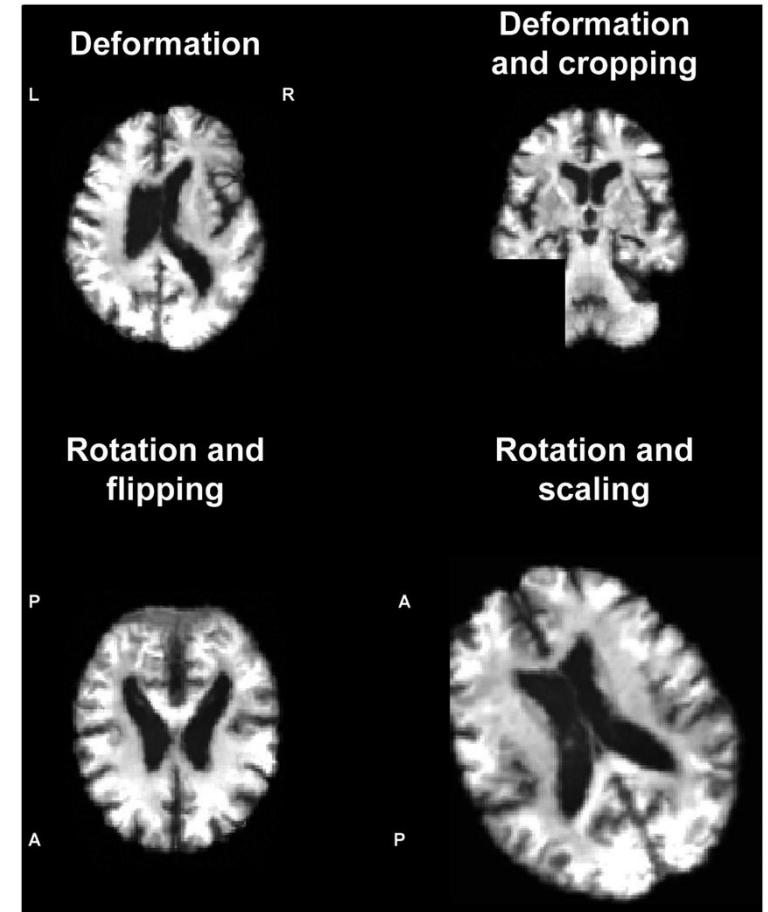


Data augmentation

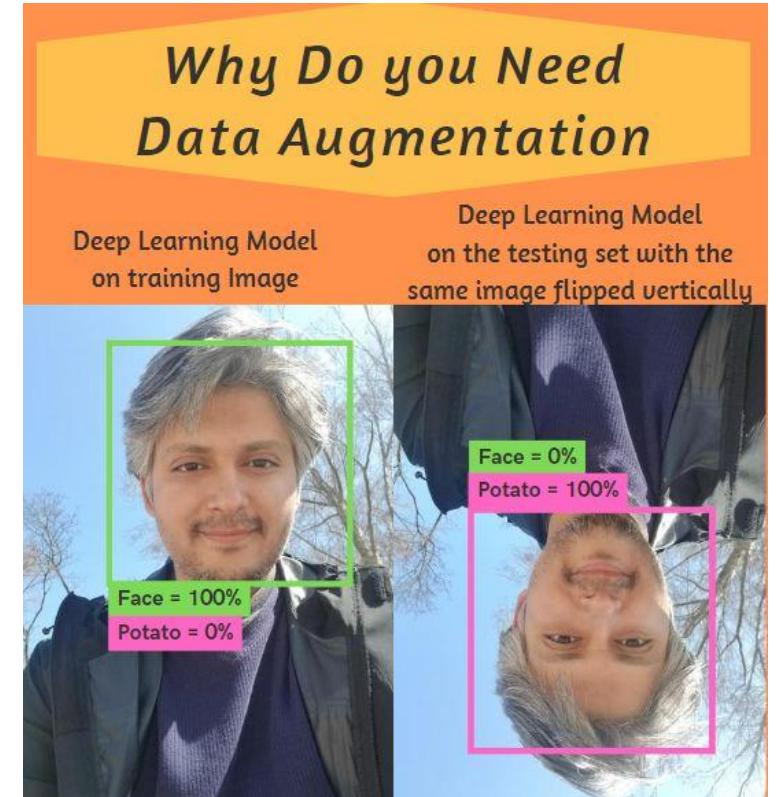
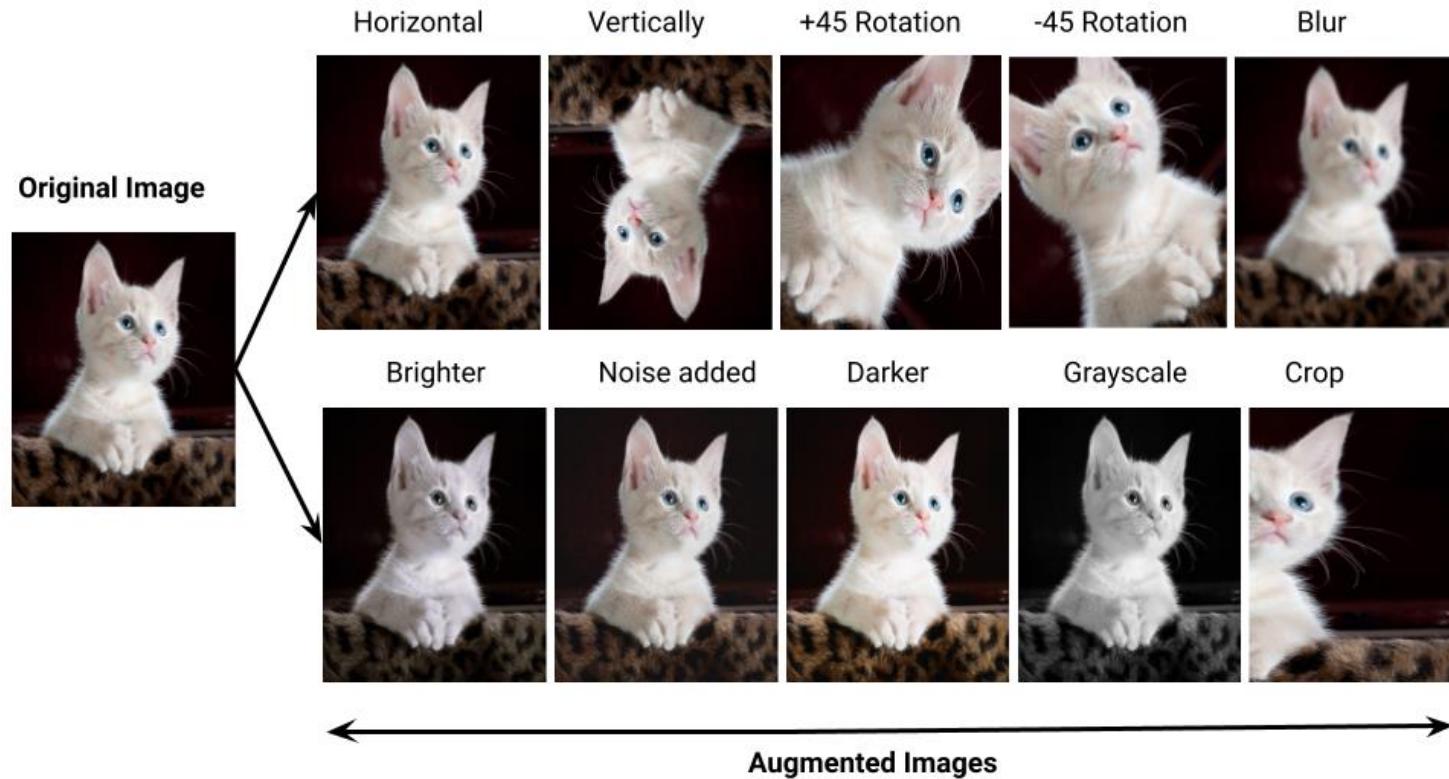
La data augmentation è il processo per cui andiamo a creare dei nuovi esempi «sintetici» a partire da quelli reali distorcendoli opportunamente in modo che l'informazione contenuta sia comunque rilevabile.

Nel caso delle features numeriche spesso si sommano noise Gaussiani che hanno una varianza comunque più bassa rispetto alla differenza tra gli esempi. Nel caso delle immagini possiamo applicare tantissime altre trasformazioni (prospettiche, rotazioni, noise, blur, esposizione, ecc.)

In questo modo possiamo moltiplicare i nostri dati in ingresso rendendo il modello più robusto a nuovi dati.



Data augmentation

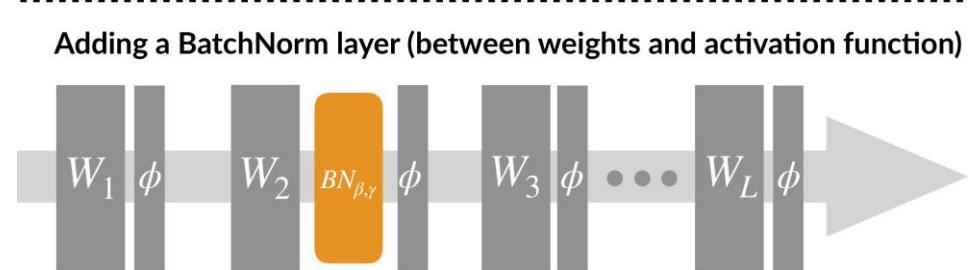
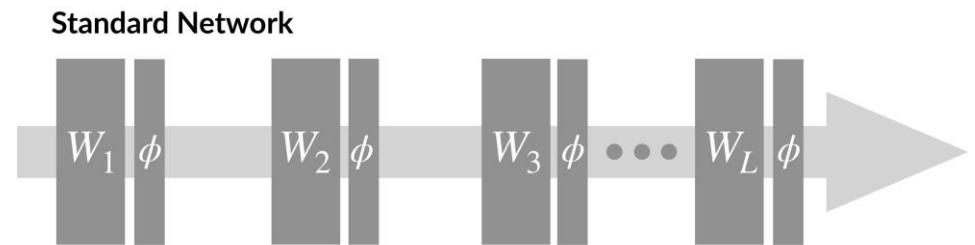


Batch normalization

Un altro layer utile all'apprendimento, anche se non previene l'overfitting, è il Batch Normalization.

Dove inserito prende tutte le features predette dai neuroni del layer precedente e normalizza gli output rispetto al singolo batch di addestramento, riscalando i valori in modo che si distribuiscano come una normale.

Si è dimostrato nel tempo che questo favorisce molto il tempo di apprendimento e la capacità del modello di convergere.



Layer funzione

Ci sono altri layer che non hanno impatto sulle performance del training o del modello ma che sono utili per costruire modelli più complessi.

Alcuni di questi layer sono

- **Flatten**, ovvero un layer che preso un array n-dimensionale in ingresso lo riduce ad un vettore
- **Concatenate**, che prende due o più array n-dimensional e li concatena lungo un asse
- **Maximum, Minimum, Average**, ovvero layer che di tutte le features estrae un solo valore
- **Reshape**, per cambiare la struttura di un array-ndimensionale

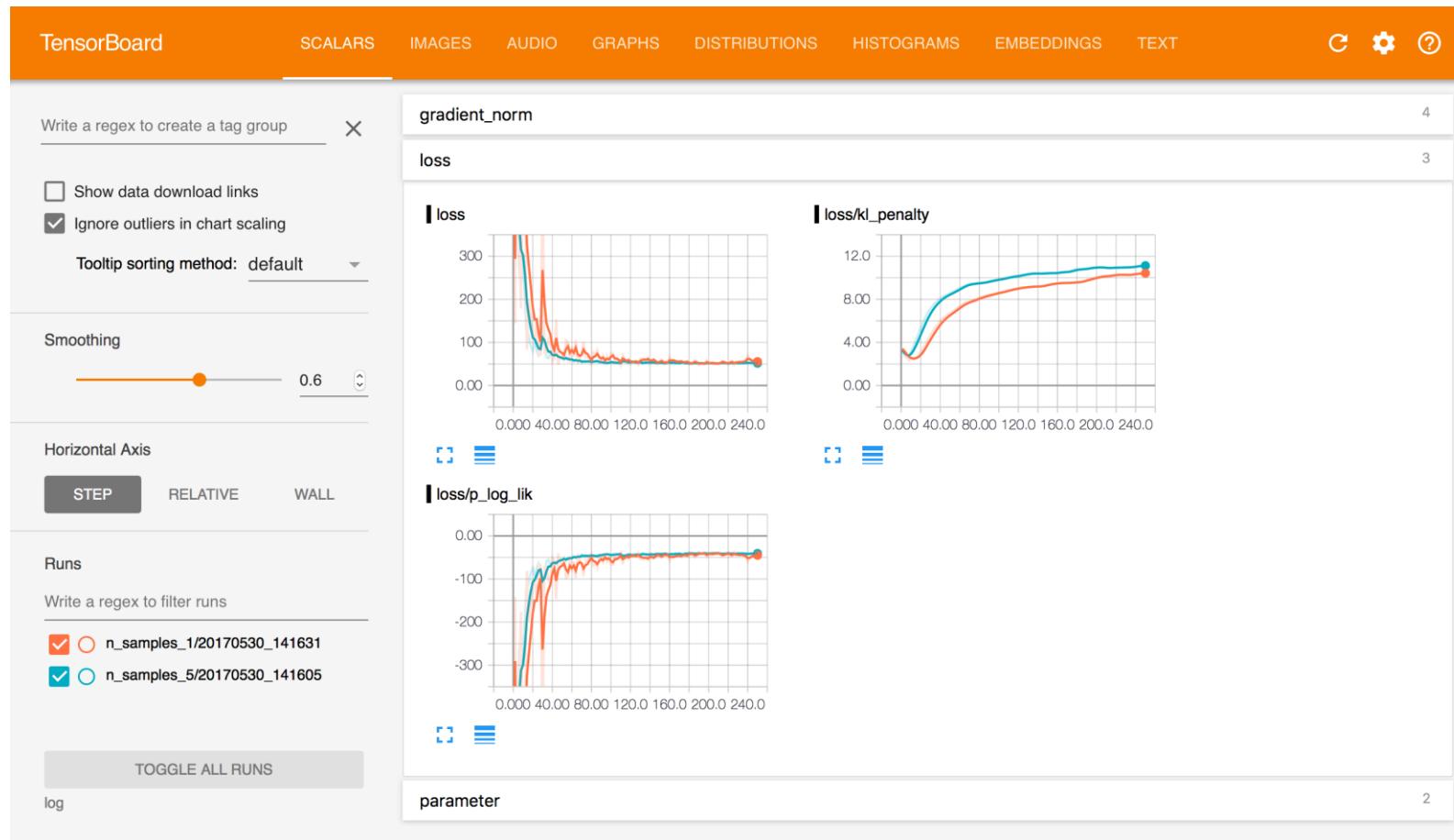
Callbacks

Infine, una volta lanciato il training di un modello possiamo eseguire, al termine di ogni epoca, delle callbacks, ovvero funzioni che possono aiutarci.

Tra queste

- **EarlyStopping**, una callback che ferma l'addestramento quando osserva che la loss tra training e validazione si scosta troppo e quindi che potrebbe aver iniziato ad overfittare il modello
- **ModelCheckpointer**, una callback che ci salva il modello, ad esempio ogni volta che facciamo uno score migliore in validazione
- **Tensorboard**, una callback comoda per loggare tutti i dati di training e le caratteristiche del modello per poterle osservare in tensorboard

Tensorboard



Recap

Abbiamo visto

- Il ciclo di training custom
- Alcuni layer funzione che possiamo utilizzare
- Come prevenire l'overfitting
- Le callbacks

Exe w/ Tensorboard



11 – Convolutional Neural Networks

Daniele Gamba

2022/2023

Lezione precedente

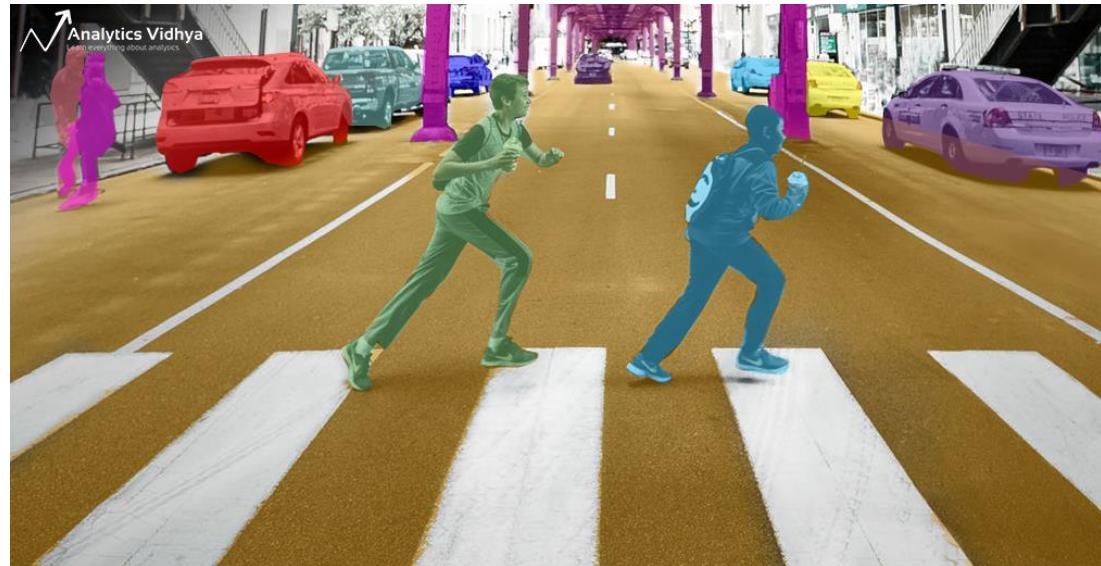
Abbiamo visto come si struttura un MLP

- Sequenziale e non
- Con input e output multipli
- Con diversi layer di uscita
- Con Residual Blocks
- Il ciclo di training custom
- Alcuni layer funzione che possiamo utilizzare
- Come prevenire l'overfitting
- Le callbacks

Altri problemi

Nei problemi che abbiamo visto finora abbiamo sempre avuto un dataset abbastanza tradizionale, ovvero una tabella di input X (N_esempi, N_features) e una o più uscite y.

Ma se i nostri dati fossero delle immagini? O dei segnali nel tempo senza un inizio e una fine precisi?



MNIST

Il "Modified National Institute of Standards and Technology database", o MNIST, è un dataset di 70.000 numeri scritti a mano libera.

L'obiettivo era riconoscere automaticamente i caratteri sulle lettere per lo smistamento automatico e sugli assegni.

Yann LeCun e altri riuscirono con successo a raggiungere una performance superiore al 99% nel 1999.



Exe

Proviamo ad affrontare il problema del MNIST con le tecniche di machine learning che conosciamo

https://knowyourdata-tfds.withgoogle.com/#dataset=mnist&tab=STATS&select=default_segment.mnist.label.value

<https://colab.research.google.com/drive/1ynkAvgUh2KfHrFBGTvEUifrwkEfMo6Wu#scrollTo=q6oosyU-8e6S>

Informazione locale

Come abbiamo visto è molto difficile costruire un algoritmo che prenda in ingresso delle immagini rispetto alle tecniche che conosciamo.

Le features di un'immagine, il valore di ogni pixel, ha un significato solamente se contestualizzato rispetto ai suoi pixel vicini.

L'informazione è quindi locale e circoscritta, solamente guardando l'immagine nel suo insieme riusciamo a comprenderne il significato.

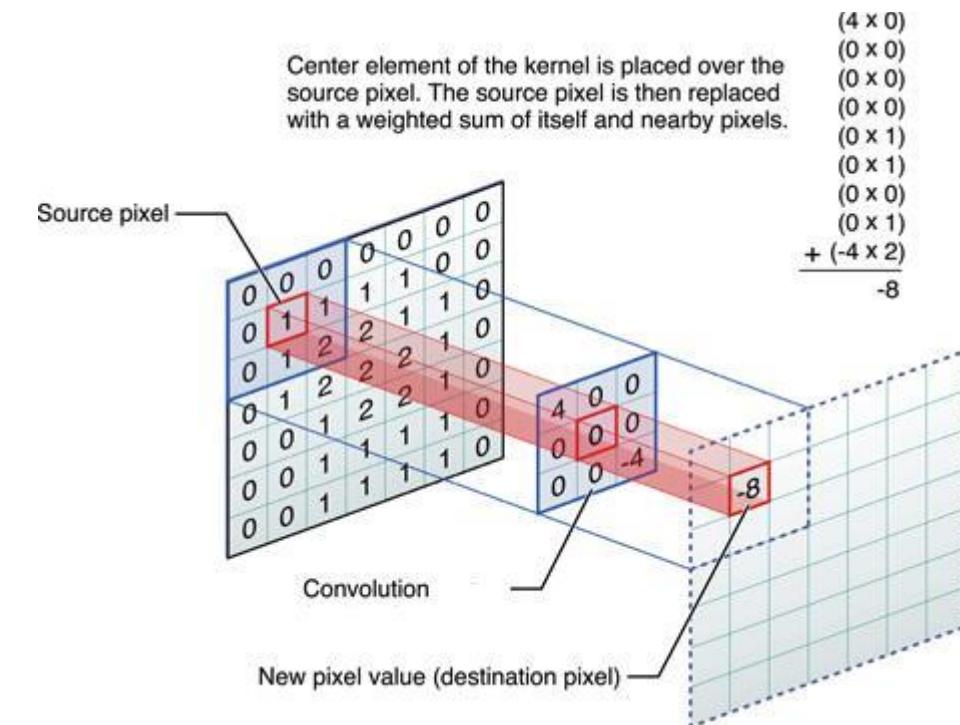
Convoluzione

La convoluzione è l'operazione matematica di far scorrere due vettori uno rispetto all'altro.

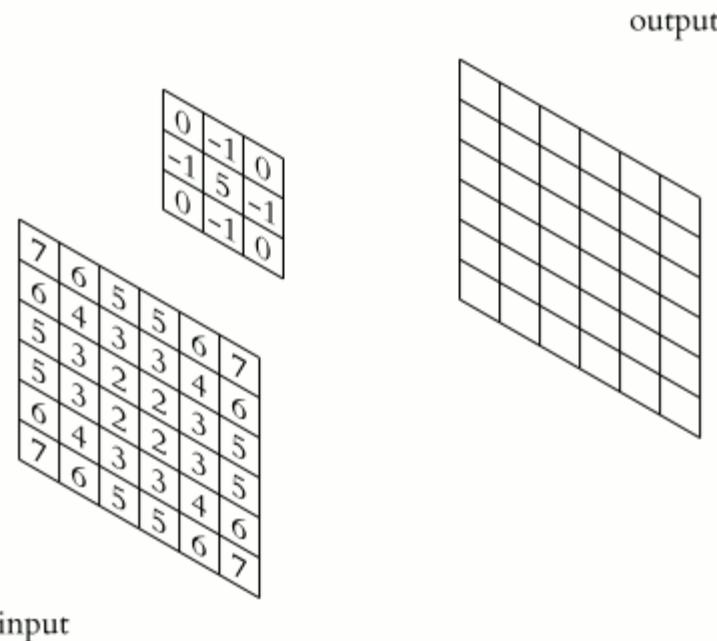
Nel caso di immagini, l'idea è di far scorrere una matrice più piccola, chiamata **kernel**, su tutta la nostra immagine.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau$$

To **convolve a kernel** with an **input signal**:
flip the signal, move to the desired time,
and accumulate every interaction with the kernel



Convoluzione



$$\begin{matrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} 6 & & \\ & & \\ & & \end{matrix}$$

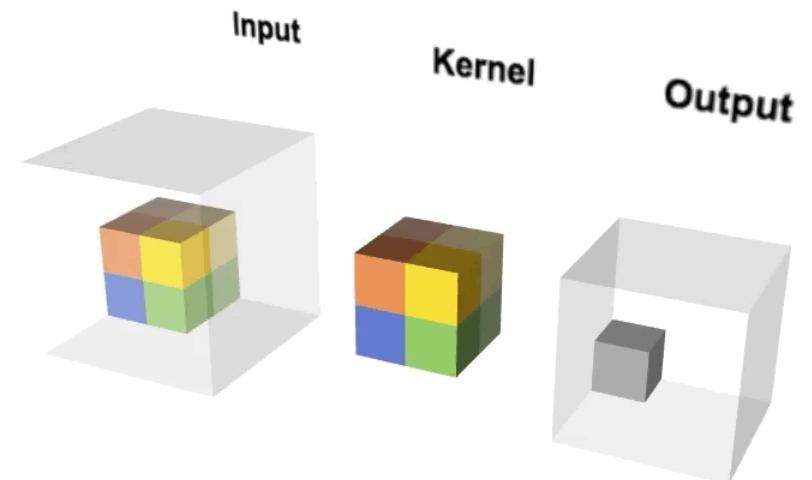
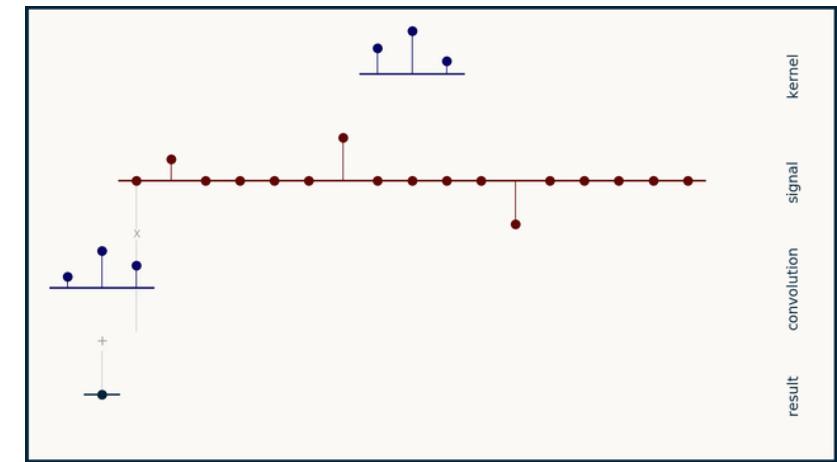
$$7 \times 1 + 4 \times 1 + 3 \times 1 + 2 \times 0 + 5 \times 0 + 3 \times 0 + 3 \times -1 + 3 \times -1 + 2 \times -1 = 6$$

Convoluzione

La convoluzione può essere 1D, 2D o 3D.

I valori che assume il kernel che scansiona l'ingresso sono i parametri della nostra funzione.

Gli iperparametri sono diversi, il primo è sicuramente la **dimensione della matrice** di kernel.

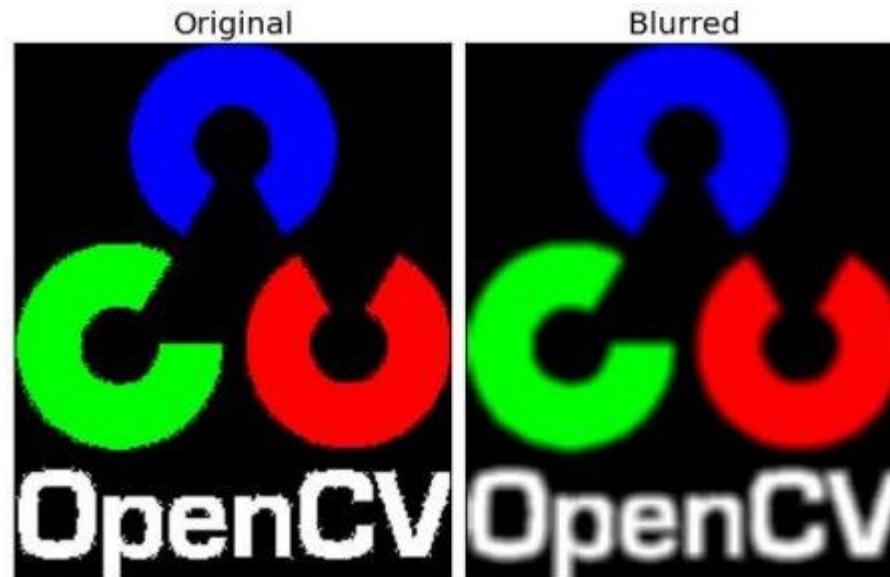


Convoluzione tradizionale

Spesso i kernel delle convoluzioni si chiamano anche *filtri* perché nella computer vision tradizionale esistono e si applicano da molto prima del deep learning convoluzioni per elaborare le immagini.

A seconda dei valori e delle dimensioni che si assegnano al kernel possiamo ottenere risultati diversi.

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Convoluzione tradizionale

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

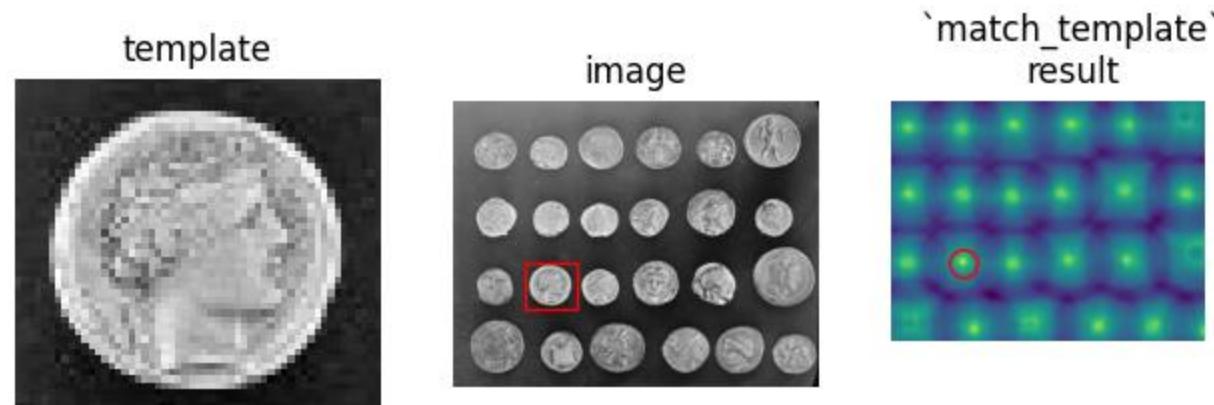
$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



Template matching

Il template matching è un problema di computer vision tradizionale in cui bisogna ritrovare un template all'interno di una immagine più grande.

È un problema notoriamente difficile e l'approccio tradizionale prevede la convoluzione del nostro template su tutta l'immagine. In ogni posizione si calcolerà un indice di correlazione e al termine verrà valutato il massimo ottenuto.



Template matching

Come per il MNIST, il template matching tradizionale soffre terribilmente variazioni di

- Scala
- Forma (e calligrafia)
- Prospettiva
- Rotazione

Per questo hanno preso sempre più piede le tecniche di deep learning che riescono ad interpretare meglio le features locali e ricostruire comunque la presenza del nostro oggetto di interesse.

Conv2D

Tornando al nostro kernel di convoluzione nelle reti neurali, possiamo definire un layer Conv2D che definisce

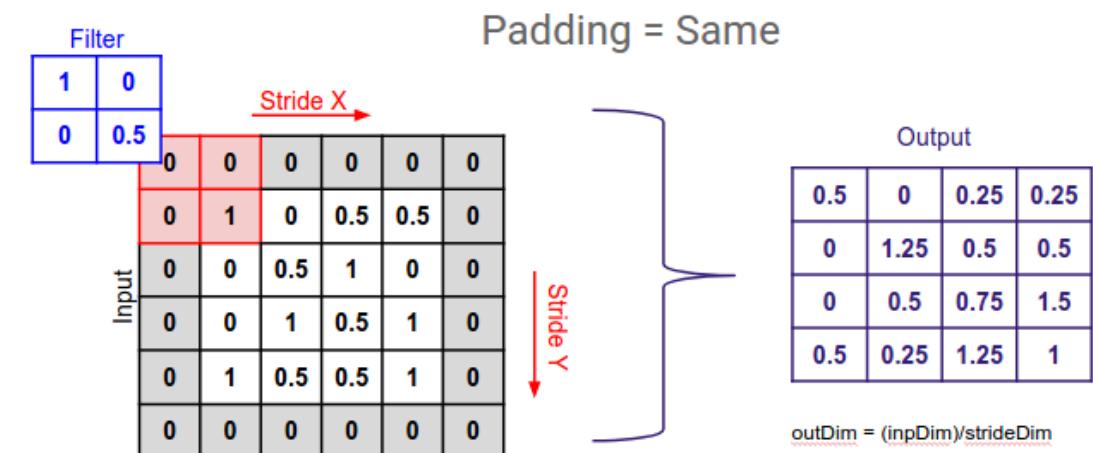
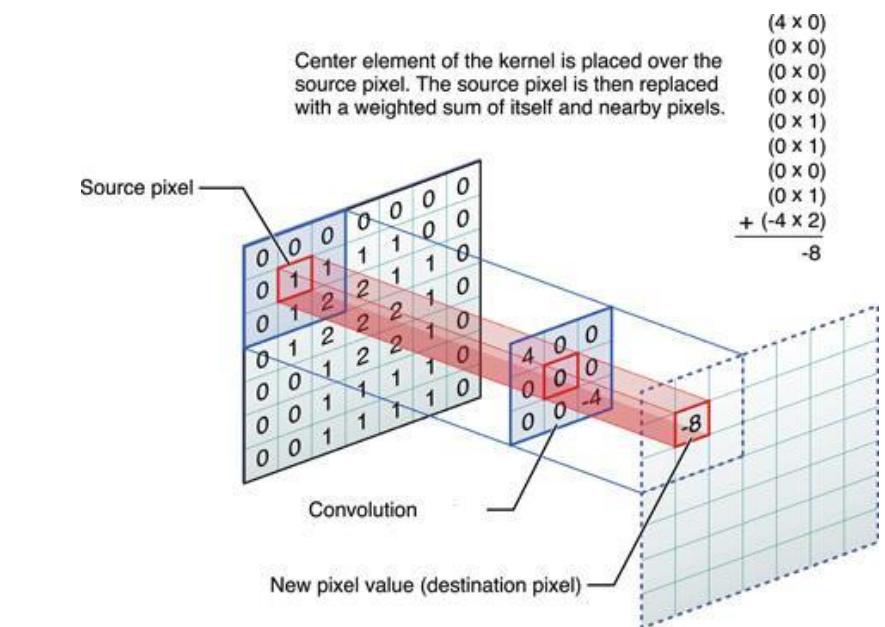
- La dimensione dei nostri kernel
- Il numero di kernel che vogliamo applicare
- La funzione di attivazione da applicare eventualmente dopo il prodotto scalare
- Un insieme di altri iperparametri che andremo a vedere

Padding

Dato che la convoluzione parte «all'interno» della nostra immagine, la matrice in uscita avrà ogni dimensione che è di $2 * (\text{lato kernel} // 2)$ in meno.

Questo comporta spesso delle dimensioni dei nostri layer della rete molto strane e difficili da dividere. Per questo si usa aggiungere intorno alla matrice del **padding**, ovvero dei bordi di zeri che consenta alla convoluzione di ottenere delle dimensioni più trattabili.

Usando padding='same' otterremo in uscita un output che ha le stesse dimensioni della nostra matrice in ingresso.



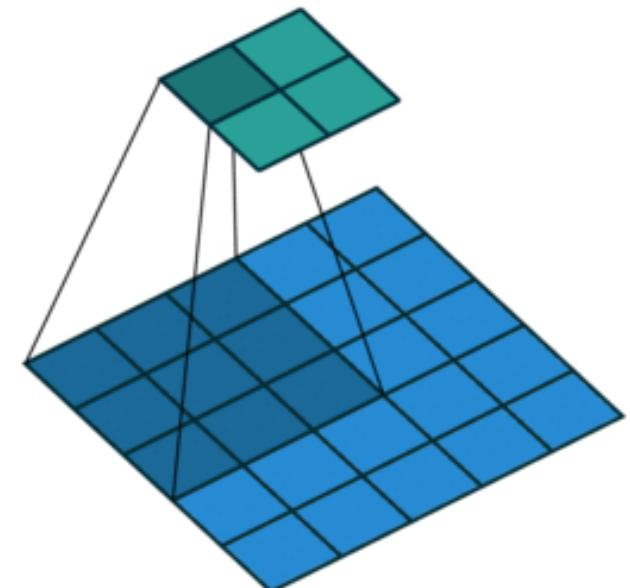
Stride

Lo **stride** è il numero di celle con cui ci spostiamo ad ogni convoluzione.

Se volessimo scansionare perfettamente tutta l'immagine, con stride=1 il nostro kernel si sposterà su tutti i pixel dell'immagine.

Con stride pari a 2 invece di spostarci di una sola casella ci sposteremo di 2, e così via.

Spostandoci di più di una casella per volta il nostro output avrà dimensioni di uscita più piccole rispetto all'ingresso anche in presenza di padding.

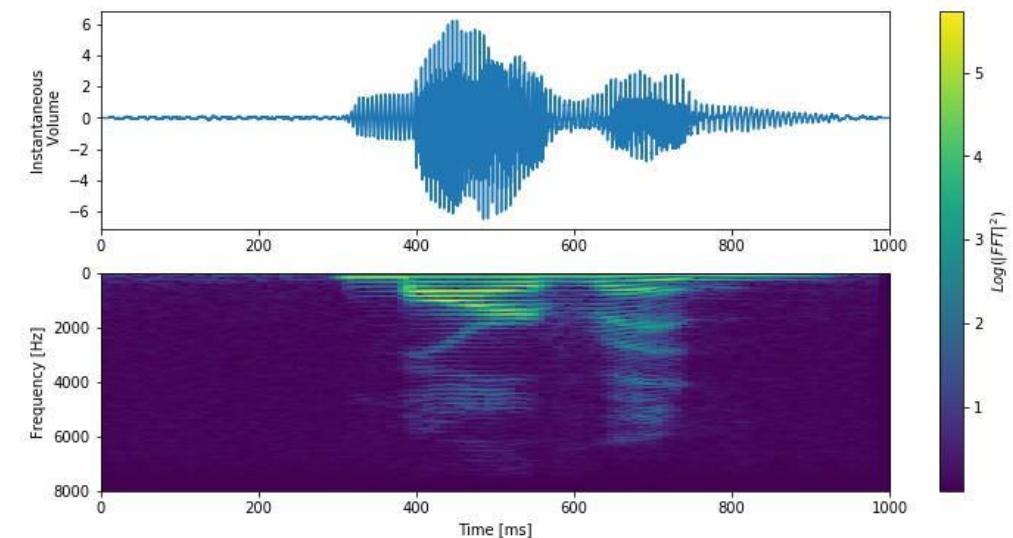
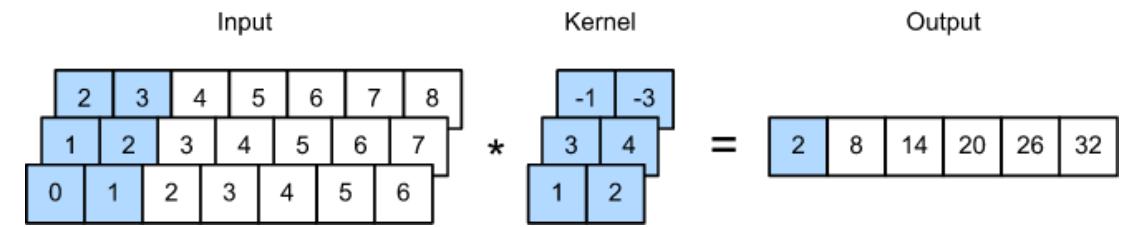


Conv1D

La convoluzione 1D si usa normalmente per le serie temporali.

La convoluzione ci consente di intercettare features locali tipiche dei segnali nel tempo.

Inoltre possiamo applicare una convoluzione 1D anche su spettrogrammi andando ad applicare kernel alti tanto quanto le nostre features e che si muove in direzione del tempo.

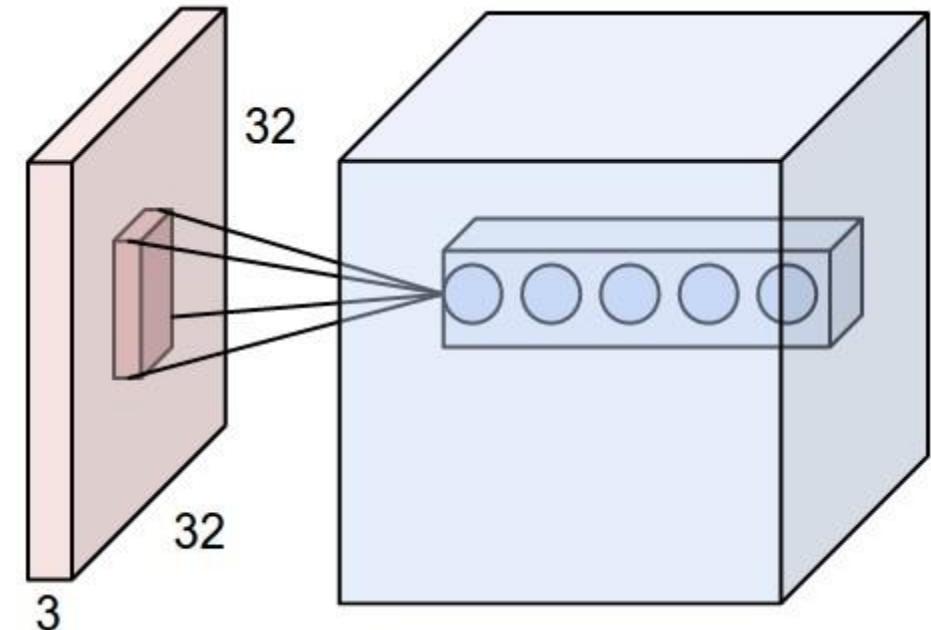


Conv2D

Una volta definito il nostro layer, iperparametri e la funzione di attivazione, il numero di kernel che abbiamo deciso proietterà N diverse elaborazioni in un ipercubo.

Il nostro kernel sarà profondo tanto quanto il layer in ingresso.

Saranno tutte rappresentazioni con lo stesso livello di dettaglio della nostra matrice in ingresso (eventualmente sottocampionato con lo stride), ma ci serve un altro strumento per ridurre la dimensionalità e poter interpretare meglio le features.



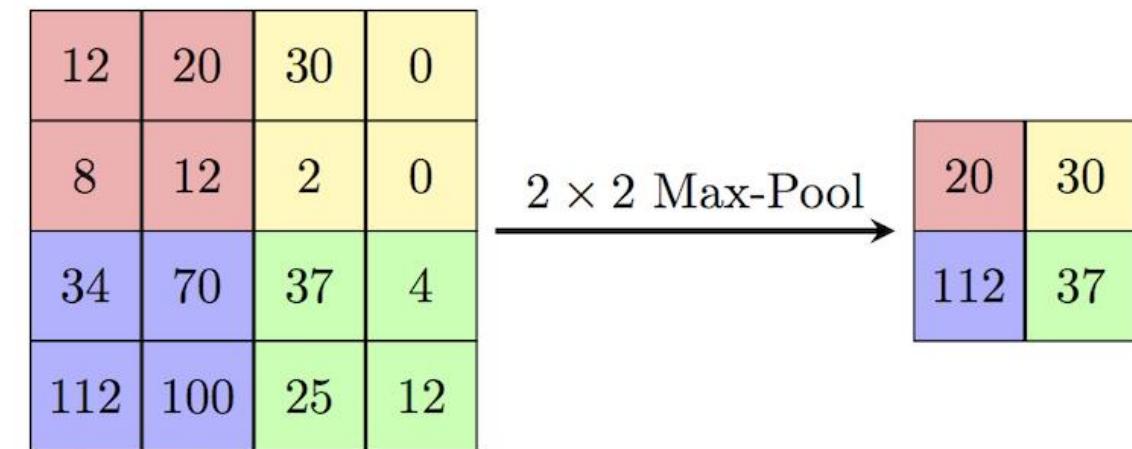
Pooling

Una volta elaborate le nostre features ci serve un modo per ridurre la dimensionalità del nostro layer per poter interpretare a più alto livello le nostre features.

Per questo esiste il layer di **pooling**, in cui andiamo a prendere il massimo, minimo, la media, ecc. ogni x celle per costruire una matrice che ha dimensione divisa per x .

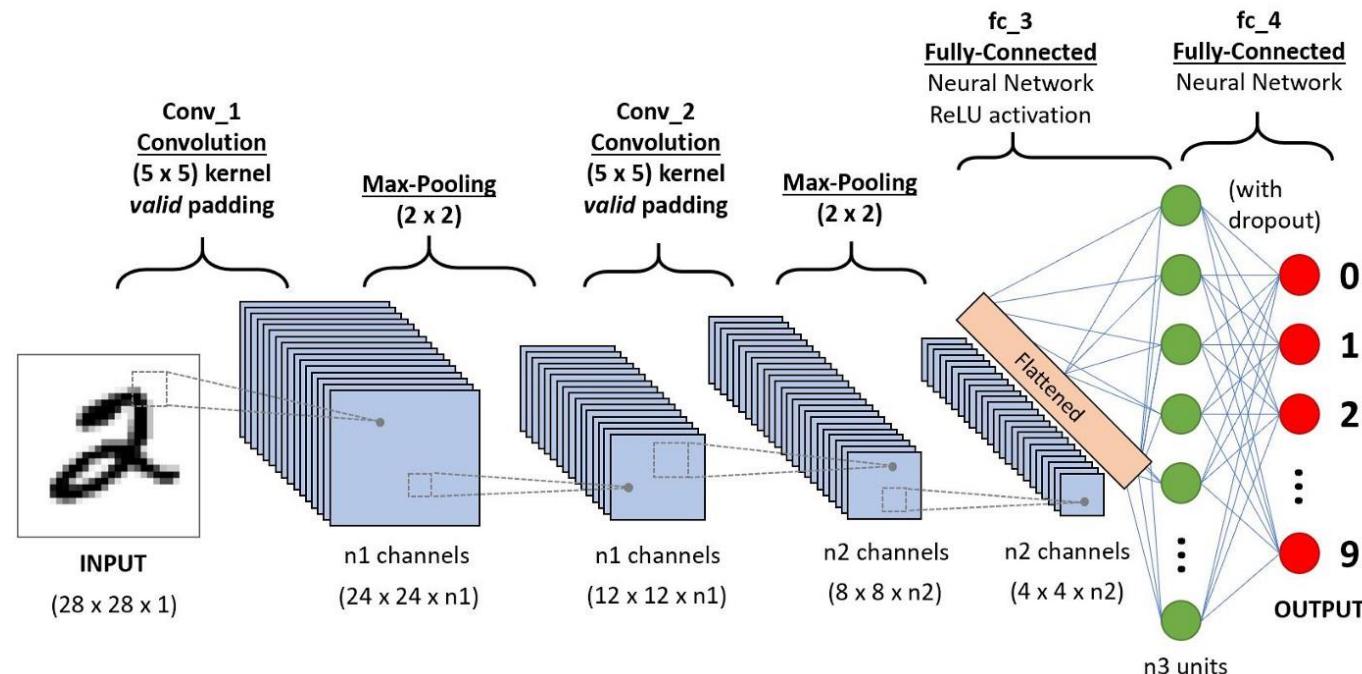
In questo caso con un Max-Pool 2x2 andiamo a prendere il massimo ogni 2x2 celle e costruiamo il nostro output. Da una matrice 4x4 otteniamo una nuova matrice 2x2.

Proprio per l'applicabilità del pooling si cerca di avere matrici divisibili per multipli di 2, eventualmente usando il padding nella convoluzione.



CNN

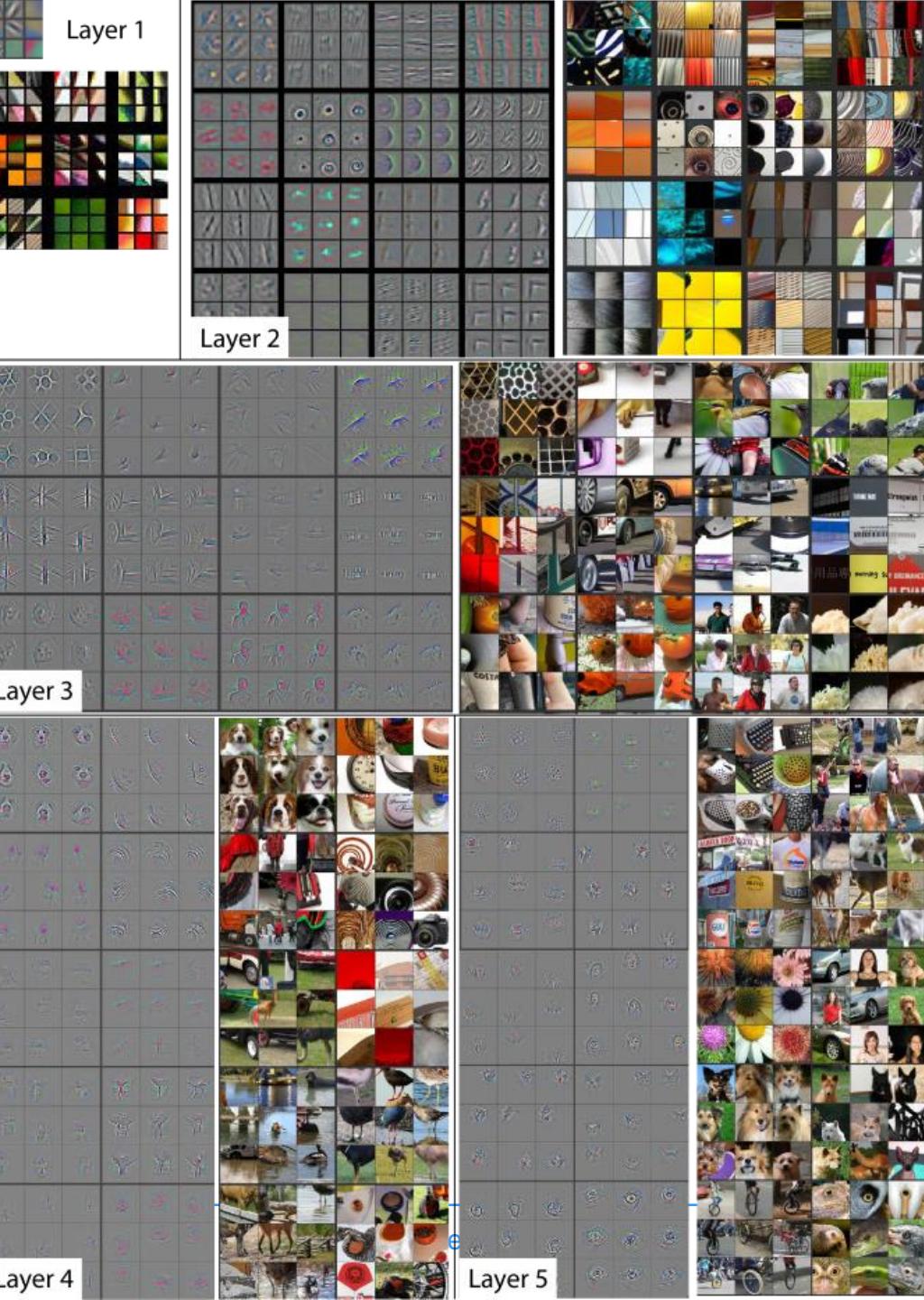
Una Convolutional Neural Network è quindi una struttura che sfrutta layer di convoluzione ed eventuali layer di pooling per eseguire diversi task. Normalmente si tende a **ridurre la dimensionalità su x ed y** tramite pooling e ad **aumentarla in z** tramite il numero di kernel. In fondo alla rete si procederà poi a fare un Flatten del risultato e ad applicarvi dei layer densi per classificare il risultato delle features convoluzionali.



Visualizzazione features

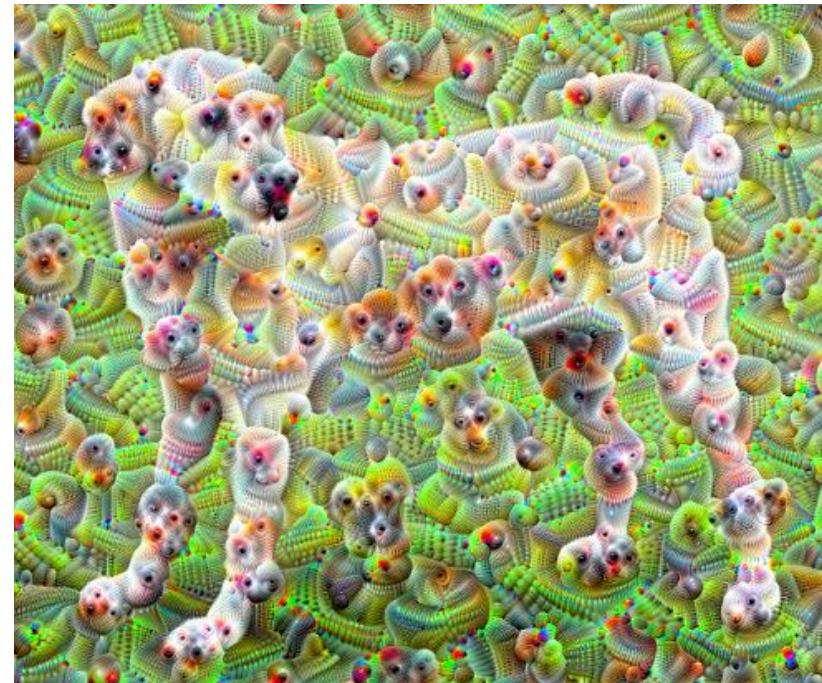
Come abbiamo già visto, intuitivamente nei primi layer si hanno le features più semplici per mettere in mostra linee, tondi, ..., e con il pooling e le convoluzioni successive andiamo ad interpretare combinazioni di queste features che diventano via via più complesse e numerose.

Es. Visualizzare le features intermedie



Deepdream

Deepdream è un paper in cui si è voluto mettere in mostra quali sono i pattern che sono determinanti per una rete nel decidere di che classe è un oggetto e lo fanno riproiettando i pattern nell'immagine stessa. Di fatto è come osservare quello che sta sognando la rete neurale.



Deepdream



Recap

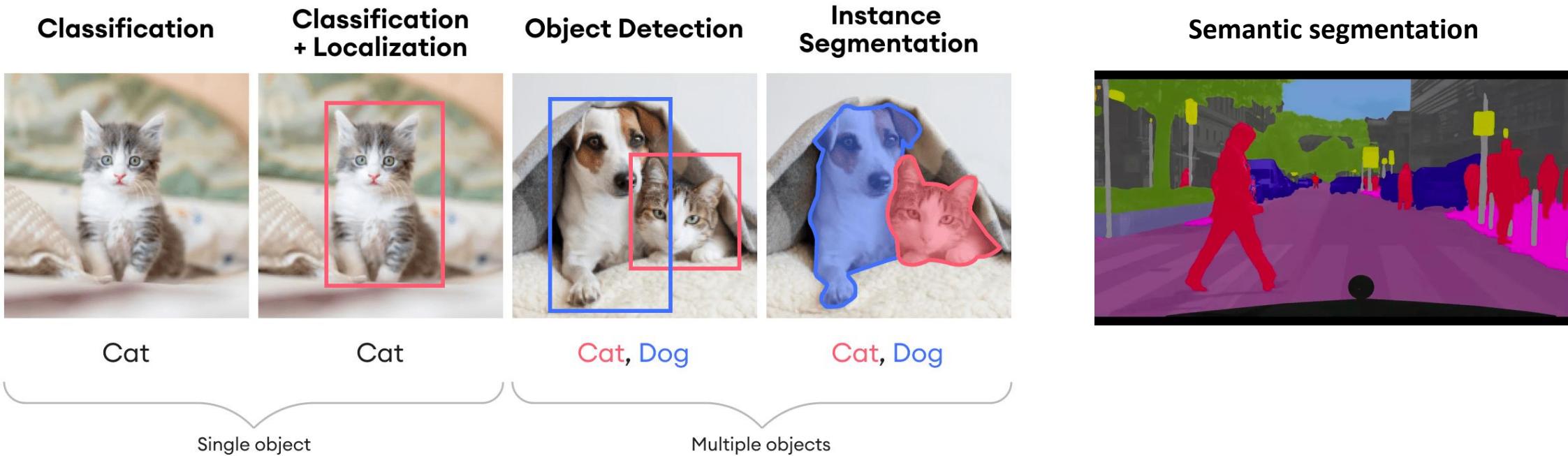
Abbiamo visto come strutturare una CNN tramite

- Convoluzioni
- Padding
- Pooling
- Una architettura standard per la classificazione

Ora possiamo riprovare a classificare il nostro MNIST sfruttando una CNN

Task

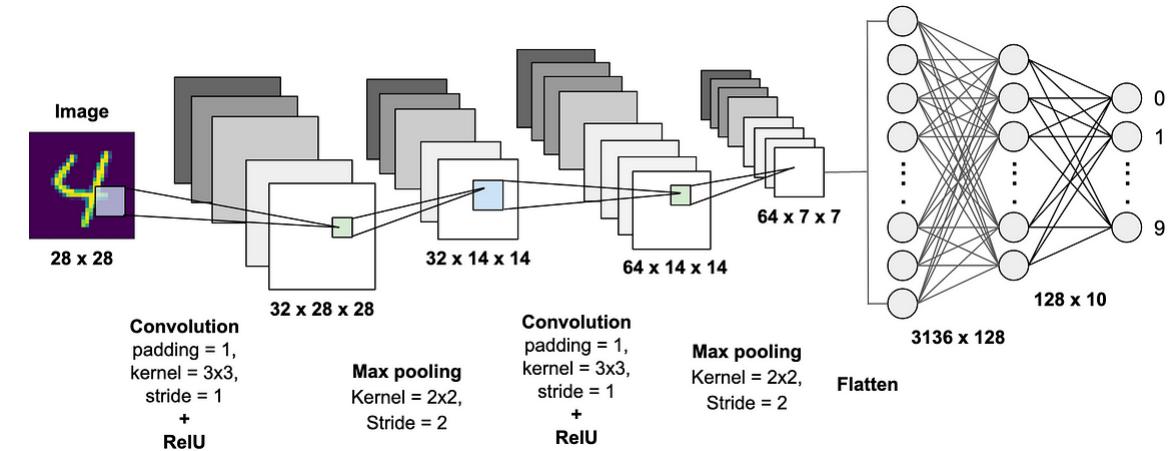
Le CNN sono particolarmente utili per lavorare su dati 1D e 2D per risolvere alcuni tipi di task



Classification

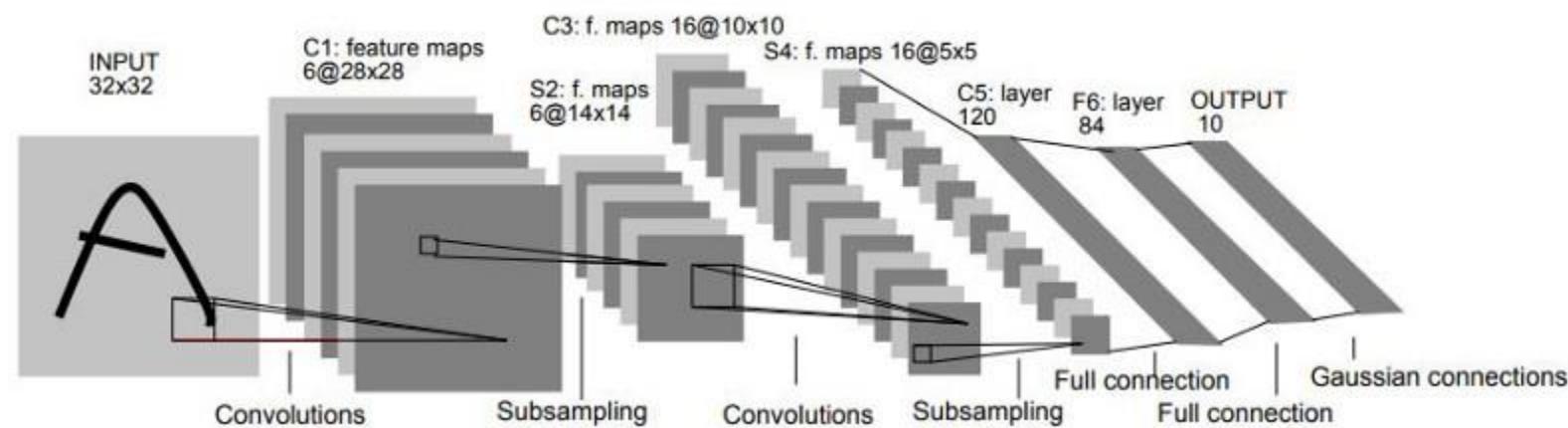
Come abbiamo visto il primo task è quello della classificazione.

Data un'immagine dobbiamo dire a che classe appartiene. In questo caso la testa della rete sarà un MLP con l'output multiclass ed eventualmente multilabel.



LeNet5

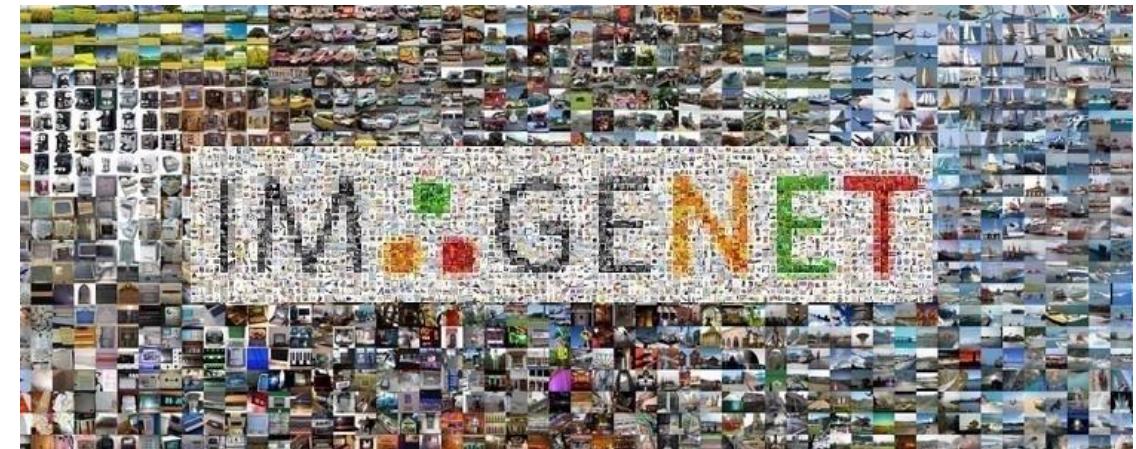
Una delle prime implementazioni di CNN è proprio la LeNet5 di Yann LeCun con la quale ha ottenuto lo 0.8% di errore su MNIST nel 1998. È una semplice CNN che ha formato bene sul task di riconoscimento caratteri ma che non riusciva a scalare su task più complessi.



ImageNet

ImageNet è uno dei dataset più importanti e difficili da affrontare.

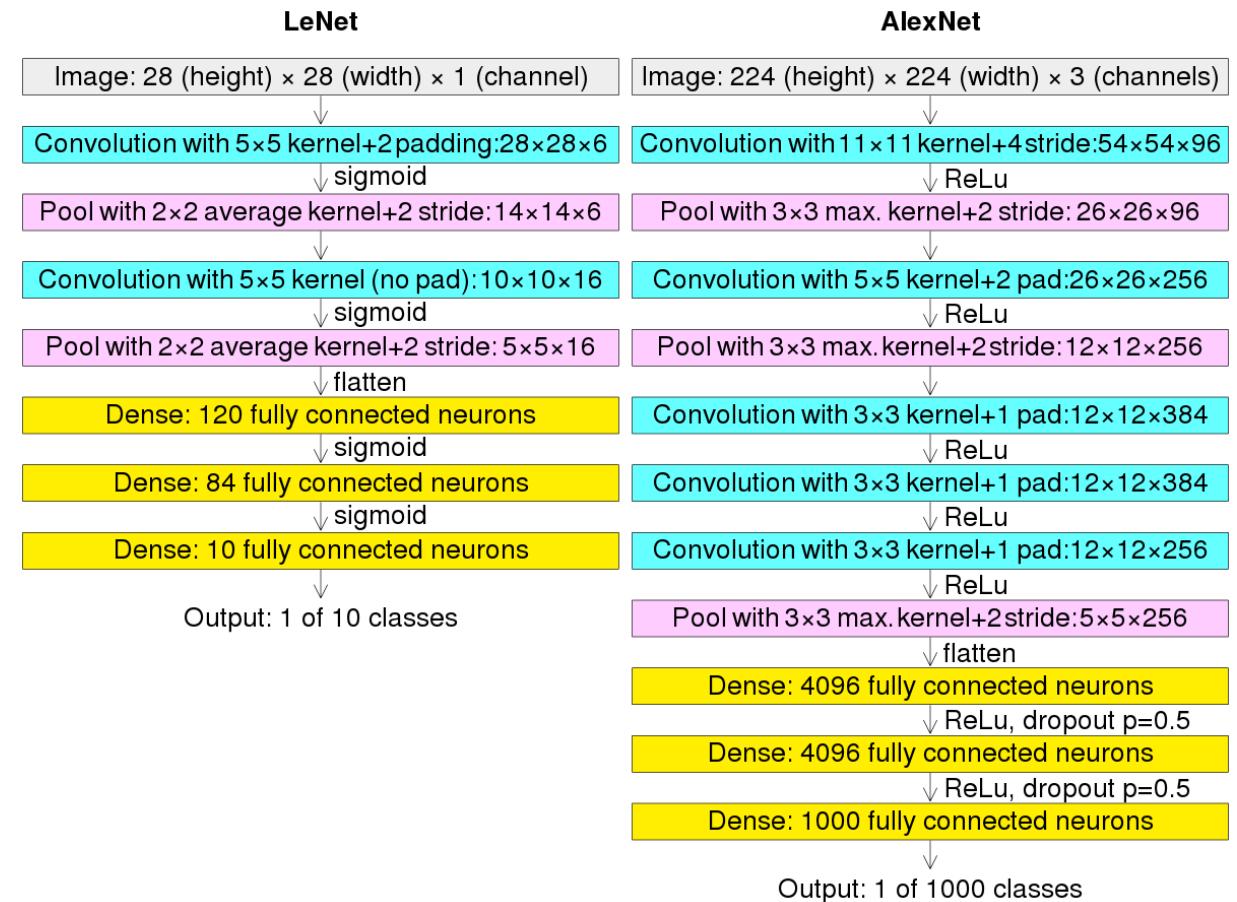
Fino al 2012 la competizione consisteva in 10.000.000 di immagini appartenenti a più di 10.000 categorie.



AlexNet

Nel 2012 la AlexNet ha vinto la competizione con un margine notevole (17% di errore Top-5 contro il 26% del secondo classificato). La rete era semplicemente più larga e profonda oltre ad alcune altre migliorie.

La rete aveva circa 60 milioni di parametri.

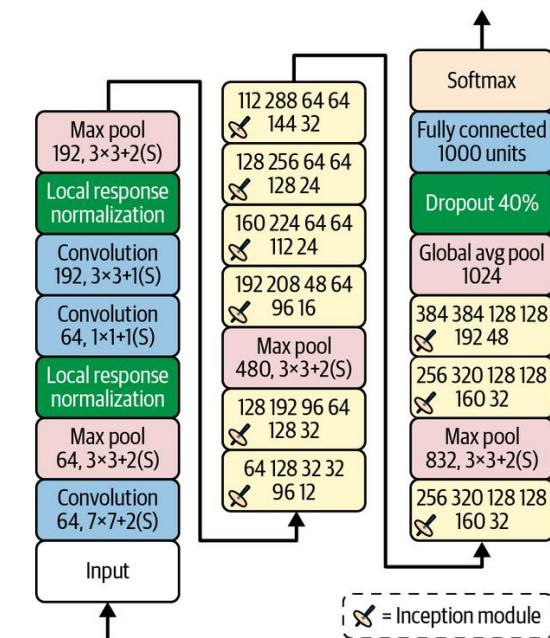
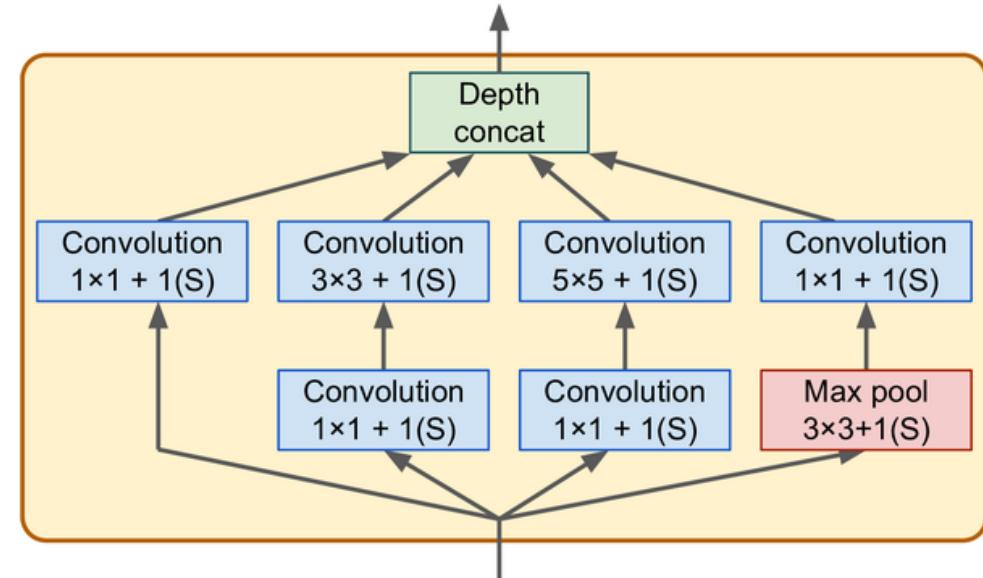


Google LeNet

Nel 2014 Google vince la stessa competizione portando l'errore Top-5 al 7% con una rete 10 volte più leggera della AlexNet (circa 6 milioni di parametri).

La performance è data dal fatto che la rete è molto più profonda e implementa **l'Inception module**.

Grazie a questo modulo si riescono ad imparare features molto più efficienti.

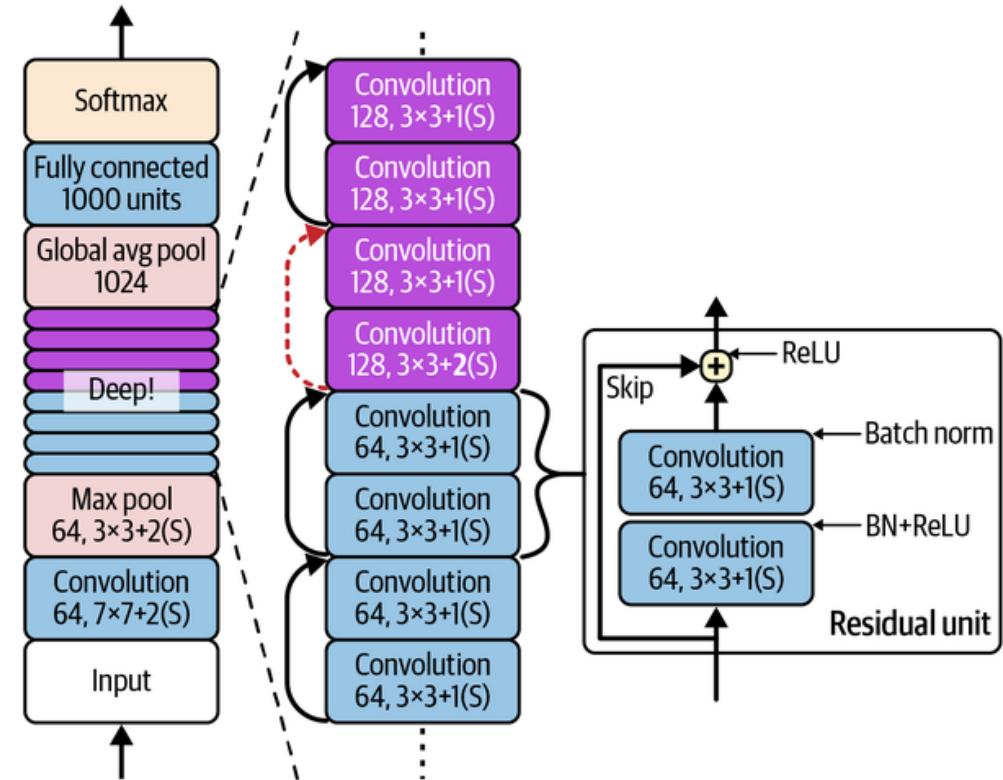


ResNet

Nel 2015 Kaiming He et al. vinsero proponendo la **ResNet** con un errore Top-5 di solo 3.6%.

La ResNet aveva meno parametri ma ben **152 layer**, confermando l'intuizione che diluendo l'informazione su più layer si fosse in grado di interpretare meglio ciò che si stava osservando.

Per addestrare questa rete così profonda servì implementare le **skip-connection** che abbiamo visto nella scorsa lezione.



Altri modelli

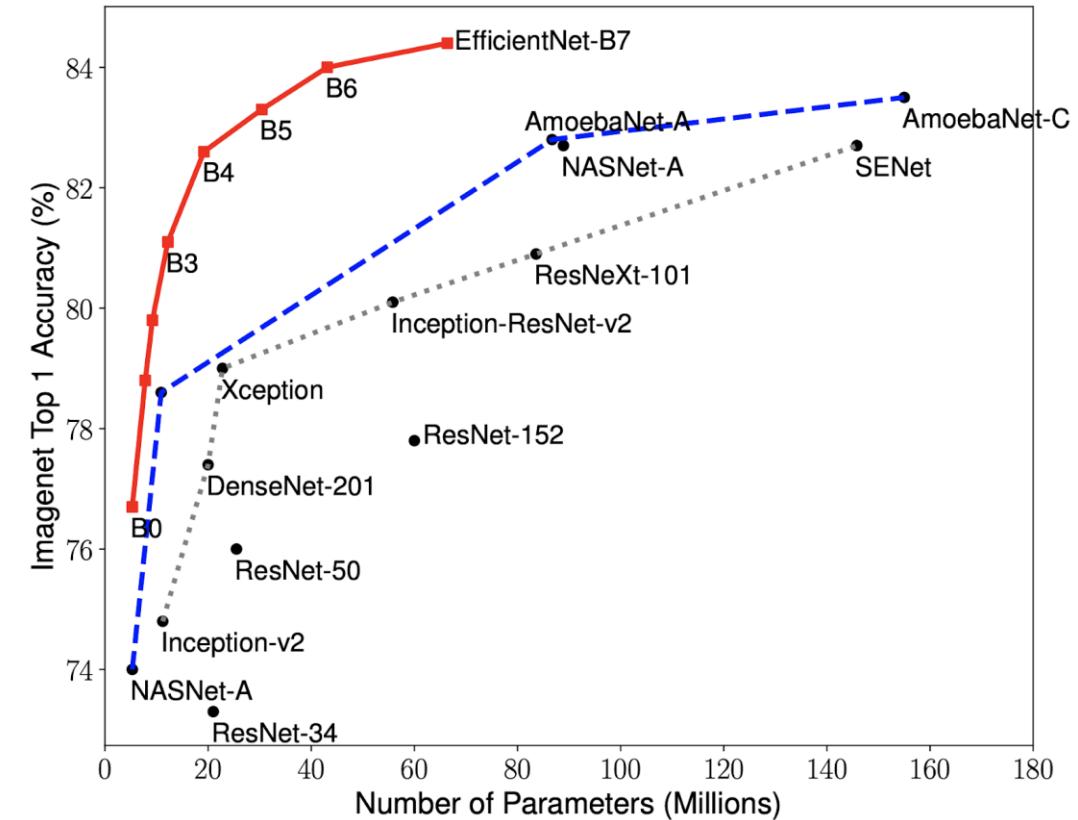
Storicamente ci sono altre reti che sono diventate estremamente popolari

MobileNet

Sono reti pensate per essere eseguite su smartphone, quindi piccole e veloci, ne esistono di diversi tipi ma sono ampiamente usate.

EfficientNet

È una delle reti più efficienti, con performance molto alte su ImageNet addestrata via via su immagini di risoluzione sempre più alta.



Modelli pre-addestrati

Come abbiamo visto la scorsa lezione possiamo usare dei modelli pre-addestrati su un task, prenderne una parte dei pesi e riusarli per la nostra applicazione.

Keras ci mette a disposizione una serie di modelli pre-addestrati per fare **transfer learning**.

Questi modelli sanno già interpretare molto bene le immagini ed estraggono features molto significative. Ci basterà tagliarne gli ultimi layer e aggiungere la nostra testa per classificare sugli oggetti di nostro interesse.

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1

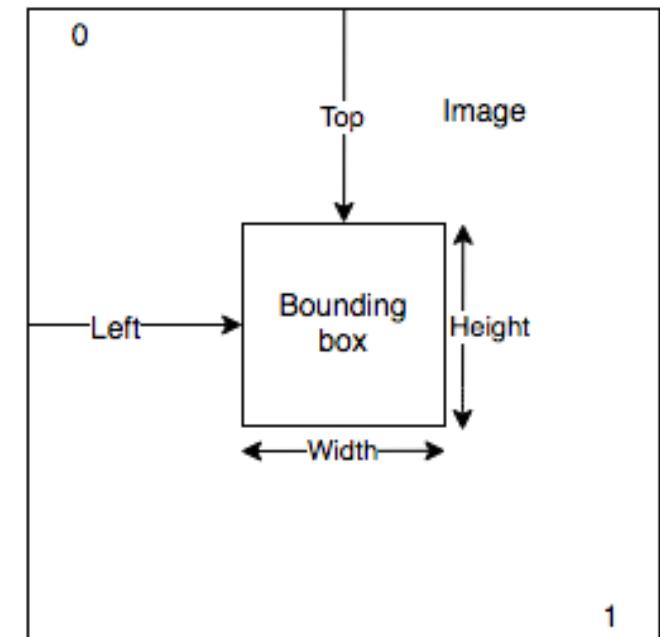
Exe

Prima di procedere proviamo a prendere una CNN pre-addestrata, tagliarne la testa per vedere gli embedding in output e fare classificazione su MNIST classificando gli embedding o facendo transfer learning con una nostra testa.

Object localization

Nel caso volessimo non solo classificare quale oggetto è in una immagine ma avere anche una indicazione di dove si trova possiamo chiedere alla rete di estrarre una posizione ed una scala dell'oggetto. In questo modo la nostra uscita avrà oltre alla probabilità che vi sia l'oggetto anche delle coordinate **x, y, w, h** (width, height).

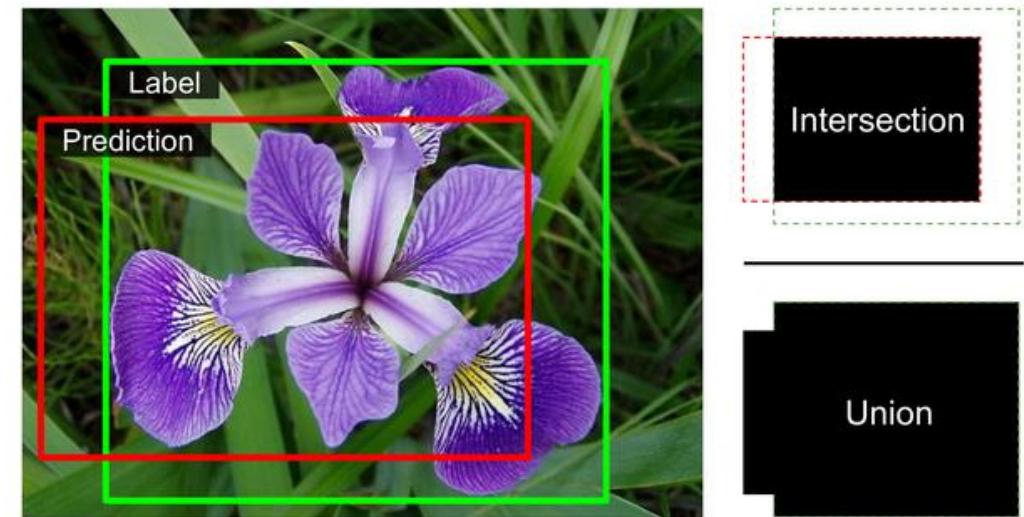
In questi casi avremo quindi delle label che prevederanno anche la **bounding-box** dell'oggetto, ovvero il riquadro che lo racchiude.



IoU

Quando abbiamo delle bounding-box come label non possiamo più usare l'MSE o la cross-entropy come loss, ma ci serve qualcosa che indichi un grado di accuratezza della nostra bounding box.

Per questo è stata introdotta la Intersection-over-Union che indica l'area di intersezione diviso l'area di unione. La perfezione si ha quando le due BB sono perfettamente sovrapposte.



Object detection

Quando nella nostra immagine sono presenti più oggetti vogliamo che per ogni classe venga generata in output la bounding box e la sua confidenza. Non trattando più un solo oggetto ma possibilmente N nell'immagine, bisogna trovare un modo di codificare un numero variabile di oggetti in uscita.

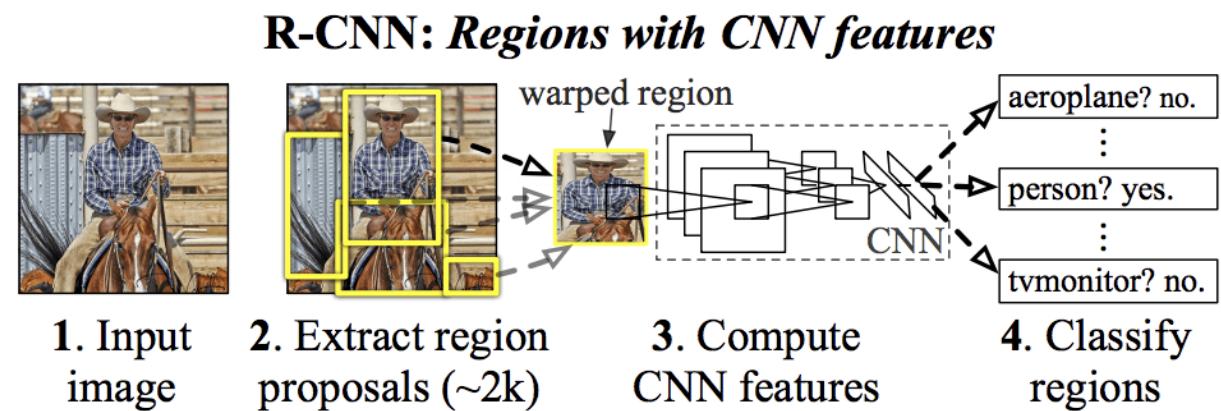
Questo problema è stato risolto in diversi modi da diverse architetture di rete.

R-CNN

Le Region-based CNN, o R-CNN, sono reti molto semplici in 3 passaggi

- Un primo algoritmo molto semplice propone dei possibili ritagli dell'immagine, ovvero fa **region-proposal** (circa 2.000)
- Una CNN estraе delle features deep, nella prima implementazione si è usata la AlexNet
- Un classificatore, tipo SVM, classifica il vettore con le features deep per dire se nella region è presente o meno l'oggetto.

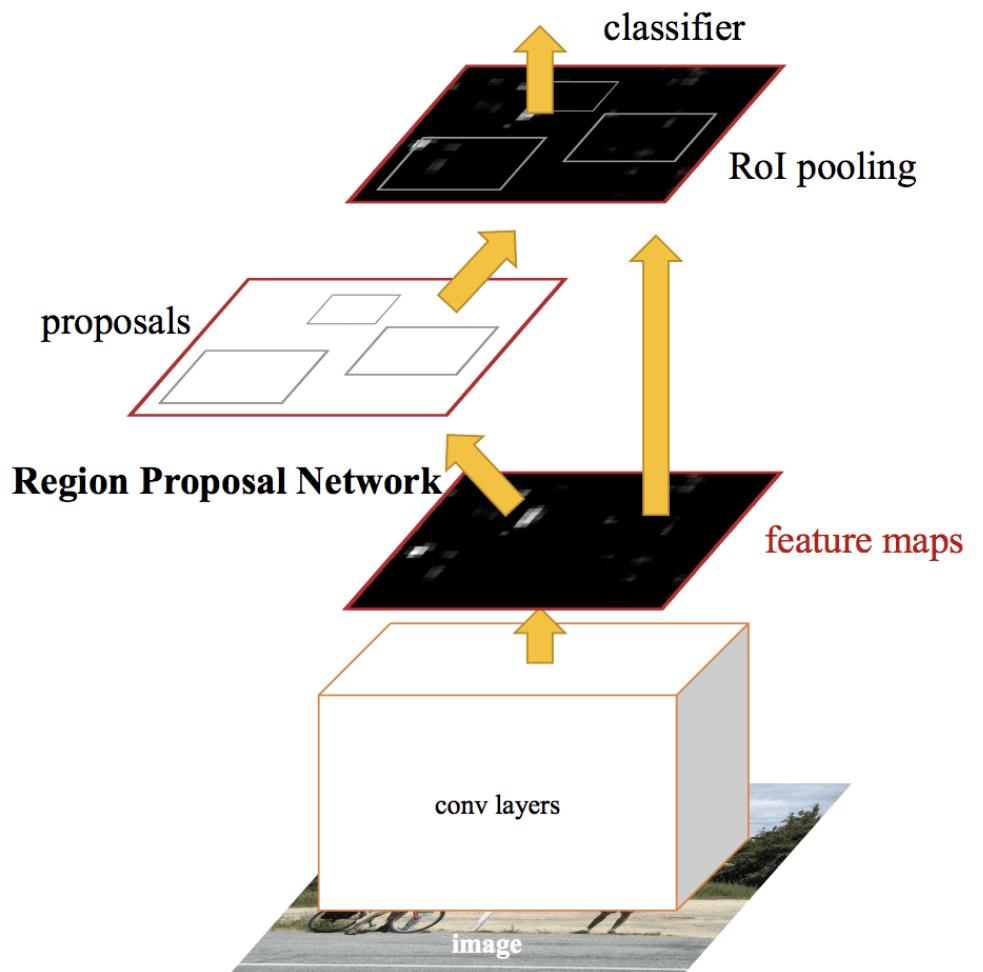
Verrà presa la region, o l'unione delle region, con la confidenza più alta. La stessa tecnica può essere usata per fare object segmentation.



Faster R-CNN

Un'evoluzione della R-CNN è la Faster R-CNN che prevede l'uso di un solo modello per fare i tre step di region-proposal, features extraction e classificazione.

Tramite la RoI Pooling si vanno a selezionare solamente le Region of Interest più promettenti. Usando un modello solo questa tecnica ha ottenuto il primo posto nel 2016 per accuratezza ed è anche molto più veloce ed accurato a proporre le region of interest (circa 300 per immagine).



Yolo

You Only Look One - Yolo, è una delle tecniche di object detection più famose ed utilizzate.

Creata da **Pjreddie** che ha dato anche vita al suo framework (darknet) e a diversi altri algoritmi.

<https://pjreddie.com/>

Il suo CV è preso d'esempio nei corsi di risorse umane

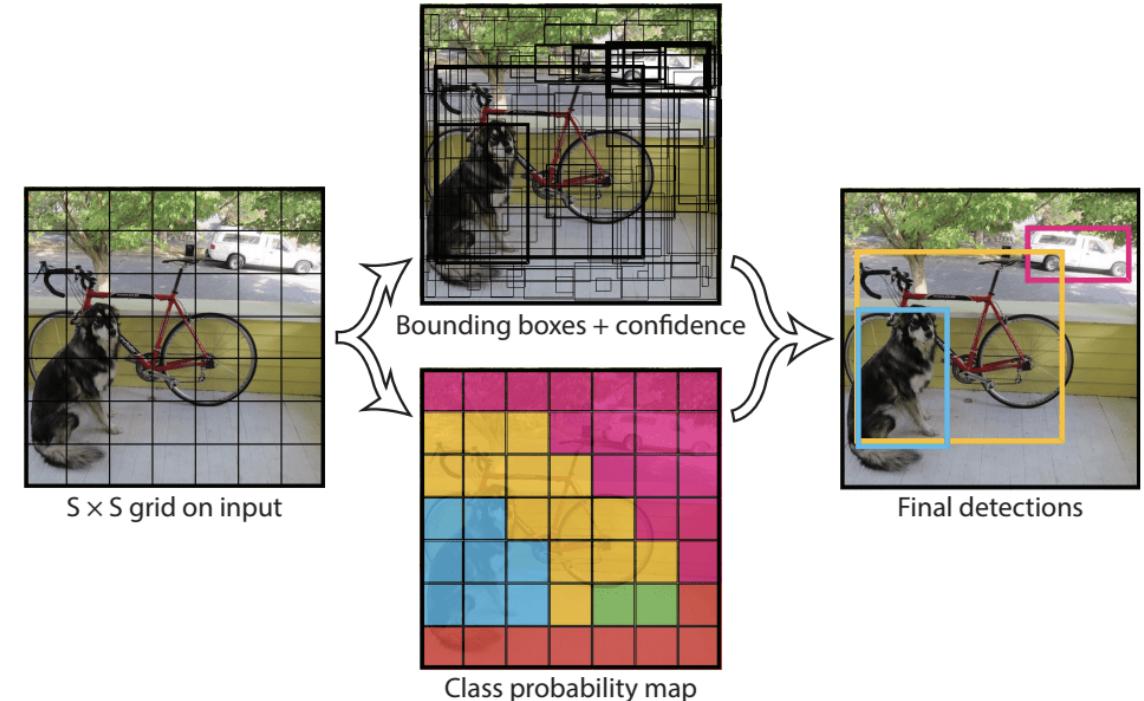
<https://pjreddie.com/static/Redmon%20Resume.pdf>



Yolo

Le reti Yolo sono solitamente più veloci ma meno accurate delle R-CNN o di altri tipi di modelli.

Si basa sul dividere l'immagine in celle, ogni cella è responsabile di classificare un oggetto se il centro della sua bounding box cade al suo interno. Oltre alla classificazione deve anche dire l'offset del centro, altezza, larghezza e confidenza.



Altri modelli

Oltre a Yolo e Faster R-CNN ci sono molti altri modelli più recenti che si usano per fare object detection

Yolo-based: **Yolo-v3, Yolo-v5, Yolo-v7, YoloR, YoloX, ...**

Normalmente più veloci ma un po' meno accurati

EfficientDet, ConvNext,

Sempre basati su convoluzioni, generalmente più accurati in funzione dell'anno di uscita

Transformer-based: **ViT, DETR, DINO**

Basati su Transformer che vedremo nella lezione legata al text processing

Recap

Rimangono da vedere

- Object segmentation
- Visual Embeddings
- Gradcam
- Generation
- Training pipeline



12 – Convolutional Neural Networks

Daniele Gamba

2022/2023

Lezione precedente

Abbiamo visto cos'è una CNN

- Convoluzioni
- Padding
- Pooling
- Object classification
- Object detection
- Tuning di una Yolo per identificare i cartelli stradali

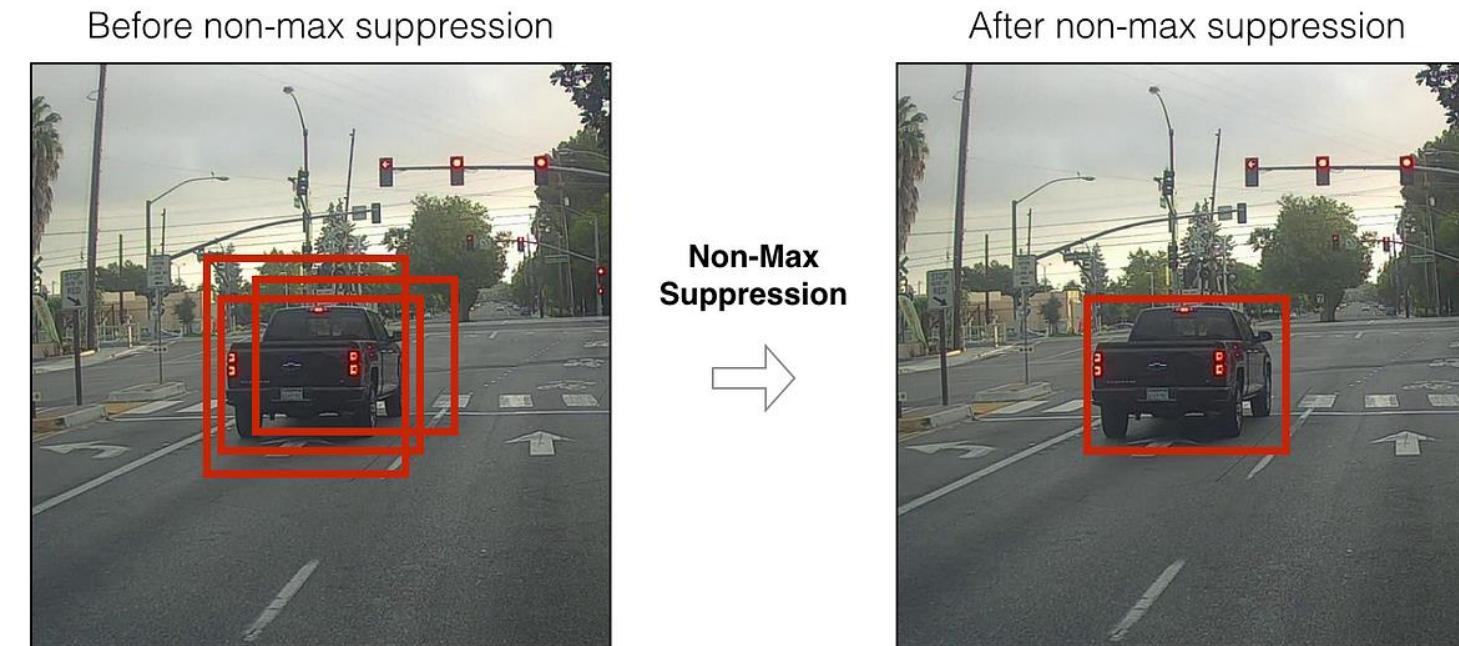
Recap

Rimangono da vedere

- NMS
- Object segmentation (unet + yolo head)
- Visual Embeddings
- Gradcam
- Generation (diffusion + vae)
- Training pipeline

Non-maxima suppression

I modelli basati su anchor (es. Yolo) eseguono N predizioni dell'oggetto di interesse perché ciascuna anchor crede di possedere il centro dell'oggetto. Di conseguenza ci troviamo a dover discriminare tra tante predizioni su qual è quella che meglio rappresenta l'oggetto.



Non-maxima suppression

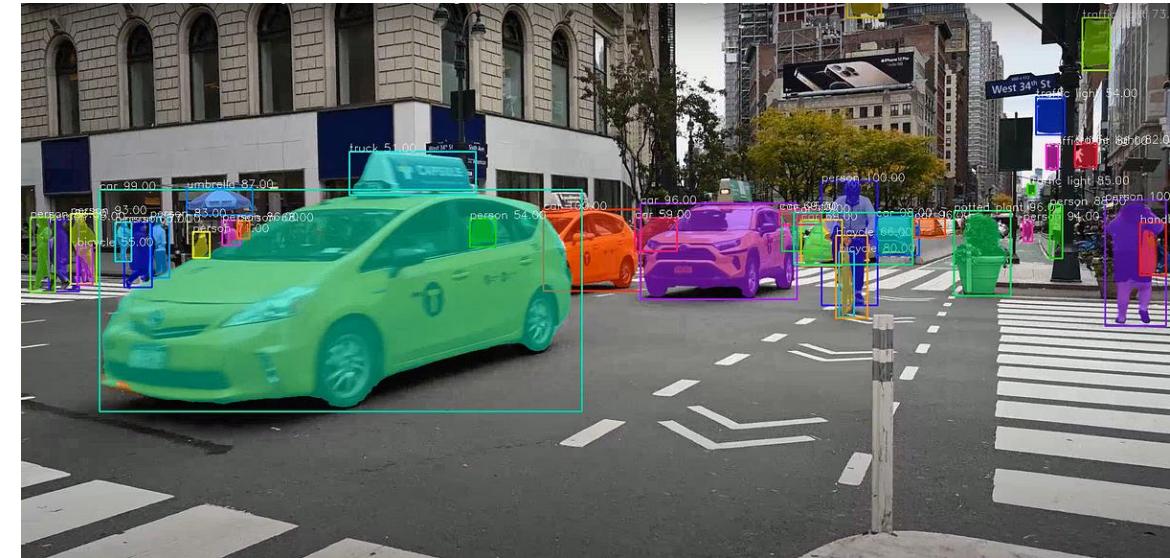
Per filtrare le bounding box

- Prendiamo la bb con la confidenza più alta e la teniamo come buona
- Prendiamo tutte le altre predizioni della stessa classe, se l'IoU supera una certa soglia le rimuoviamo dal set delle proposte
- Ripartiamo dal primo punto per ogni classe e ogni bb con IoU inferiore alla soglia

Object segmentation

Nel caso in cui volessimo addestrare un algoritmo non solo a riquadrare in una bouding box il nostro oggetto ma proprio ad indicarci i singoli pixel.

Il task di semantic segmentation tratta invece di identificare solamente la classe di ciascun pixel senza identificare che siano istanze di oggetti diversi

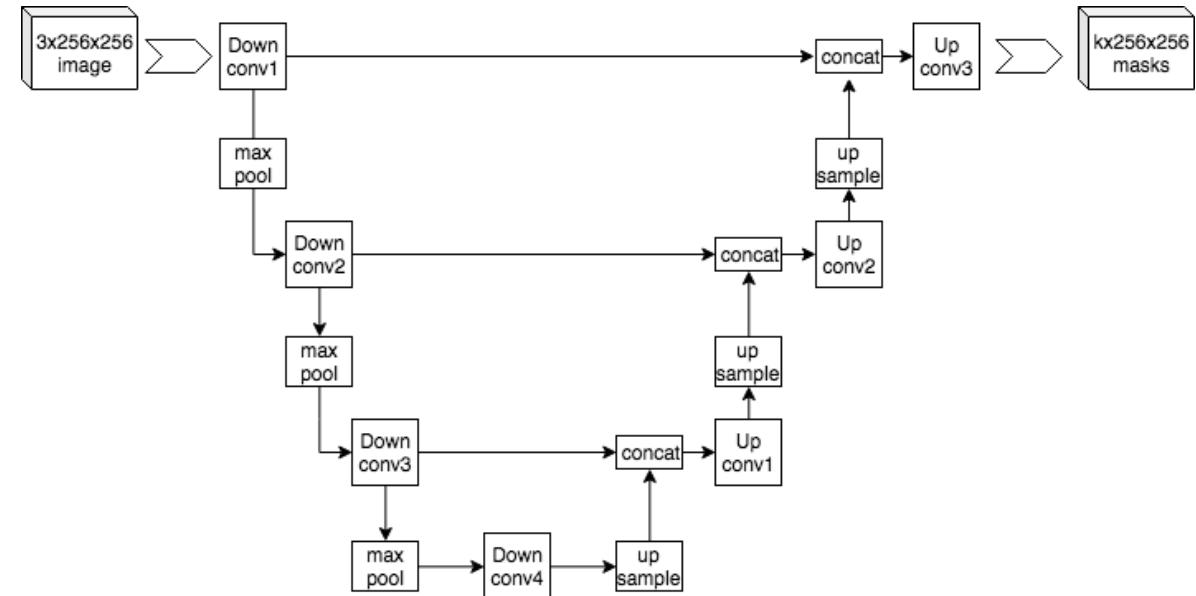


U-Net

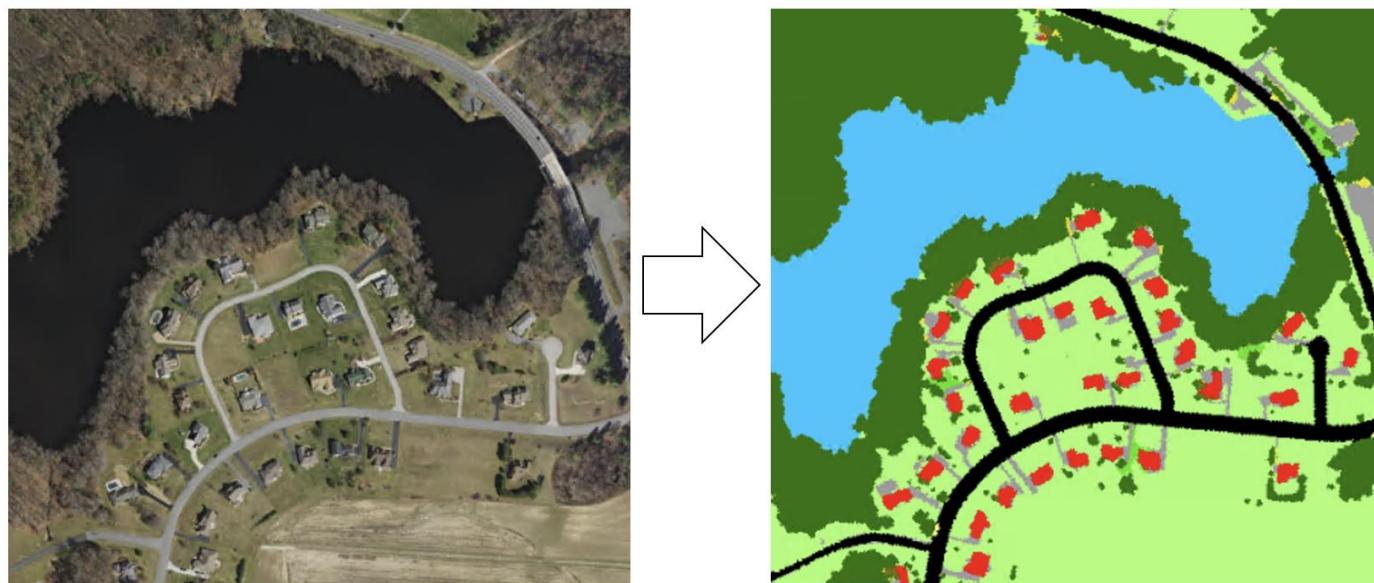
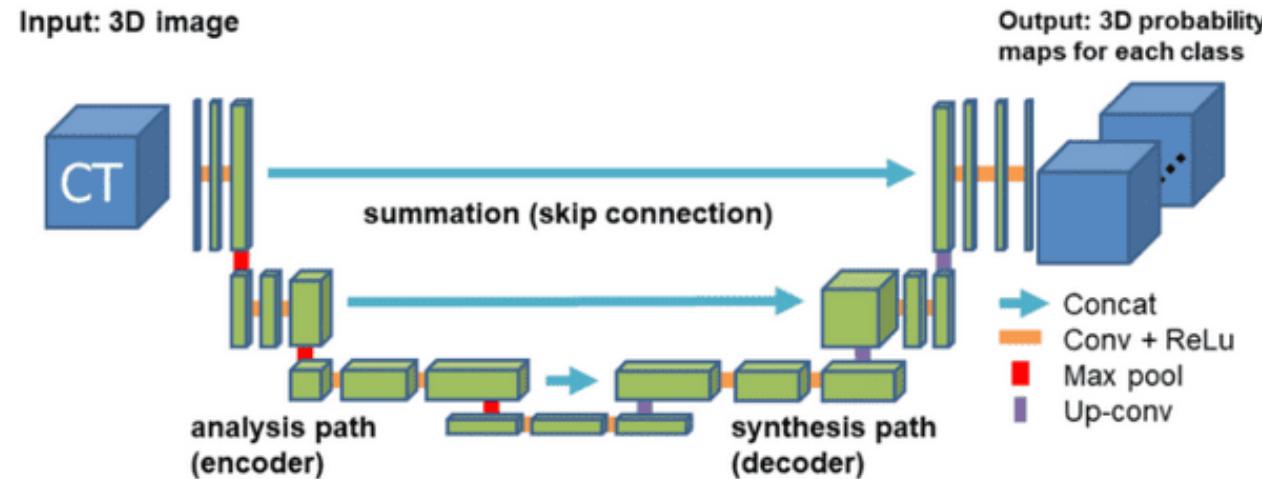
Una delle tecniche principali per far segmentazione è la U-Net.

Per com'è costruita cerca di creare una U in cui abbiamo convoluzioni che estraggono via via features sempre più deep, e connessioni dirette che trasportano il dato a diversi livelli di dettaglio.

Le features deep interpretano i gruppi di pixel, i layer cercano poi di rimappare sull'immagine più grande l'informazione.



U-Net



U-Net

Per addestrare le U-Net abbiamo bisogno di label molto più espressive che possono essere

- Poligoni che racchiudono l'oggetto di interesse
- Vere e proprie immagini / mappe che vogliamo ottenere in uscita dalla rete

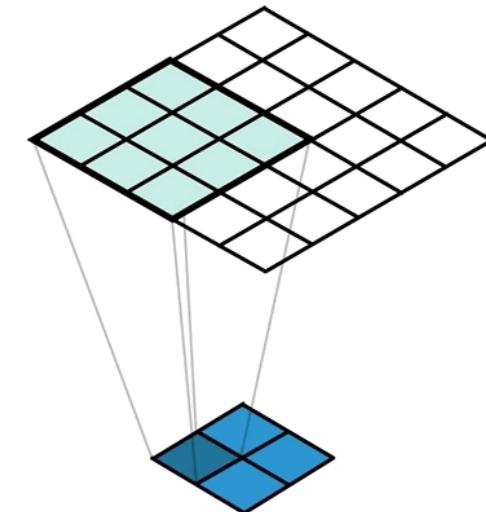
La funzione di costo deve cercar di mantenere un buon livello di dettaglio, per cui si tende ad usare funzioni di costo che non scalano quadraticamente gli errore più lontani ma funzioni come MAE.

Ci serve anche sapere come passare da un layer a bassa risoluzione ad uno più grande.

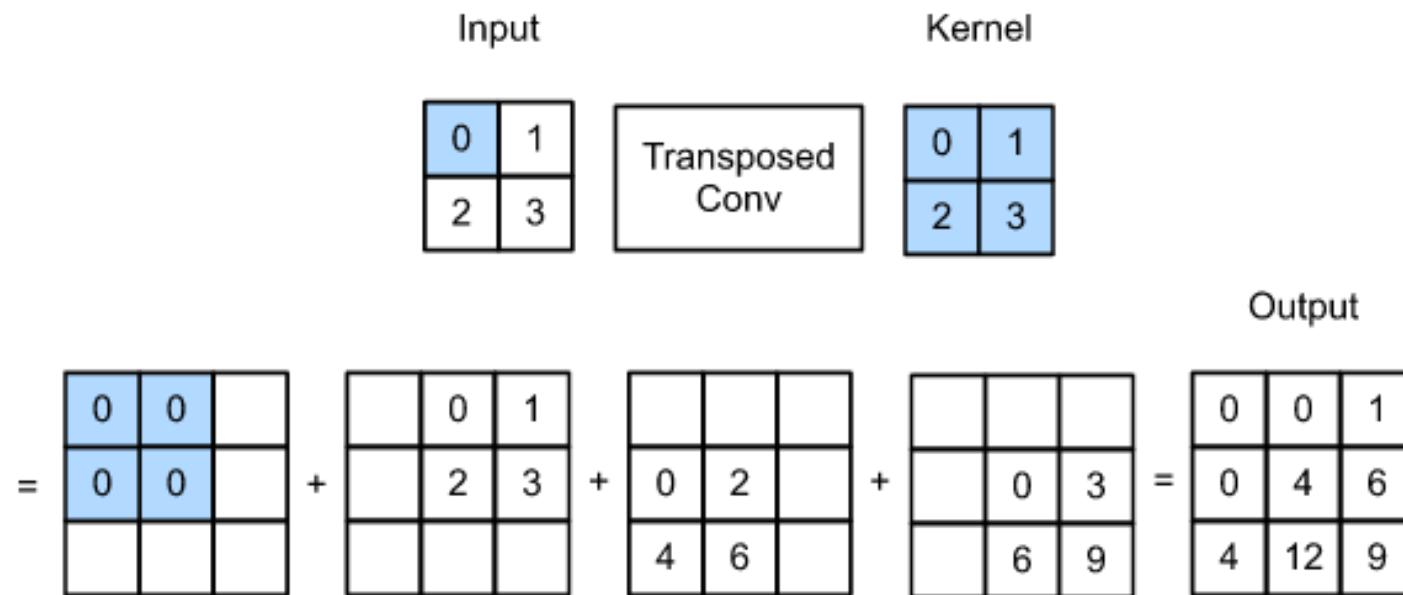
Conv2DTranspose

Nella U-Net dobbiamo salire di dimensione una volta raggiunto il livello più basso della nostra CNN. Per farlo possiamo usare la Conv2DTranspose per fare upsampling.

Il nostro layer imparerà un kernel (come nella convoluzione) che userà poi per fare upsampling del layer precedente.



Conv2DTranspose



Yolo & Mask R-CNN

Sia la R-CNN che la Yolo possono esser riadattate a fare object segmentation.

Entrambe lo fanno predicendo o una lista di punti che contengono il poligono dell'oggetto o andando a predire delle maschere che vengono poi posizionate all'interno di ciascuna Bounding Box.



SAM

Segment Anything Model (SAM) è l'ultimo modello rilasciato nel mondo della object segmentation da Meta.

È un modello enorme in confronto a modelli come Yolo o le UNet ed infatti è stato in larga parte addestrato in self-supervising learning su un dataset infinito (11 milioni di immagini e più di 1 miliardo di maschere).



Generate multiple valid masks for ambiguous prompts

Recap

Abbiamo visto

- non-maxima suppression
- Object segmentation con
 - Unet
 - Modelli standard (Yolo, MaskRCNN)
 - SAM

13 – RNN

Daniele Gamba

2022/2023

Sequenze di input

Fin'ora abbiamo visto reti che hanno input di dimensioni note, per esempio il nostro vettore di features o la dimensione dell'immagine.

Escludendo le Fully-Convolutional NN ci consentono di scalare su dimensioni più grandi rispetto a quelle su cui sono state addestrate, come possiamo trattare dati con lunghezze variabili?

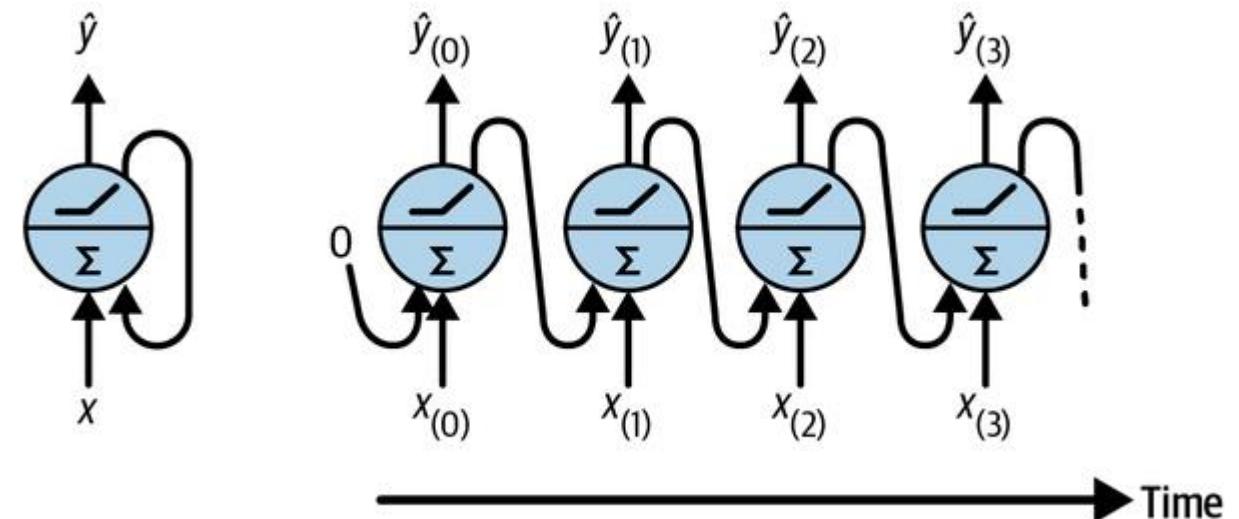
Come gestiamo sequenze in input per prevedere il futuro di una serie storica?

Recurrent Neural Network

Le reti neurali ricorrenti, o **RNN**, sono strutture che ci consentono di elaborare sequenze arbitrariamente lunghe.

Si basano sul neurone ricorrente, una struttura in grado di ricevere un ingresso e passare un valore come secondo ingresso a se stesso.

In questo modo, a prescindere dalla lunghezza della serie, possono proseguire ad elaborare input.

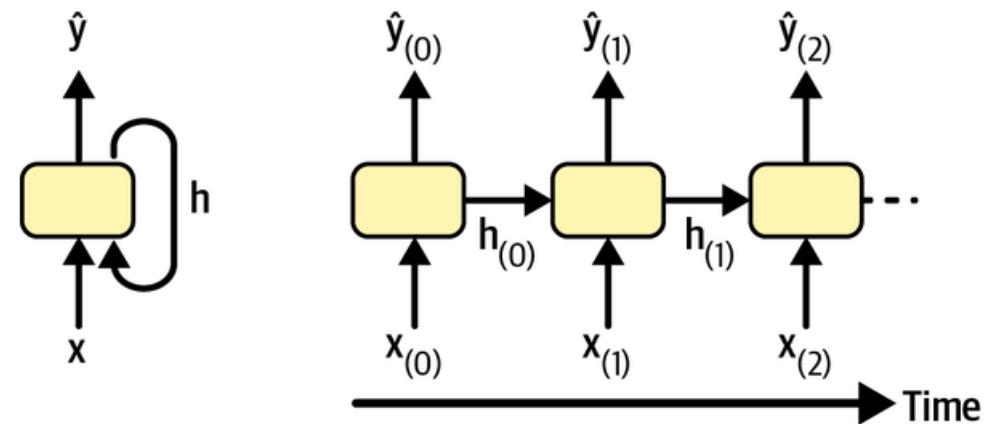
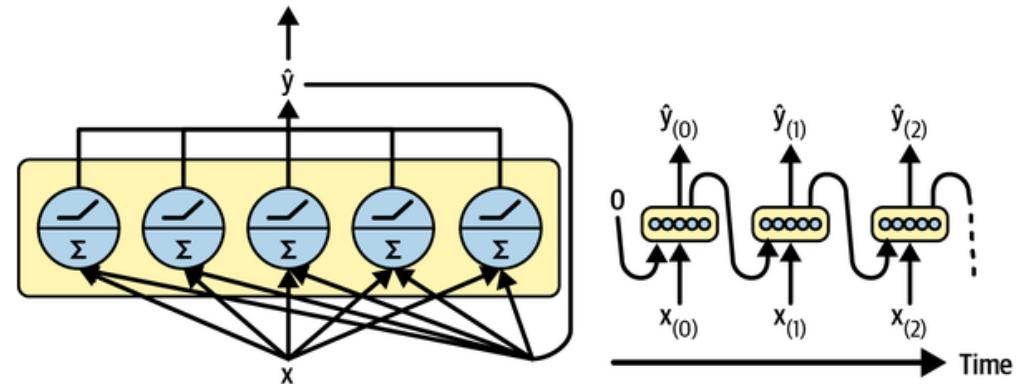


Recurrent Neuron

Ogni neurone avrà due set di pesi, quelli relativi all'input esterno x_t e quelli relativi alla sua y_{t-1}

I neuroni in questo modo accumulano una memoria su quello che è stato osservato.

Non per forza il valore ripassato equivale all'uscita, possiamo anche scegliere di passare uno stato nascosto h man mano che svolgiamo (unroll) il calcolo.



Strutture di RNN

La RNN è molto flessibile e possiamo realizzare strutture molto diverse

- Sequence to Sequence, dove sia ingresso che uscita sono calcolate insieme
- Sequence to Vector, se vogliamo rappresentare in forma densa una sequenza arbitraria
- Vector to Sequence, per svolgere il contenuto di un vettore
- Encoder Decoder, dove la rete di encoder codifica al suo interno uno stato che passa poi al decoder

La prima e l'ultima vengono usate molto per, ad esempio, fare traduzioni

Strutture di RNN

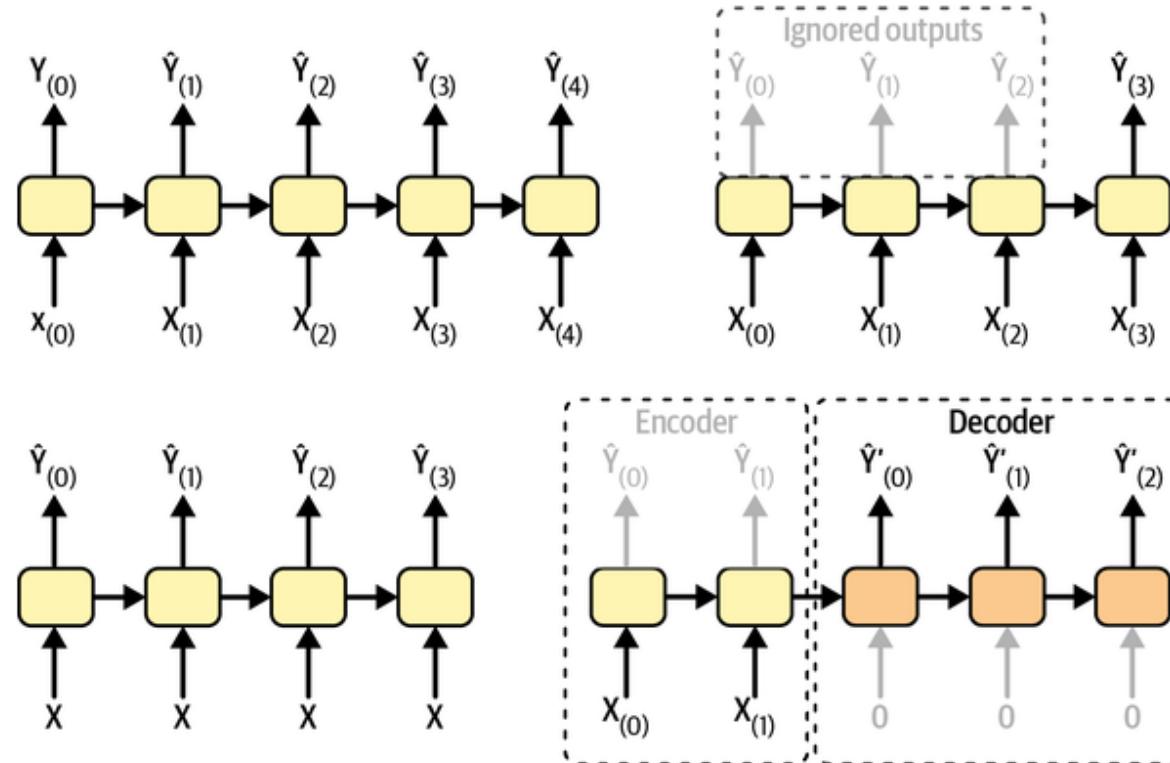


Figure 15-4. Sequence-to-sequence (top left), sequence-to-vector (top right), vector-to-sequence (bottom left), and encoder-decoder (bottom right) networks

AR, ARMA, ARIMA, ecc.

Nel fare modellizzazione di serie storiche non dimentichiamo l'importanza dei modelli tradizionali.

RNN

Una RNN non è altro che una rete neurale
che fa uso di neuroni ricorrenti.

Possiamo costruire strutture più o meno
deep, integrando anche parti dense.

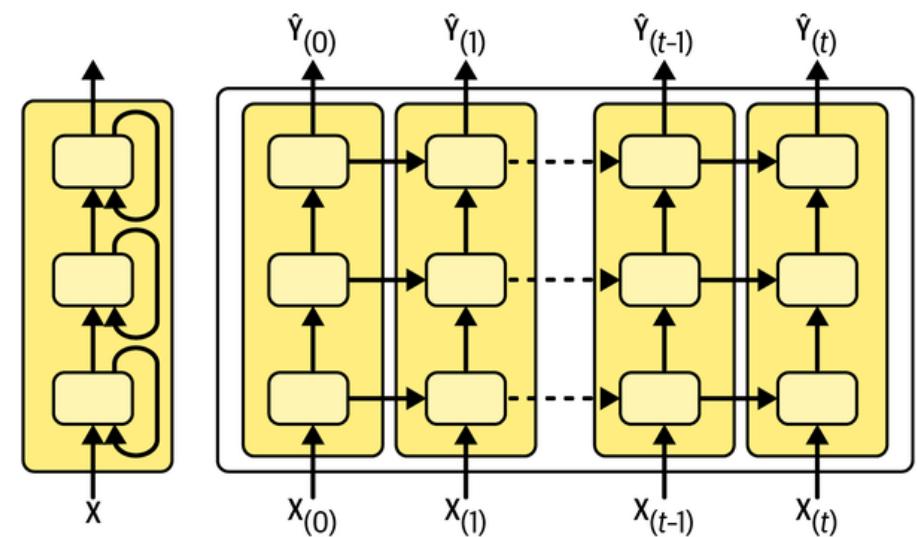


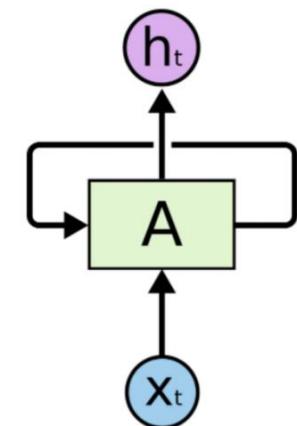
Figure 15-10. A deep RNN (left) unrolled through time (right)

SimpleRNN

Il primo Layer ricorrente che vediamo si chiama SimpleRNN.

Per poter essere usato necessita di alcuni parametri in più

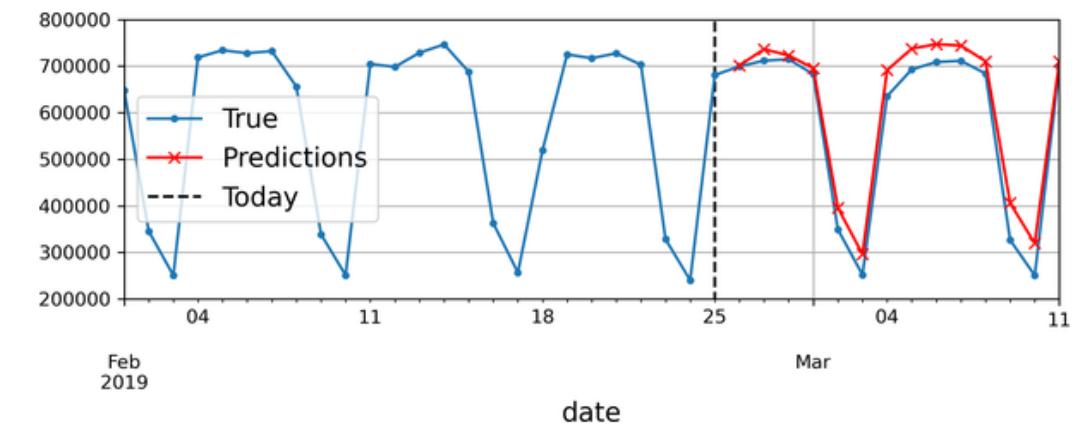
- **units**: ovvero il numero di neuroni ricorrenti
- **return_sequence**: T/F se vogliamo che venga ritornata tutta la sequenza o solo l'ultimo output
- **return_state**: se vogliamo che venga restituito anche lo stato interno oltre all'output
- **stateful**: se vogliamo che venga mantenuto lo stato venga passato al batch successivo
- **unroll**: per svolgere tutto il neurone in memoria invece che lasciarlo ricorrente, solo per sequenze definite e relativamente brevi
- **input_shape**
 - **batch_size**
 - **time_step**: possiamo indicare la lunghezza massima o None per sequenze arbitrariamente lunghe
 - **dimensionality**: il numero di features in ingresso



Forecasting

Per addestrare una rete ad eseguire del forecasting dovremo darle in ingresso la sequenza e in uscita l'istante di tempo $t+1$.

Una volta addestrata per prevedere istanti di tempo successivi accoderemo il nostro forecast del tempo $t+1$ alla sequenza in ingresso per prevedere il tempo $t+2$, ecc.



Problemi delle RNN

Le RNN tendono ad avere diversi problemi

In primis sono più lente sia in apprendimento che in inferenza dovendo svolgere ogni volta tutta la sequenza; lo step successivo deve aspettare lo stato dello step precedente.

Tendono ad aver problemi a portare avanti la memoria, un stato nascosto viene moltiplicato ogni volta internamente al neurone e difficilmente riesce ad esser trasportato molto a lungo nella sequenza. Non si può usare efficacemente il batch normalization soprattutto se la sequenza prosegue su più batch.

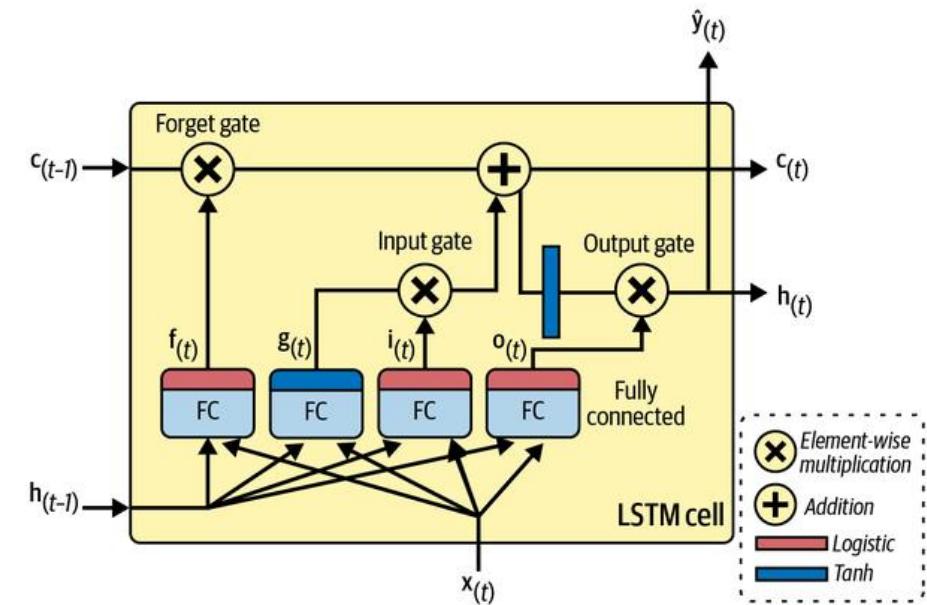
Per questo son stati inventati altri tipi di neuroni ricorrenti

Long-short term memory

Le celle LSTM sono state inventate per riuscire a portare avanti in modo efficace la memoria anche in sequenze lunghe.

Queste celle hanno due stati che vengono portati avanti, uno «long-term» c e uno «short-term» h .

La cella decide quando mantenere la memoria a lungo termine, quando cancellare la memoria, e quando leggerla dall'input attraverso una serie di *gate*.



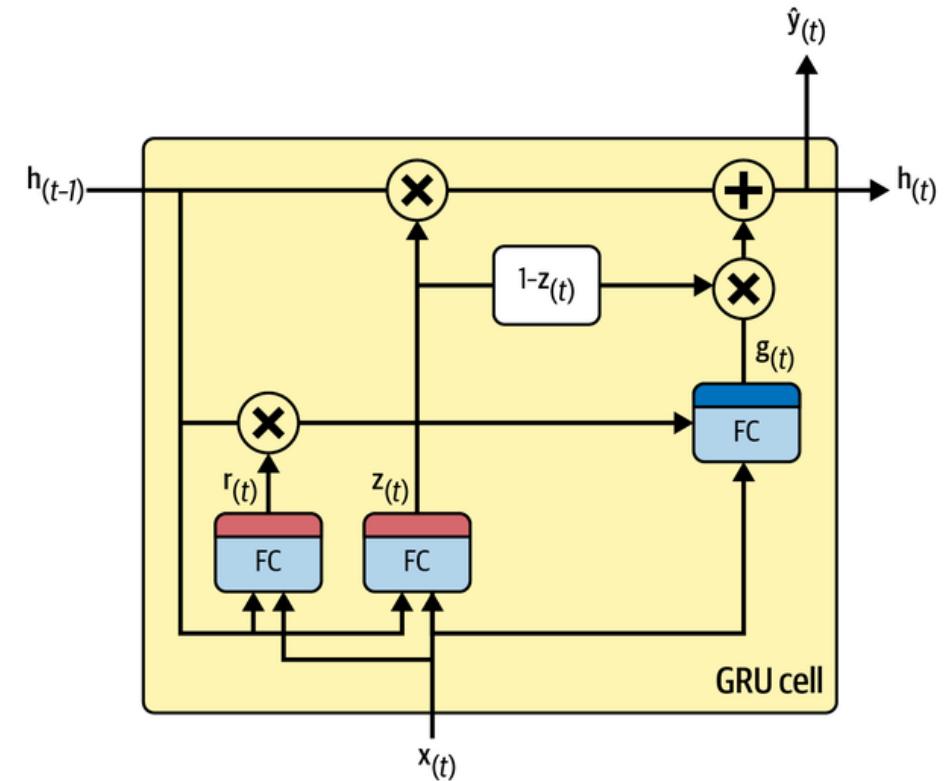
Gated-recurrent Unit

Il Gated Recurrent Unit, GRU, è un neurone che semplifica la LSTM.

Non vi è distinzione tra stato e output.

Lo stato è unico (non c'è distinzione tra long-term e short-term) e controllato da un unico gate.

In molte applicazioni nonostante fosse una semplificazione si è dimostrato performante tanto quanto la LSTM e per questo è spesso più usato.

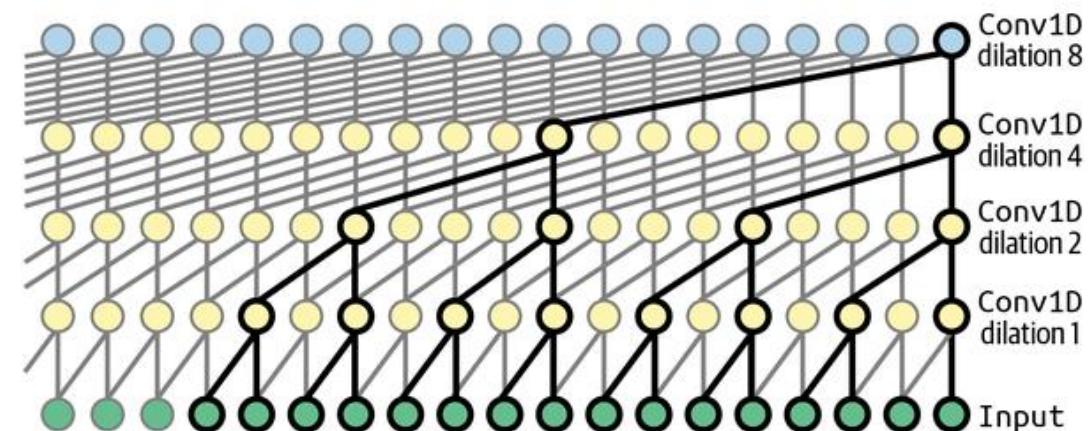


1D Convolution - WaveNet

Come abbiamo visto nella lezione precedente, una alternativa alle RNN per il processing di serie storiche di lunghezza ignota è quella di utilizzare Convoluzioni 1D.

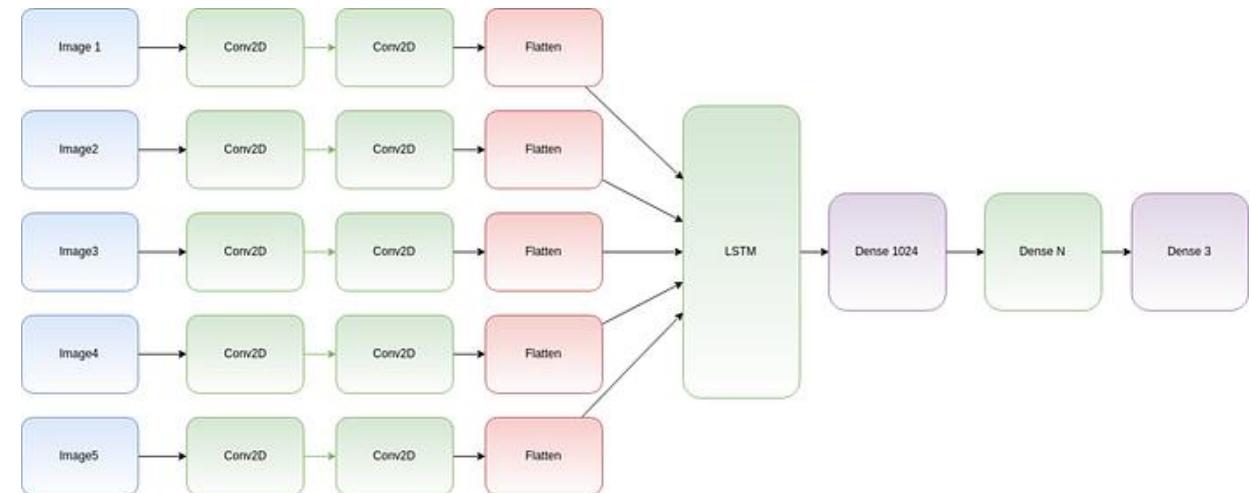
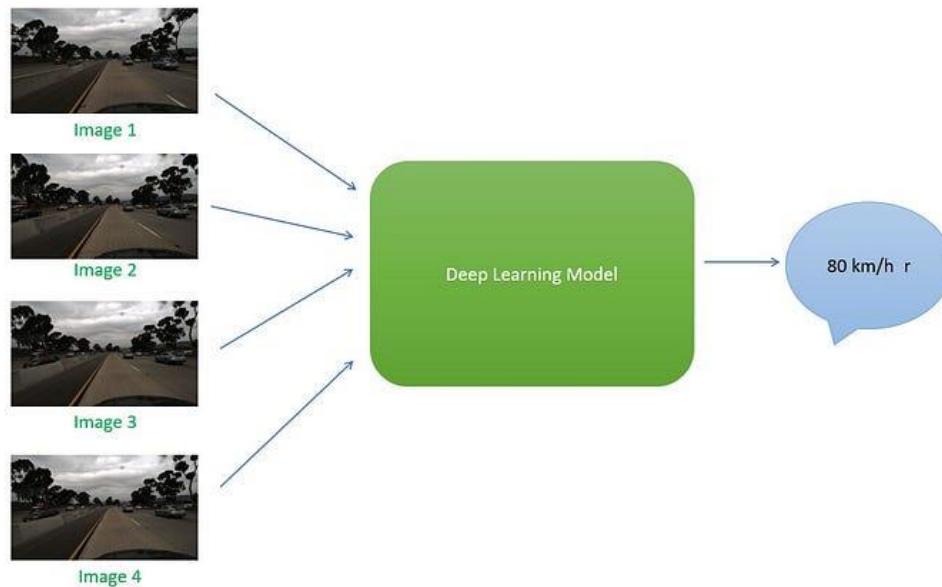
In questo senso la rete più famosa è la WaveNet, in cui sono concatenate una serie di convoluzioni 1D in cui ciascun layer raddoppia il «dilation-rate», ovvero la distanza tra gli input considerati.

In questo modo WaveNet intercetta nei layer più bassi pattern locali «short-term» mentre nei layer più in alto interpreta sequenze più lunghe pur non avendo memoria.



TimeDistributed

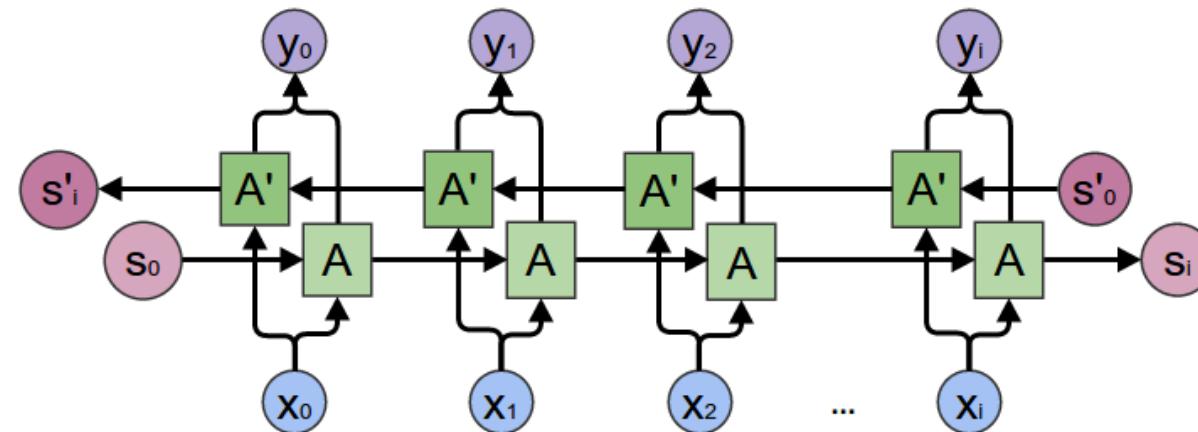
Nel caso in cui volessimo applicare dei pezzi di modello in modo indipendente rispetto ad altri, possiamo usare il decoratore TimeDistributed. In questo modo Tensorflow in automatico applicherà quel pezzo di modello lungo l'asse tempo mantenendone i pesi invariati per poi passare l'output ad una RNN o Densa.



Bi-directional RNN

Nel caso i nostri input siano in sequenza e non dobbiamo analizzarli man mano arrivano ma abbiamo a disposizione tutta la sequenza possiamo usare delle RNN bi-direzionali.

In questo modo la sequenza viene scansionata in entrambe le direzioni ed il modello è conoscenza non solo della storia precedente ma anche di quella futura. Questa tipologia di reti è molto utile nel caso, ad esempio, dell'analisi testuale.



Recap

Abbiamo visto che le RNN sono strutture ricorrenti che ci consentono di aver memoria.

Dovendo passare ad ogni neurone lo stato del neurone precedente sono più difficili da parallelizzare e pertanto più lente.

Sono ampiamente usate per fare forecasting ed elaborazione del testo.



14 – NLP

Daniele Gamba

2022/2023

Natural Language Processing

Il Natural Language Processing riguarda le applicazioni di elaborazione testo.

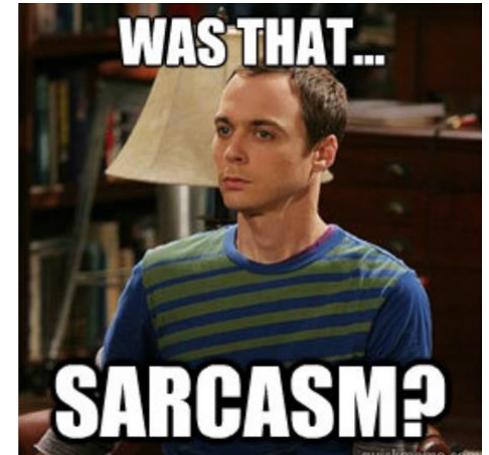
Dal testo possiamo voler estrarre diverse informazioni come

- Identificare le parti chiave di una frase
- Ricercare all'interno di un documento
- Confrontare frasi diverse
- Generare domande e risposte

Difficoltà

Il testo è storicamente difficile da trattare perché

- Abbiamo molte lettere nell'alfabeto
- Combinazioni di lettere simili non per forza hanno lo stesso significato (casa, caso, ..)
- La stessa parola assume significati diversi dal contesto
- Il senso della frase non è universalmente definito
- Lo stesso concetto si può esprimere in una moltitudine di forme scritte diverse
- Esistono diverse lingue e modi di dire
- In internet le persone tendono a scrivere decisamente male (abbreviazioni, frasi senza senso, CoSECOs!!!11!!)
- Il sarcasmo



Rappresentazione

Innanzitutto ci serve definire una rappresentazione numerica per il testo.

Quella storicamente più usata è la «one-hot encoding», ovvero per ogni parola andiamo a costruire un vettore di tanti zeri quanto è grosso il nostro vocabolario e con 1 nella posizione associata alla parola.

Chiaramente questo approccio ha dei problemi

- Ci serve un dizionario finito di parole, potenzialmente molto ampio
- La rappresentazione è estremamente sparsa
- Dobbiamo trovare un modo di rappresentare parole simili o con lo stesso significato

Pre-processing

Una prima operazione da fare, nel caso in cui volessimo rappresentare il significato di una frase, è andare a pulire il nostro input da tutta una serie di caratteri che possono darci fastidio.

Il preprocessing sarà votato a

- Mettere tutto in lowercase (CaSa -> casa)
- Togliere gli articoli (the, and, of, ..)
- Togliere plurali (nel caso inglese _s)
- Togliere virgolette, spazi, ecc.

Stemming

Un modo per avvicinarci ad una rappresentazione più densa del nostro vocabolario è quella di tenere solamente le radici delle parole attraverso lo **stemming**.

Si vanno a rimuovere i suffissi delle parole *announces*, *announced*, *announcing* per ottenere *announc*.

Si possono usare delle regole fisse o sfruttare un database più ampio di regole.

È chiaro che in lingue come l'italiano è più complicato andare a fare stemming per preservare un significato di massima (casa, caso, banca, banco, ...)

Term Frequency

Una volta fatto stemming possiamo passare a rappresentare la nostra frase vettorialmente andando a sommare le codifiche one hot di tutte le parole.

Si ottiene in questo modo quello che viene chiamata **Term Frequency** o **TF**.

d1 – jazz music has a swing rythm

d2 – swing is hard to explain

d3 – swing rythm is a natural rythm

	a	explain	hard	has	is	jazz	music	natural	rythm	swing	to
d1	1	0	0	1	0	1	1	0	1	1	0
d2	0	1	1	0	1	0	0	0	0	1	1
d3	1	0	0	0	1	0	0	1	2	1	0

Inverse Document Frequency

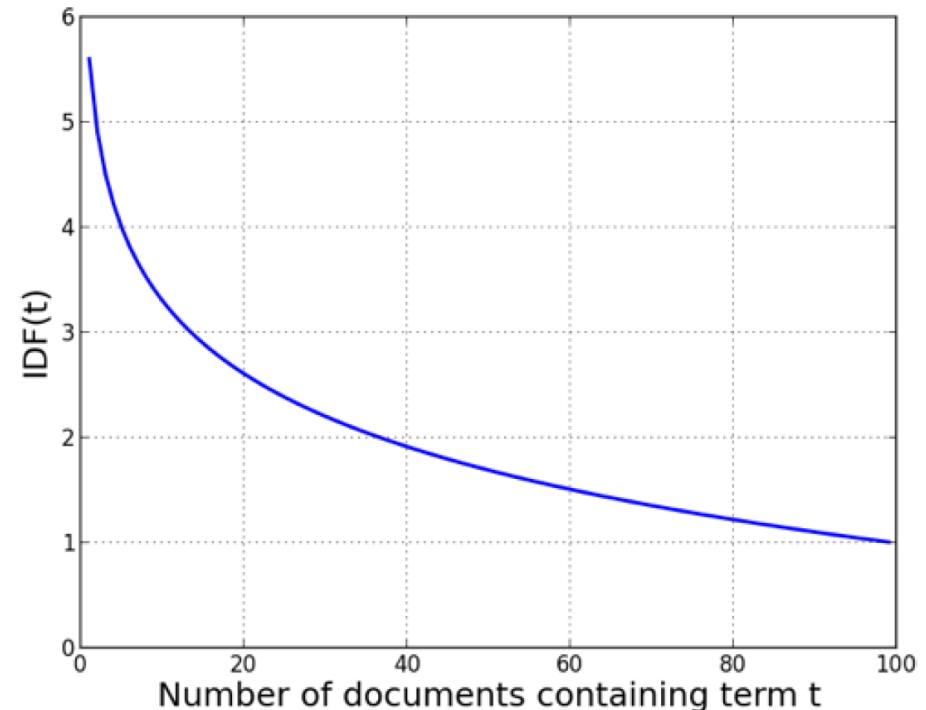
TF ha lo svantaggio che tende a favorire documenti, ad esempio nella ricerca, che semplicemente ripetono molte volte un termine senza aggiungere alcun significato. Per questo è complementare considerare i documenti in cui un termine appare meno frequentemente.

Si calcola in questo caso l'**Inverse Document Frequency** o **IDF** e di fatto calcola il livello di sparsificazione di un termine su N documenti.

$$IDF(t) = 1 + \log\left(\frac{\text{total number of docs}}{\text{Number of docs containing } t}\right)$$

Inverse Document Frequency

Le stopwords o altri termini particolarmente frequenti avranno quindi una IDF quasi pari ad 1 perché condivisa in tutti i documenti, mentre termini molto rari (e quindi significativi rispetto alla definizione di informazione) avranno valori molto più alti.



TF-IDF

La combinazione di frequenza e sparsificazione ci consente di ottenere la rappresentazione TF-IDF.

$$TFIDF(t, d) = TF(t, d) \cdot IDF(t)$$

In questo modo, ogni documento viene pesato per la frequenza delle sue parole e la significatività che queste hanno sul totale.

Mettiamo di cercare un documento specifico che tratta un argomento, parole specifiche che vengono trattate solo in quel contesto assumeranno un grande peso nel documento.

Esempio – Jazz Musicians

Consider the biography taken from Wikipedia, of 15 Jazz musicians:

Charlie Parker

Charles “Charlie” Parker, Jr., was an American jazz saxophonist and composer. Miles Davis once said, “You can tell the history of jazz in four words: Louis Armstrong. Charlie Parker.” Parker acquired the nickname “Yardbird” early in his career and the shortened form, “Bird,” which continued to be used for the rest of his life, inspired the titles of a number of Parker compositions, [...]

Duke Ellington

Edward Kennedy “Duke” Ellington was an American composer, pianist, and bigband leader. Ellington wrote over 1,000 compositions. In the opinion of Bob Blumenthal of The Boston Globe, “in the century since his birth, there has been no greater composer, American or otherwise, than Edward Kennedy Ellington.” A major figure in the history of jazz, Ellington’s music stretched into various other genres, including blues, gospel, film scores, popular, and classical.[...]

Miles Davis

Miles Dewey Davis III was an American jazz musician, trumpeter, bandleader, and composer. Widely considered one of the most influential musicians of the 20th century, Miles Davis was, with his musical groups, at the forefront of several major developments in jazz music, including bebop, cool jazz, hard bop, modal jazz, and jazz fusion.[...]

Even with this small corpus of fifteen documents, the corpus and its vocabulary are about 2,000 features after stemming and stopword removal

Esempio – Jazz Musicians

Input phrase

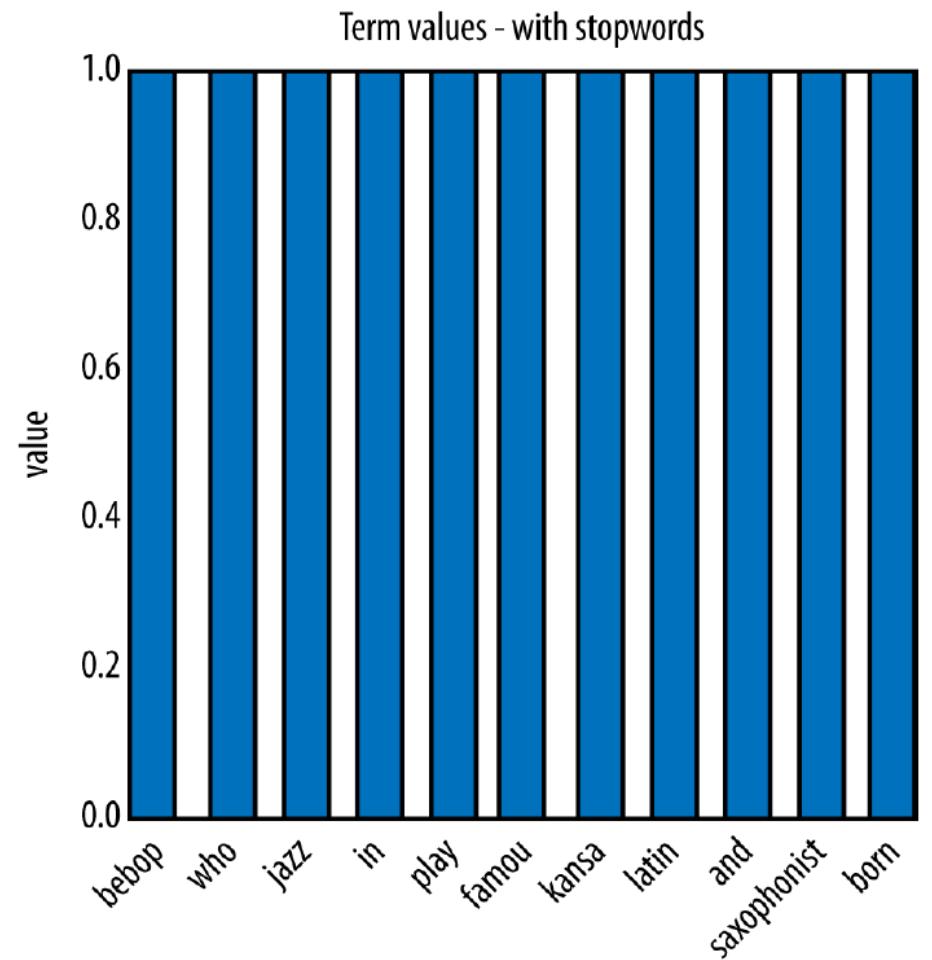
“Famous jazz saxophonist born in Kansas who played bebop and latin”

Consider it as an input in a search engine.

How to represent it

Treat it as a document.

1. **Stemming:** *kansa* as Kansas, *famou* as famous. It is not important if it is not precise, the importance is the repeatability



Esempio – Jazz Musicians

Input phrase

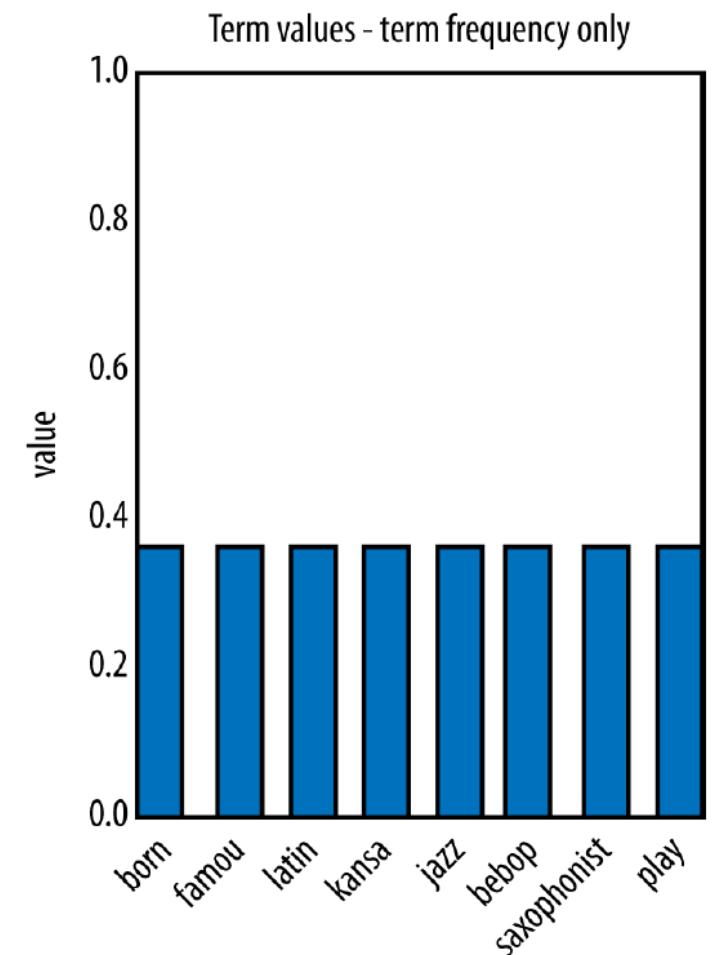
“Famous jazz saxophonist born in Kansas who played bebop and latin”

Consider it as an input in a search engine.

How to represent it - TF

Treat it as a document.

1. **Stemming:** *kansa* as Kansas, *famou* as famous. It is not important if it is not precise, the importance is the repeatability
2. **Stop word removal:** the terms *in* and *and* are removed and the value normalized



Esempio – Jazz Musicians

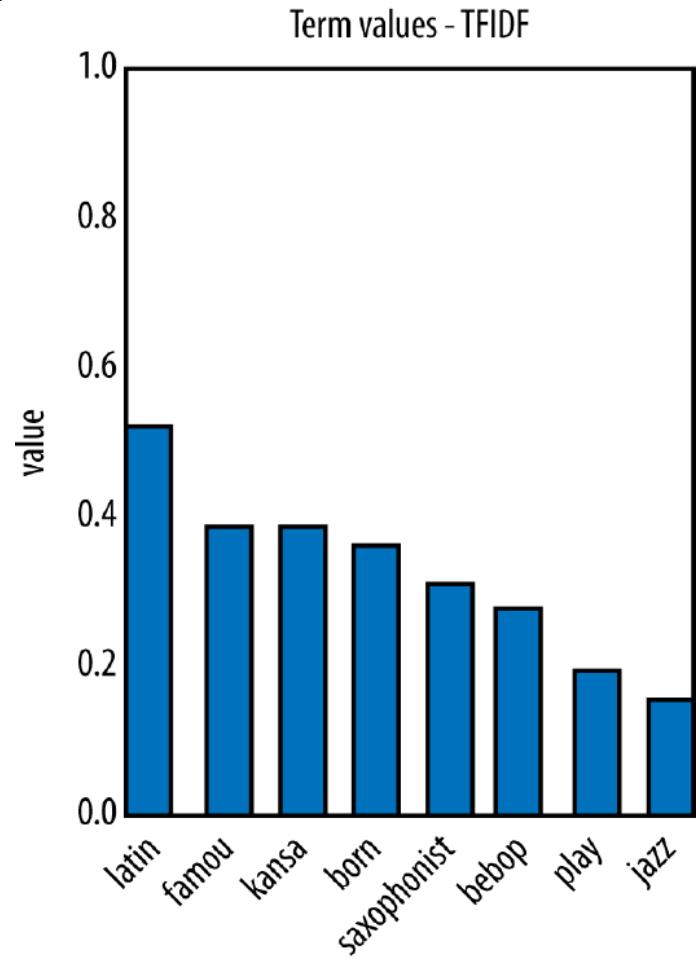
TFIDF

The next step, requires to compute TFIDF for all the 15 documents we collected, doing for each the preprocessing shown before.

Input phrase - TFIDF

Once we have the IDF for all the documents, we can compute the TFIDF of our input phrase.

Thus our input phrase is now a vector



Esempio – Jazz Musicians

Similarity

We need to find the jazz musician which has the TFIDF closest to the input phrase.

This mean that we need to compute a distance between vectors

Cosine similarity

A typical approach is to use the cosine similarity:

$$\text{sim}(A, B) = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Musician	Similarity	Musician	Similarity
Charlie Parker	0.135	Count Basie	0.119
Dizzie Gillespie	0.086	John Coltrane	0.079
Art Tatum	0.050	Miles Davis	0.050
Clark Terry	0.047	Sun Ra	0.030
Dave Brubeck	0.027	Nina Simone	0.026
Thelonius Monk	0.025	Fats Waller	0.020
Charles Mingus	0.019	Duke Ellington	0.017
Benny Goodman	0.016	Louis Armstrong	0.012

Altre rappresentazioni

TF-IDF si avvicina ad una rappresentazione più sensata ma comunque è molto limitata nella sua rappresentatività. Per questo sono state studiate diverse altre rappresentazioni del testo.

Uno dei limiti è che TF-IDF considera ogni parola come indipendente, mentre sappiamo bene che le parole hanno probabilità diverse se viste in sequenza.

N-Gram è una rappresentazione che associa le parole in rappresentazioni a 2 a 2 nel caso N=2.

N-Gram

«*The quick brown fox jumps*»

BOW: { 'quick' - 'brown' - 'fox' - 'jumps' }

2-gram: { 'quick' - 'brown' - 'fox' - 'jumps' - 'quick_brown' - 'brown_fox' - 'fox_jumps' }

3-gram: { 'quick' - 'brown' - 'fox' - 'jumps' - 'quick_brown' - 'brown_fox' - 'quick_brown_fox' - 'brown_fox_jumps' } - 'fox_jumps' - - 'fox_jumps' - -

La tripletta *exceed_analyst_expectation* è molto più significativa di considerare separatamente le tre parole *analyst*, *expectation* e *exceed*.

Il grosso limite di questa rappresentazione però è che aumenta esponenzialmente la cardinalità del dataset.

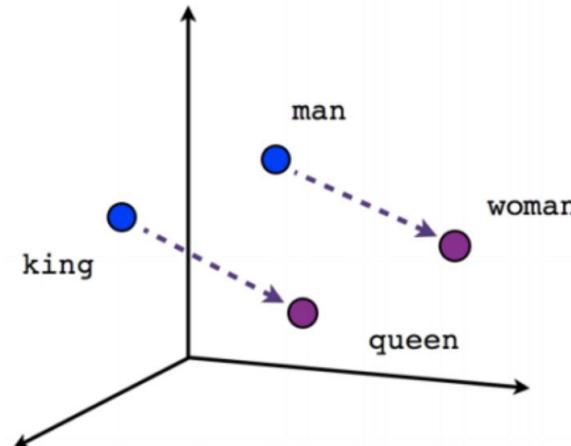
Word2Vec

Con l'avvento del deep-learning si è cercato di rappresentare le parole con rappresentazioni più dense, dove il vettore che caratterizza la parola è un vettore di cardinalità limitata (es. 300).

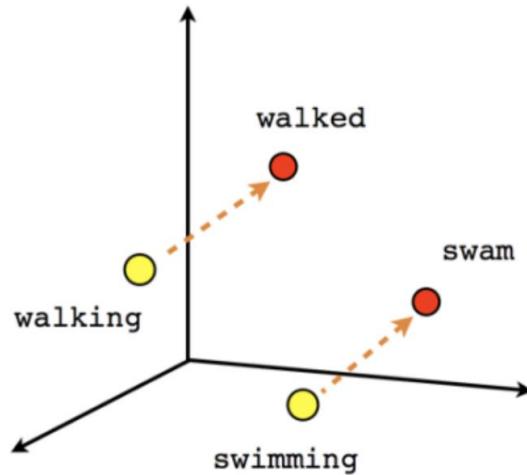
Questo vettore, chiamato embedding, contiene il significato della parola posto in uno spazio N dimensionale e in cui si cercano di far valere delle relazioni numeriche tra le parole.

Questa rappresentazione densa ci consente di trovare le parole più simili di significato, trovare gli opposti fino a svolgere vedere e proprie operazioni come sottrarre il «significato» di *uomo* da *re* e sommare *donna* per ottenere *regina*.

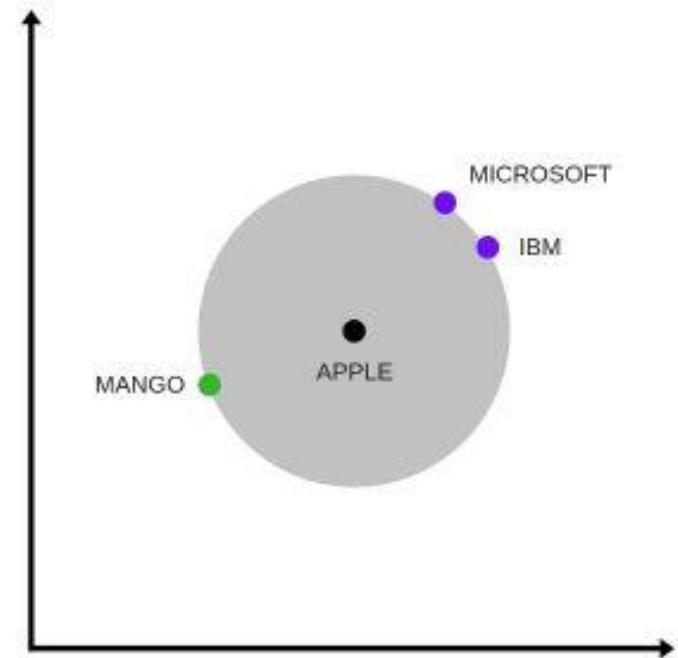
Word2Vec



Male-Female



Verb tense



Doc2Vec

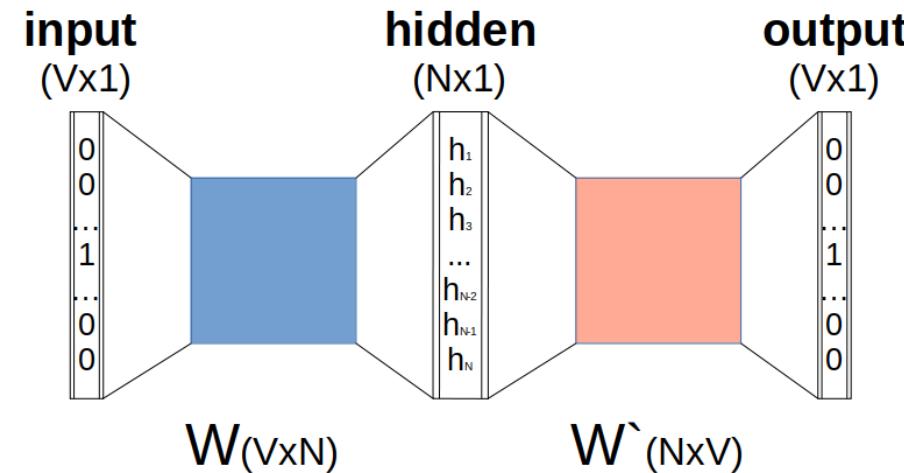
Espandendo la rappresentazione non più solo alla singola parola ma andando a codificare tutto il documento possiamo codificare il significato di intere parti di testo sempre in vettori densi.

Ma come possiamo addestrare questa rappresentazione Densa?

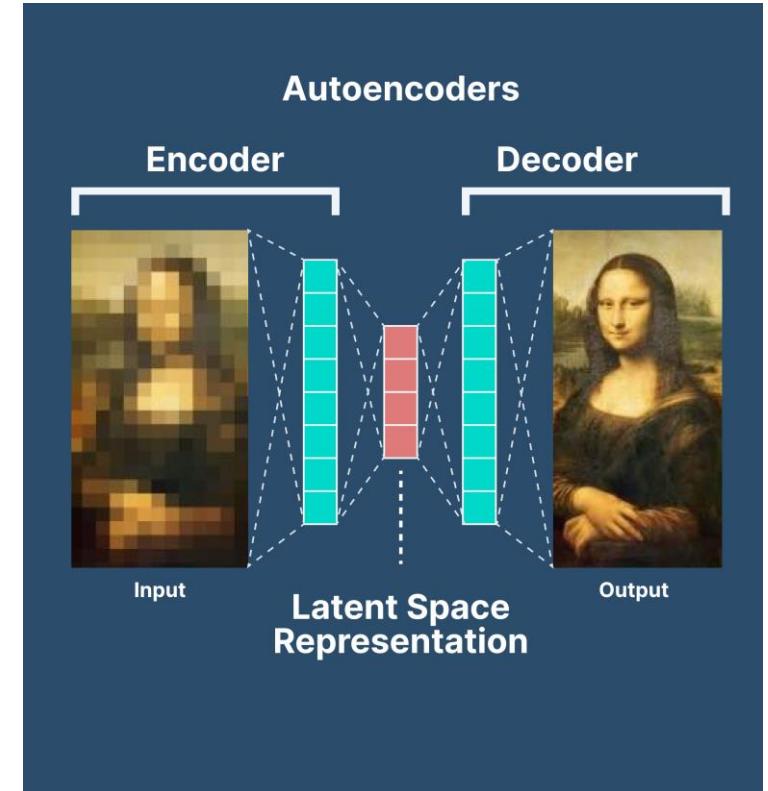
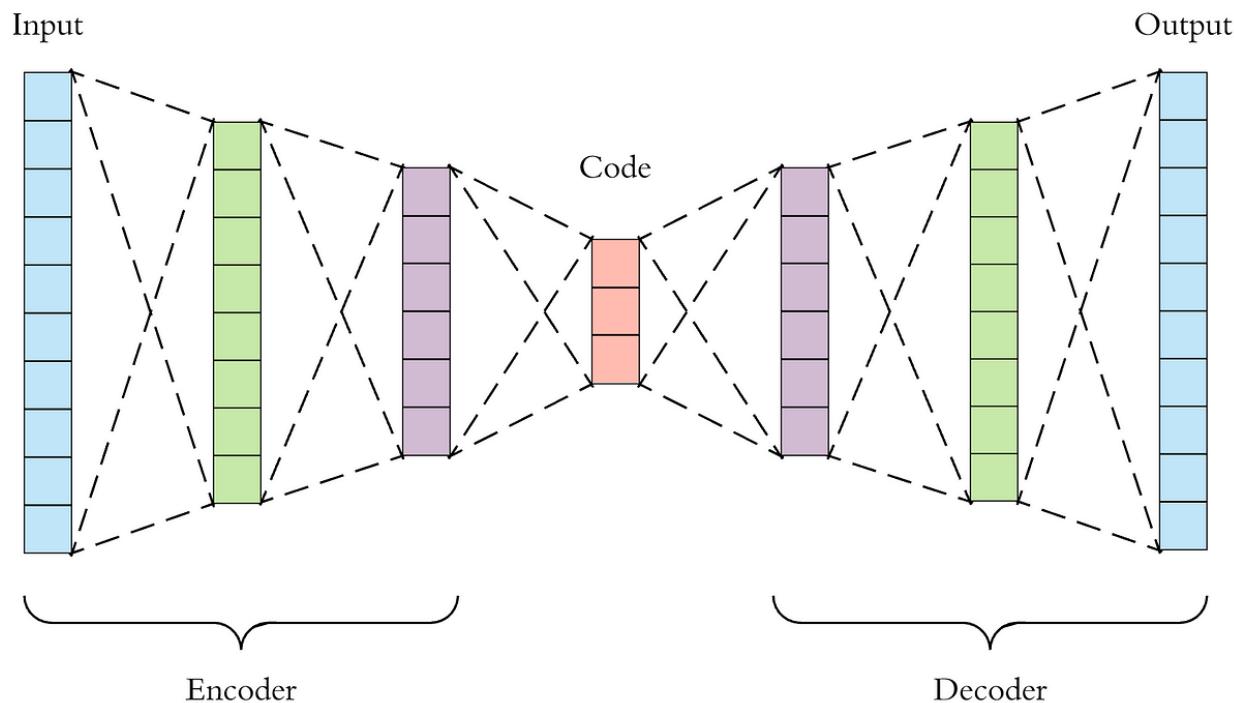
Autoencoder

In generale si addestrano rappresentazioni dense di input sparsi tramite autoencoder.

L'autoencoder è una struttura di rete neurale che prende N input, li comprime in una rappresentazione intermedia dimensionalmente molto minore che poi de-comprime per ricostruire il vettore originale. Di dimensione N . La loss è calcolata a partire dalla differenza tra ingresso ed uscita.



Autoencoder

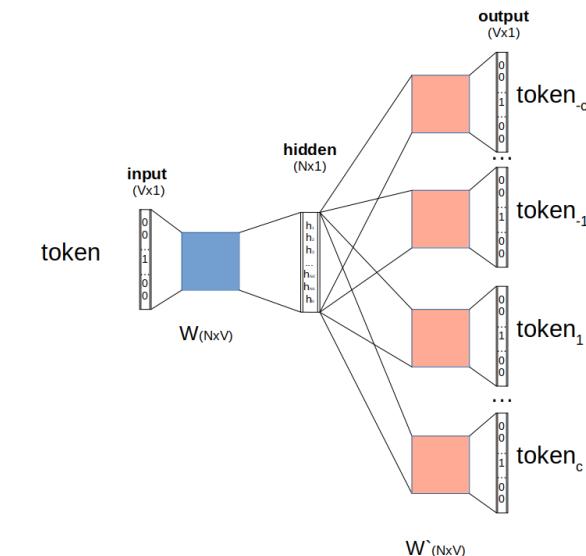
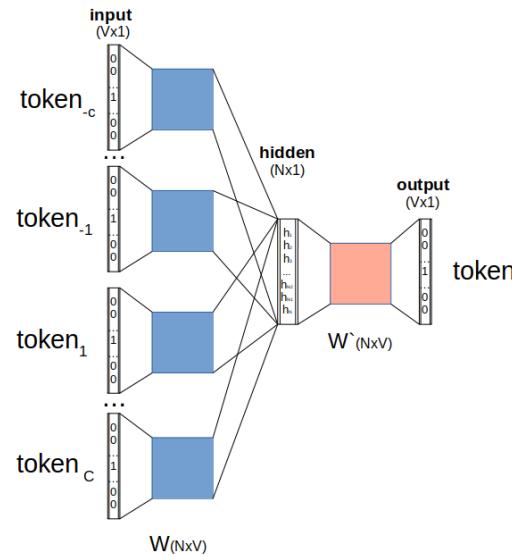


CBOW e Skip-Gram

Vi sono due approcci simili per l'addestramento dei vettori di embedding.

CBOW (sinistra) punta a ricostruire un token mancante passando in input il contesto. Le matrici in ingresso sono tenute costanti.

Skip-Gram (destra) prova a ricostruire invece le parole più probabili dato un termine. Le matrici in uscita possono variare.



Char-RNN

Un altro approccio è quello di codificare tutta la conoscenza all'interno di un modello.

In questo modo non abbiamo una rappresentazione densa vettoriale ma è il modello stesso che contiene l'informazione su come interpretare e digitare i caratteri.

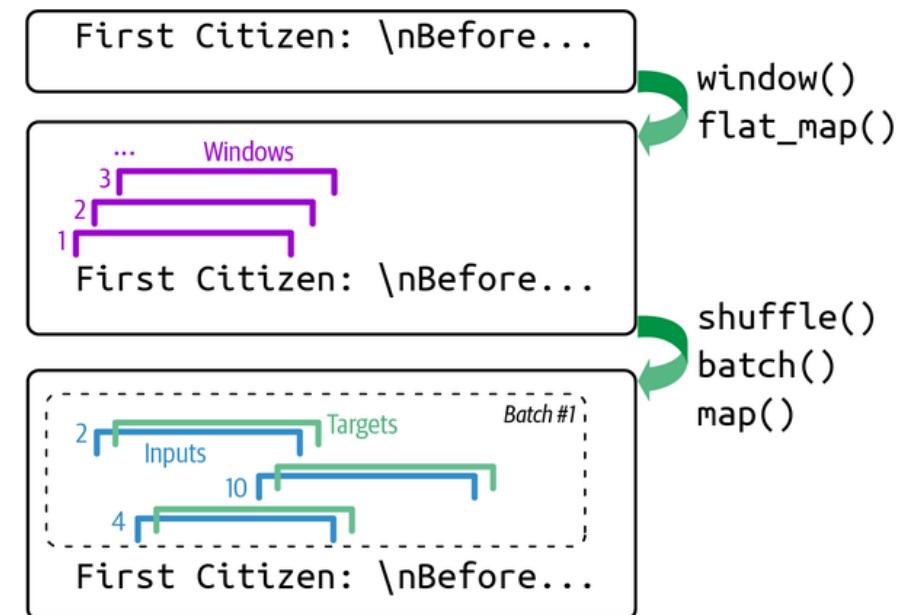
Visto che i caratteri sono una sequenza potremmo usare una RNN che sia in grado di prevedere il singolo carattere successivo in una cosiddetta CharRNN.

Char-RNN

Per costruire la nostra CharRNN ci serve innanzitutto codificare il nostro dataset in modo che, data una sequenza, preveda il carattere successivo man mano viene svolta.

Possiamo poi codificare una semplice RNN per prevedere il carattere successivo codificato OneHot.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=n_tokens, output_dim=16,
                              batch_input_shape=[1, None]),
    tf.keras.layers.GRU(128, return_sequences=True, stateful=True),
    tf.keras.layers.Dense(n_tokens, activation="softmax")
])
```



Temperature

Nel caso della generazione del testo si rischia di finire in alcuni loop per cui viene generata sempre la stessa parola ripetutamente.

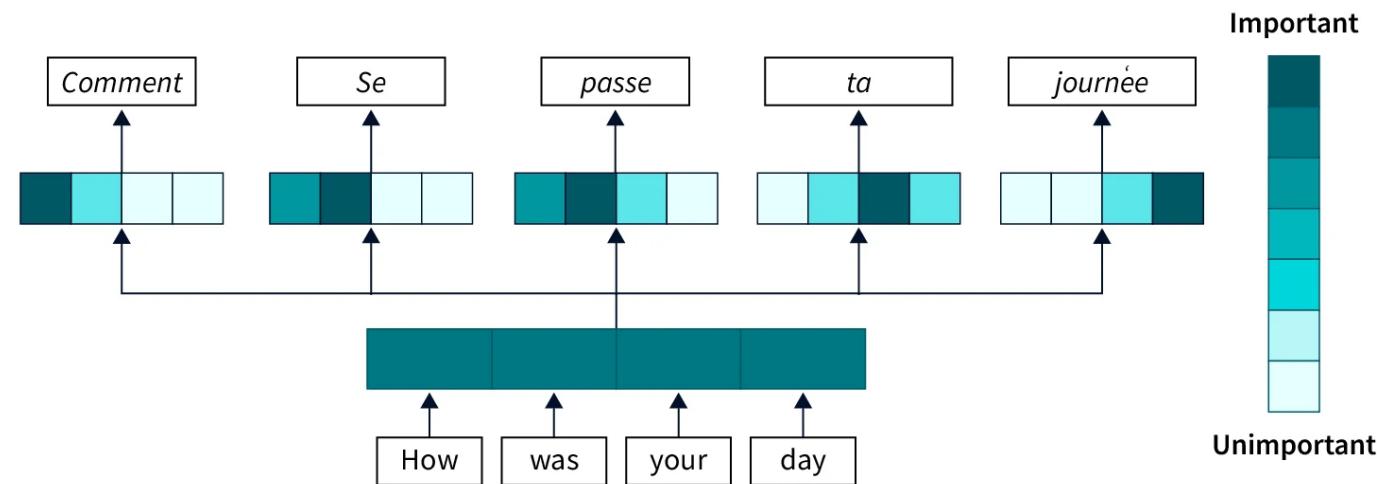
In questi casi si usa il concetto di **Temperature**, ovvero si divide la probabilità ottenuta di ogni output per un valore di temperatura e si sceglie a random tra i più probabili.

Temperature basse (tendenti ad 1) prediligeranno sempre e solo l'output più probabile e sono usate ad esempio nella generazione di equazioni, mentre temperature alte (ad esempio 2-3) tendono a scegliere a random tra output diversi e quindi a generare output più fantasiosi.

Attention

Un concetto fondamentale che ha rivoluzionato gli ultimi anni del machine learning è l'attention mechanism.

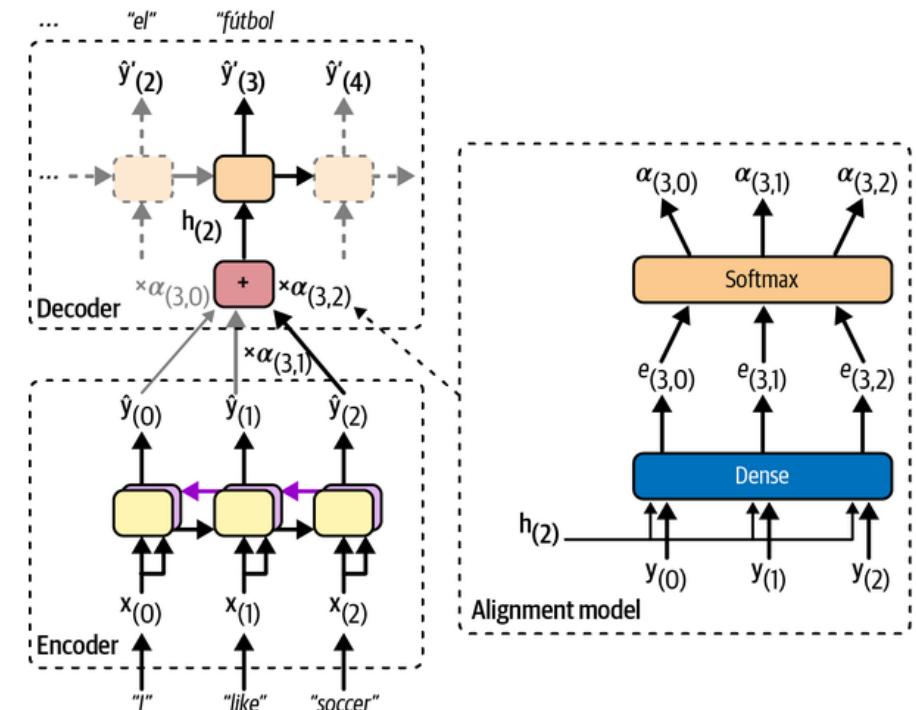
Se selettivamente la nostra rete può concentrarsi solamente su parti diverse della sequenza riusciamo ad intercettare significati più complessi perché leggiamo parole che magari sono lontane tra loro.



Neural Translation with Attention

Prendiamo l'esempio della Neural Translation, ovvero la traduzione tramite Encoder / Decoder e RNN.

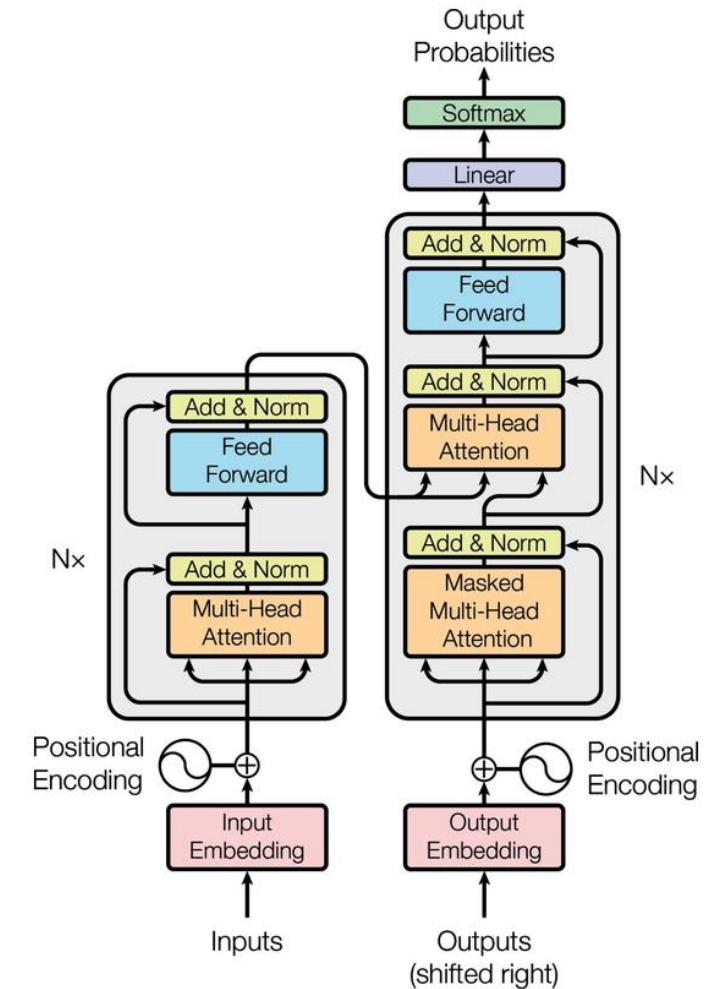
Possiamo dinamicamente calcolare dove il nostro decoder deve prestare attenzione ad ogni istante di tempo tramite un «alignment model» che pesa gli output ad ogni istante di tempo.



Attention is all you need

In un famoso paper chiamato «attention is all you need», un team di Google ha deciso di rimuovere la parte di RNN dal modello di Neural Translation per dare vita ad una architettura interamente basata su **Attention** chiamata **Transformer**.

In questo modo non vi è uno scorrimento del testo ma solamente una codifica della posizione in ingresso (*positional encoding*) che viene data in pasto alla testa che calcola l'attenzione.



Transformers

best transformers of all time

Microphone icon and magnifying glass icon.

All Images Videos Shopping News More Settings Tools

Best Transformers

 Bumblebee Mark Ryan	 Optimus Prime Peter Cullen	 Megatron Hugo Weavi...	 BERT Devlin et al.	 Ironhide Jess Harnell	 Starscream Charlie Adler
--	---	--	---	--	---

@debo

Transformers

Dal 2018 in poi è esplosa l'adozione dei transformer, rendendoli di fatto l'architettura preferita per i task di NLP.

Al momento, l'architettura più famosa più famosa è il **Generative Pre-Trained Transformer**, o GPT di OpenAI.



GPT vs Human

I modelli generativi sono estremamente potenti ma possono essere anche facili da ingannare.

Sfidate Gandalf a farvi dire la password

<https://gandalf.lakera.ai/>





15 – Autoencoders, GANs & Diffusion Models

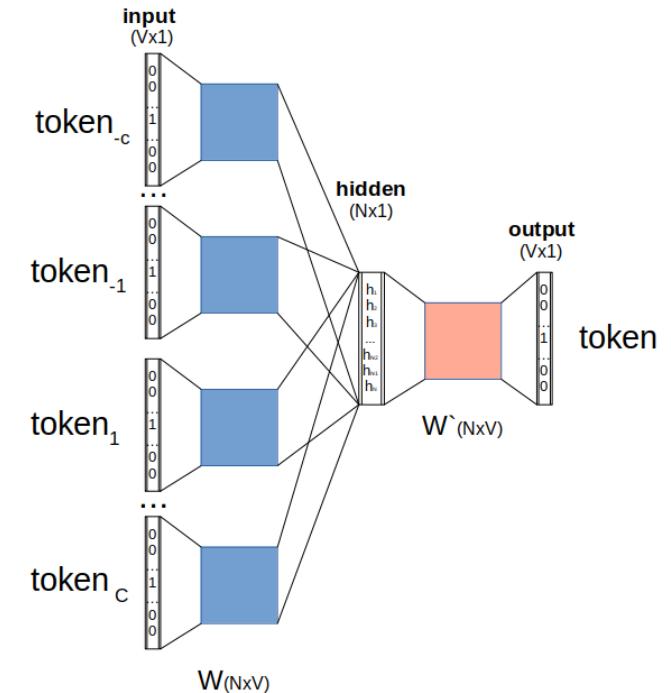
Daniele Gamba

2022/2023

Autoencoders

Abbiamo già visto gli autoencoder come una struttura ottima per comprimere informazioni molto sparse in spazi vettoriali più piccoli e densi.

Una delle applicazioni, nel testo, ci consente di ridurre un vettore molto sparso di più di 50.000 parole codificate one-hot, in un vettore di sole 300 variabili, molto più trattabile.



Autoencoder

Gli autoencoder imparano in modo non-supervisionato una *rappresentazione latente*.

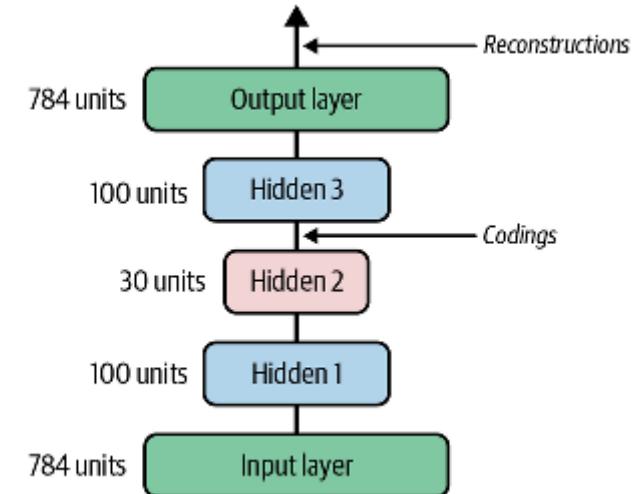
Questo metodo di comprimere e decomprimere l'informazione è molto utile non solo per ridurre la dimensionalità ma anche per

- Identificare dati anomali
- Ricostruire pattern mancanti
- Generare nuovi dati

Autoencoder

Abbiamo visto che un autoencoder è definito da un **encoder** ed un **decoder**.

L'encoder comprime preservando il massimo dell'informazione, il decoder la decomprime cercando di ricostruirla nella forma originaria. La funzione di costo è semplicemente la differenza tra ingresso ed uscita.



```
stacked_encoder = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(30, activation="relu"),
])

stacked_decoder = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(28 * 28),
    tf.keras.layers.Reshape([28, 28])
])

stacked_ae = tf.keras.Sequential([stacked_encoder, stacked_decoder])

stacked_ae.compile(loss="mse", optimizer="adam")
```

Autoencoder

L'esempio precedente imposta un autoencoder per comprimere le immagini del Fashion MNIST, un dataset di immagini di vestiti, dalla loro dimensione originale di 28×28 a un solo vettore di 30 valori per poi decomprimerli.

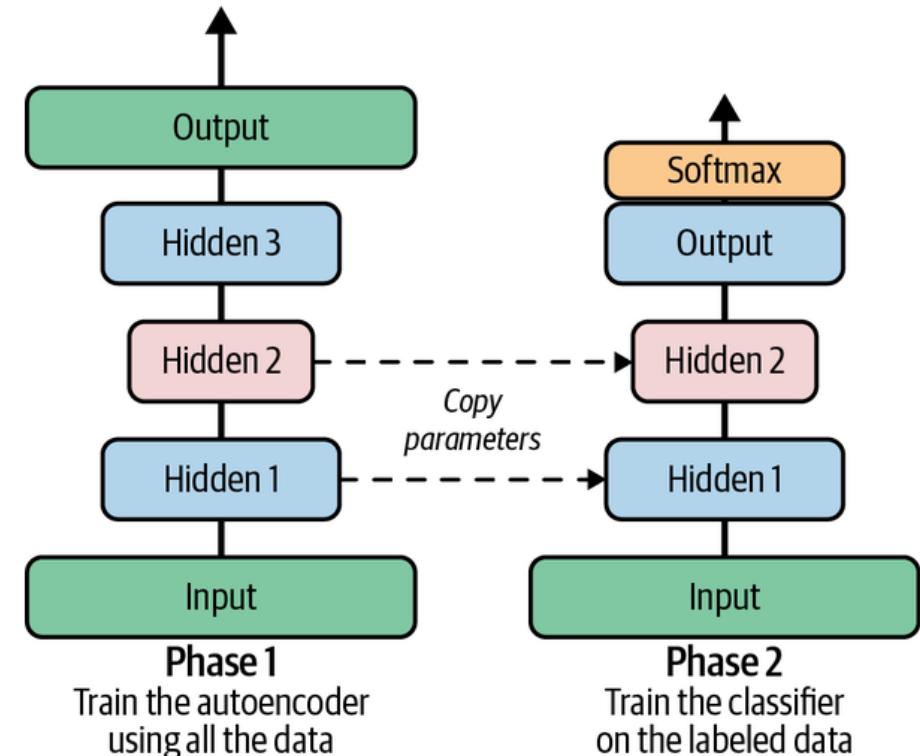
Visualizzando i risultati è chiaro che passare da 784 variabili a 30 perdiamo informazione, ma la rete riesce comunque a preservare le informazioni principali.



Pre-train con autoencoder

Uno dei vantaggi principali degli autoencoder è quello di estrarre features altamente significative in modo totalmente non-supervisionato.

Grazie a questa caratteristica possiamo costruire modelli anche avendo a disposizione pochi dati etichettati e molti senza label. Possiamo infatti addestrare il nostro encoder e decoder, per poi rimuovere il secondo ed agganciare una testa di classificazione con molti meno parametri.

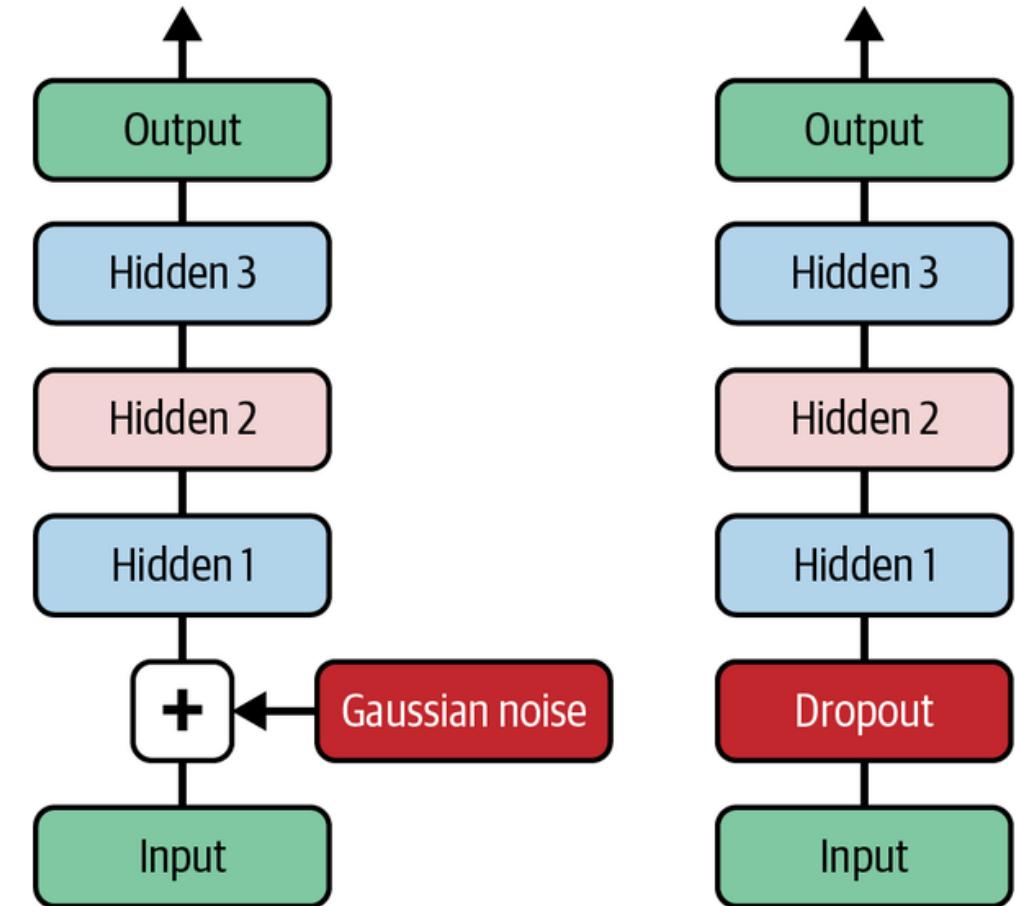


Denoising autoencoder

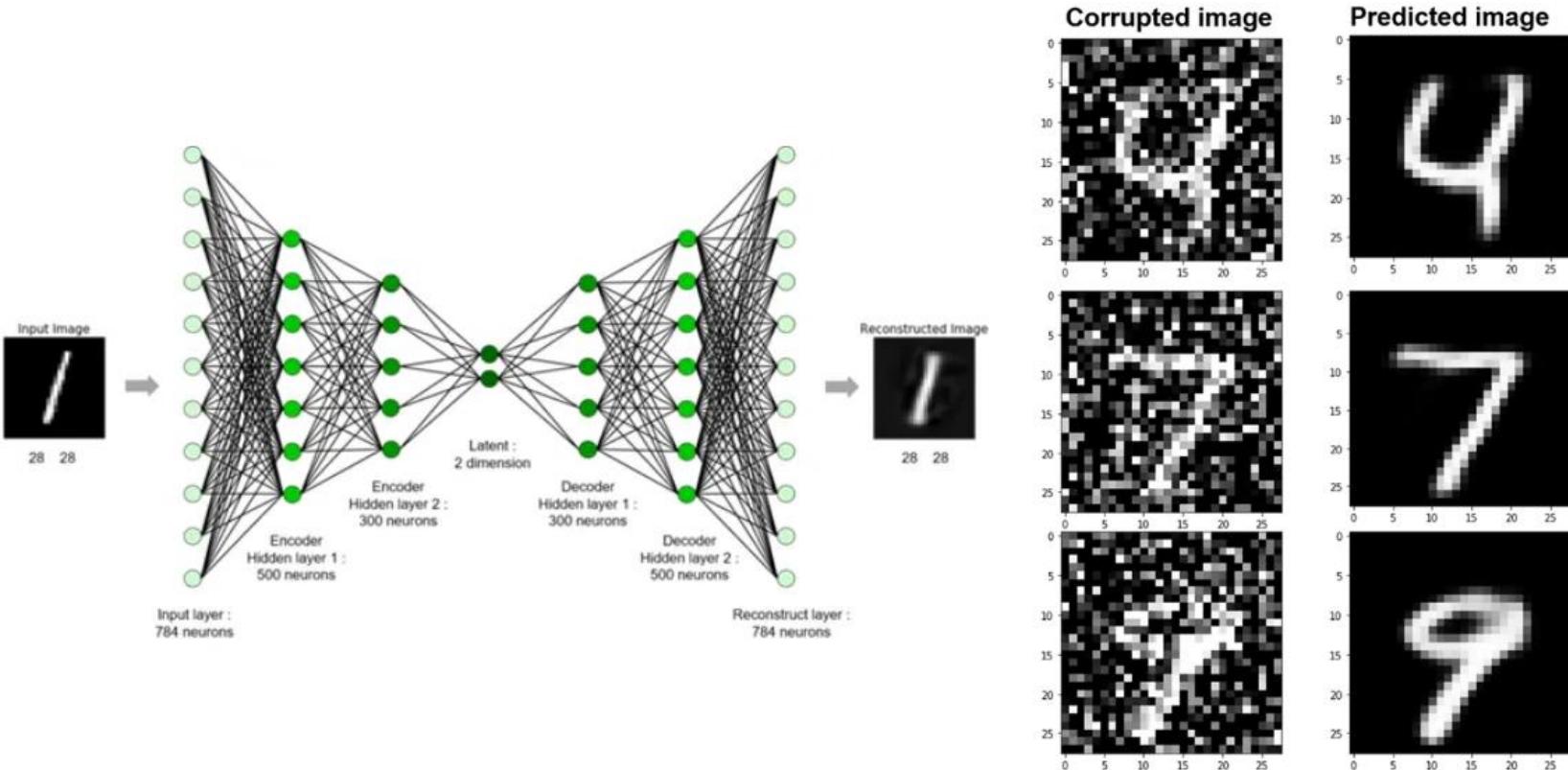
Un altro approccio per far imparare features utili all'autoencoder è quello di sommare noise o aggiungere del dropout in ingresso.

In questo modo il nostro input è rovinato e deve esser ricostruito dalle altre features combinandole.

Oltre ad esser molto utile in generale come pre-addestramento, possiamo usare questa tecnica anche per pulire dei dati potenzialmente sporchi all'interno del dataset o come vero e proprio task.



Denoising autoencoder

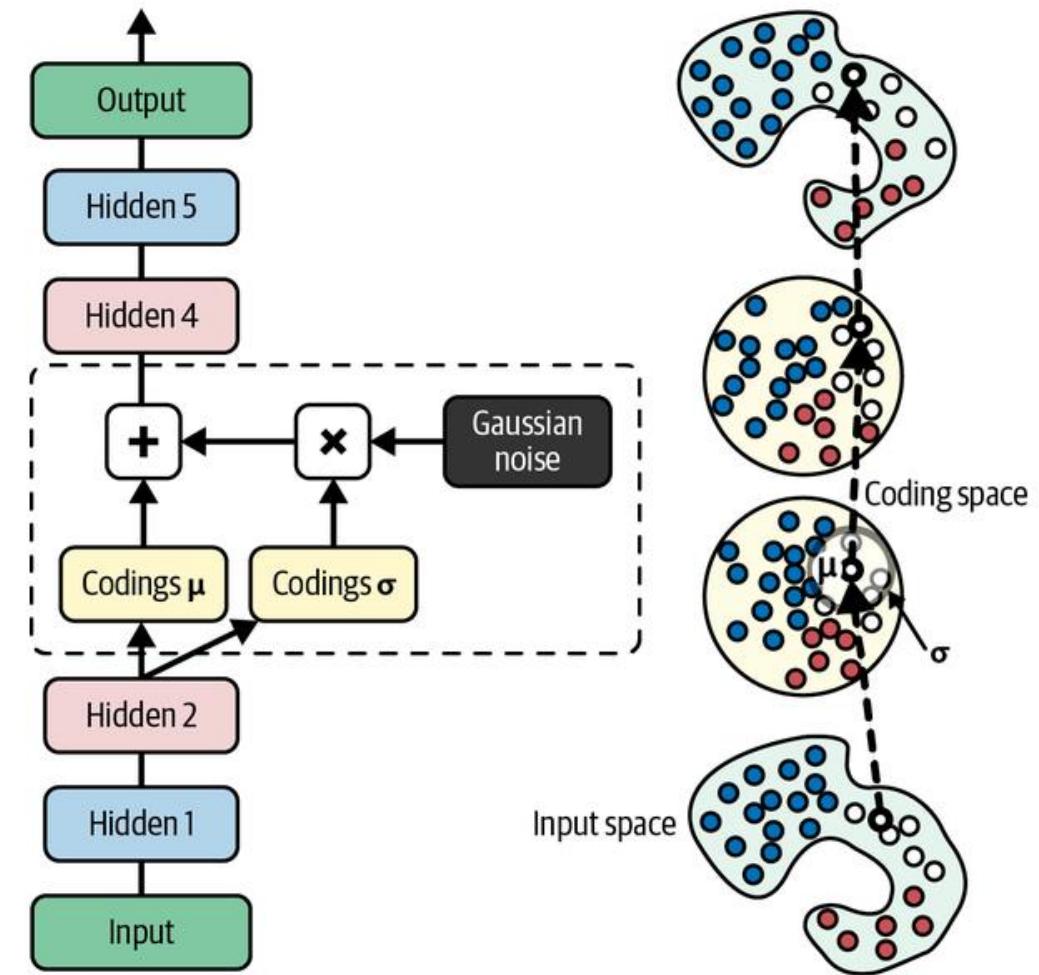


Variational autoencoder

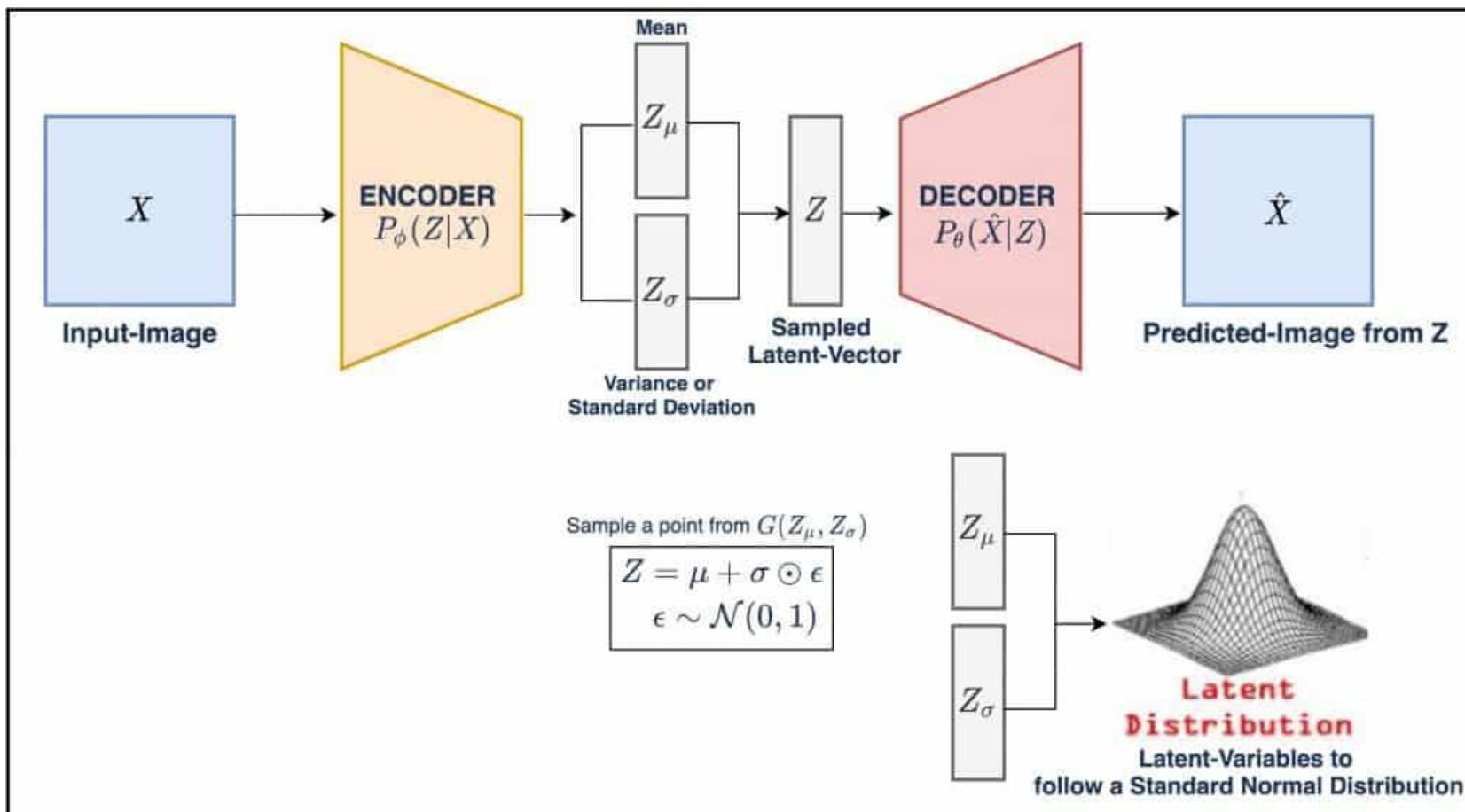
Nel 2013 è stata inventata una nuova categoria di autoencoder chiamati variational.

Sono innanzitutto oggetti probabilistici, sia in training che in inferenza, e sono generativi, nel senso che possono produrre nuovi output simili a quelli su cui sono stati addestrati.

Il nostro encoder stima una media e una varianza per ogni posizione del vettore codificato. Queste vengono usate per campionare un nuovo esempio (frutto della somma di noise gaussiano) che verrà poi decodificato dal decoder.



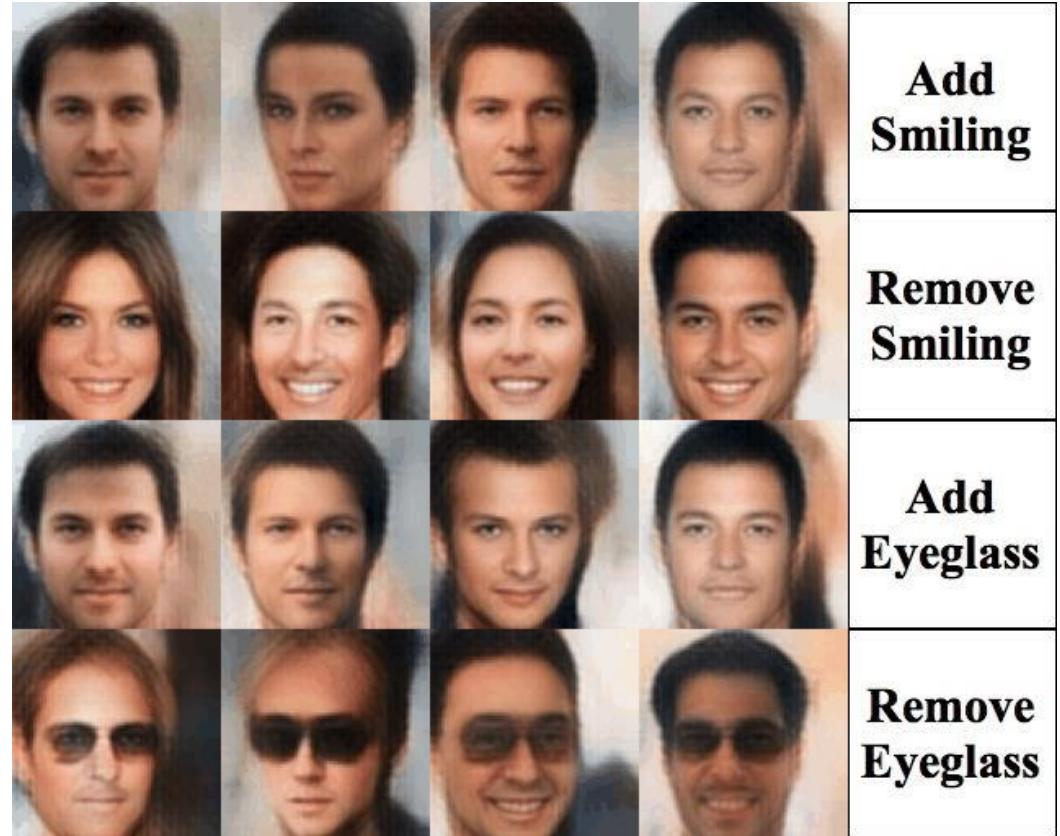
Variational autoencoder



Variational autoencoder

Una volta addestrato il VAE, il nostro spazio latente risulta essere interamente campionabile, per cui possiamo andare a codificare un esempio per prendere un punto di partenza e poi iniziare a modificare il vettore per vedere come questo cambia.

Oppure opportunamente fissando in training certi valori possiamo anche associare alcune caratteristiche in modo più o meno esplicito a parti del nostro embedding.



Variational autoencoder

Benché ci consenta di effettivamente generare nuovi esempi, l'autoencoder sfrutta comunque una compressione e quindi una perdita di dettaglio importante.

Nei task di generazione immagini un altro approccio ha preso il sopravvento per la qualità dei dettagli che era in grado di generare.



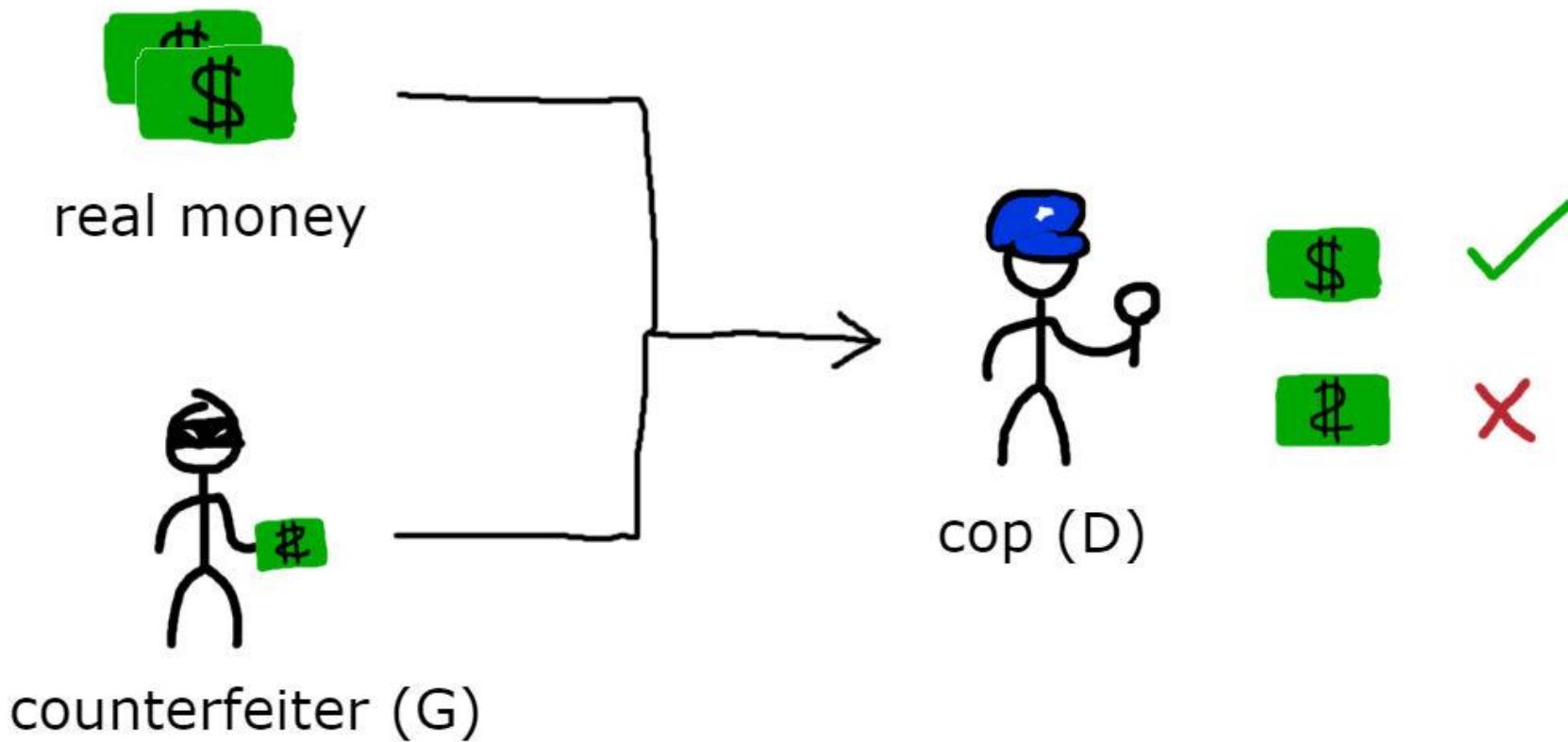
Le **Generative Adversarial Network** sono reti *generative* per creare nuovi dati verosimili.

Si compongono di due reti avversarie

- Un **generatore**, che ha come compito quello di forgiare nuovi esempi verosimili
- Un **discriminatore**, che deve discriminare se l'esempio fornito è vero o generato

Il generatore cerca di imbrogliare il discriminatore mentre il secondo cerca di scoprire sempre se è vero o meno il dato che gli è stato fornito.

GAN

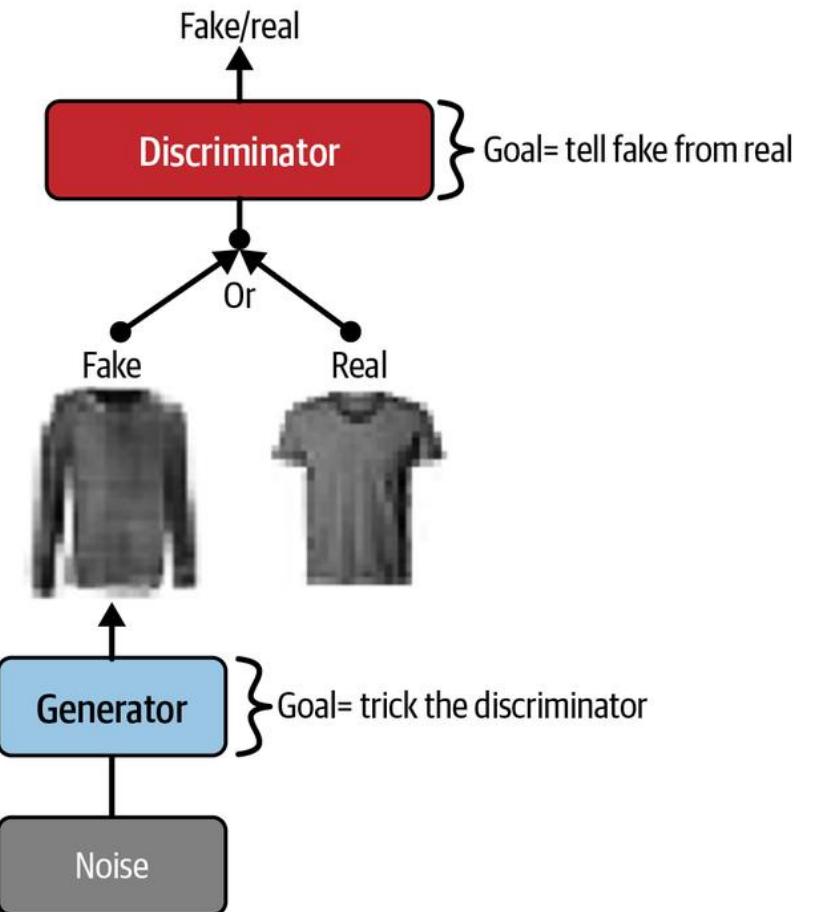


GAN

Il generatore parte con la sua generazione da un vettore noise random.

Questo stesso vettore è equivalente, terminato al training, all'embedding del VAE che possiamo usare per generare nuovi dati.

Le GAN sono più difficili da addestrare rispetto ad altri modelli perché avremo due reti da addestrare insieme e di cui si cerca di bilanciare l'apprendimento.



GAN

Le GAN non hanno un ciclo di training lineare come quello di altri modelli, per cui dobbiamo scriverci il ciclo di ottimizzazione da soli.

Nello specifico, inizieremo ad addestrare il discriminatore generando un nuovo esempio del generatore da del noise ed aggiungendolo al batch di training.

Successivamente addestreremo il generatore usando l'intera sequenza per semplificare il codice.

```
codings_size = 30

Dense = tf.keras.layers.Dense
generator = tf.keras.Sequential([
    Dense(100, activation="relu", kernel_initializer="he_normal"),
    Dense(150, activation="relu", kernel_initializer="he_normal"),
    Dense(28 * 28, activation="sigmoid"),
    tf.keras.layers.Reshape([28, 28])
])
discriminator = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    Dense(150, activation="relu", kernel_initializer="he_normal"),
    Dense(100, activation="relu", kernel_initializer="he_normal"),
    Dense(1, activation="sigmoid")
])
gan = tf.keras.Sequential([generator, discriminator])

def train_gan(gan, dataset, batch_size, codings_size, n_epochs):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        for X_batch in dataset:
            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.train_on_batch(X_fake_and_real, y1)
            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            gan.train_on_batch(noise, y2)

train_gan(gan, dataset, batch_size, codings_size, n_epochs=50)
```

GAN

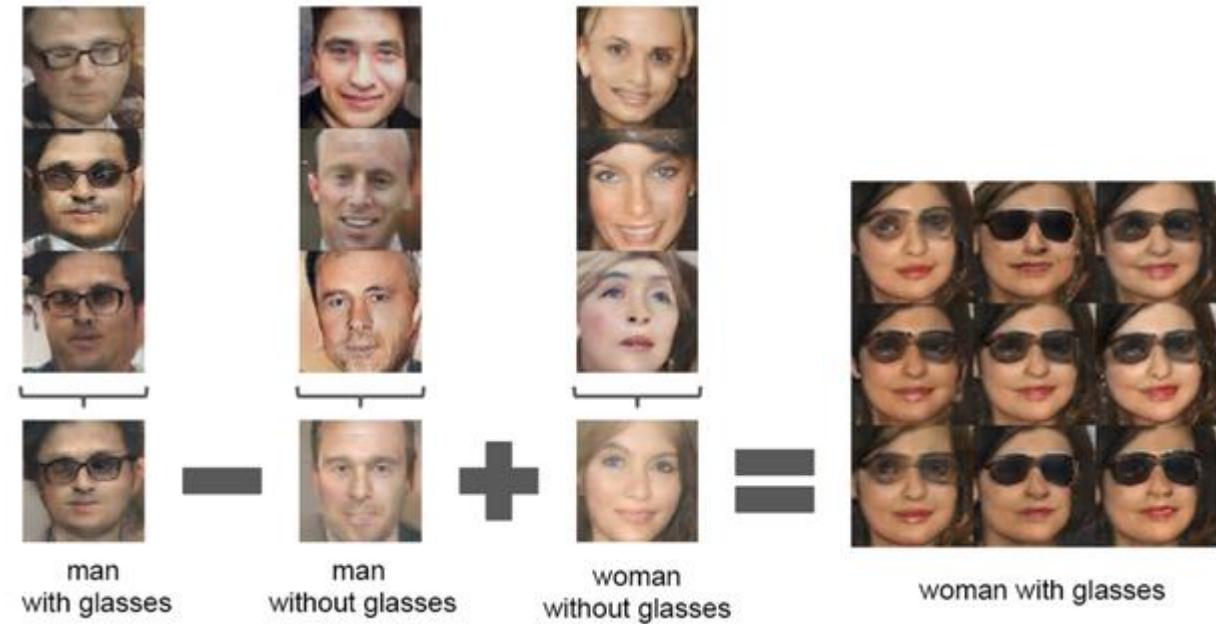
Le GAN sono più difficili da addestrare anche perché, ad esempio, il generatore potrebbe scoprire delle fallo del discriminatore e iniziare a generare solamente oggetti di una classe in cui il discriminatore fa più fatica.

Inoltre è facile che il training sia instabile avendo due reti che spingono una contro l'altra, risultando in un collasso dell'addestramento che porta a zero tutti i pesi del discriminatore o del generatore.

DCGAN

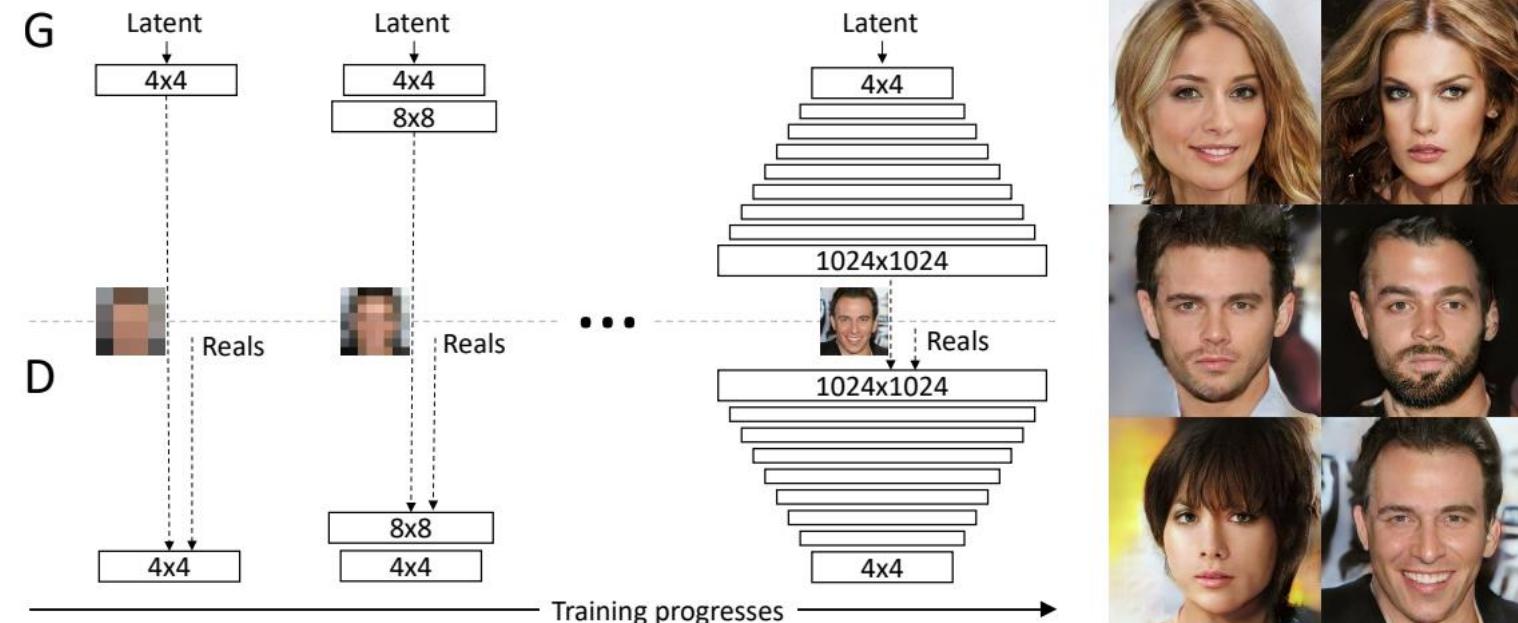
Dal 2015 le GAN costruite a partire da reti deep convoluzionali, principalmente usate per la generazione di immagini, si chiamano Deep Convolutional GAN o DCGAN.

Le DCGAN sono completamente convoluzionali, non usano pooling ma stride e usano molta batch normalization. Hanno aperto la strada a generare immagini sempre più complesse e condividono con il Word2Vec la proiezione dei dati in uno spazio in cui è possibile far delle operazioni matematiche tra le immagini (sfruttando il loro embedding).



Progressive Growing GAN

I risultati sono sorprendenti ma c'era ancora ampio margine di miglioramento, nel 2018 un team di Nvidia ha proposto di addestrare delle GAN a risoluzioni piccole e crescere via via di risoluzione aggiungendo layer convoluzionali sia a discriminatore che a generatore. In questo modo è estremamente più facile addestrare GAN con un livello di dettaglio maggiore. Tutti i pesi sono lasciati comunque addestrabili.

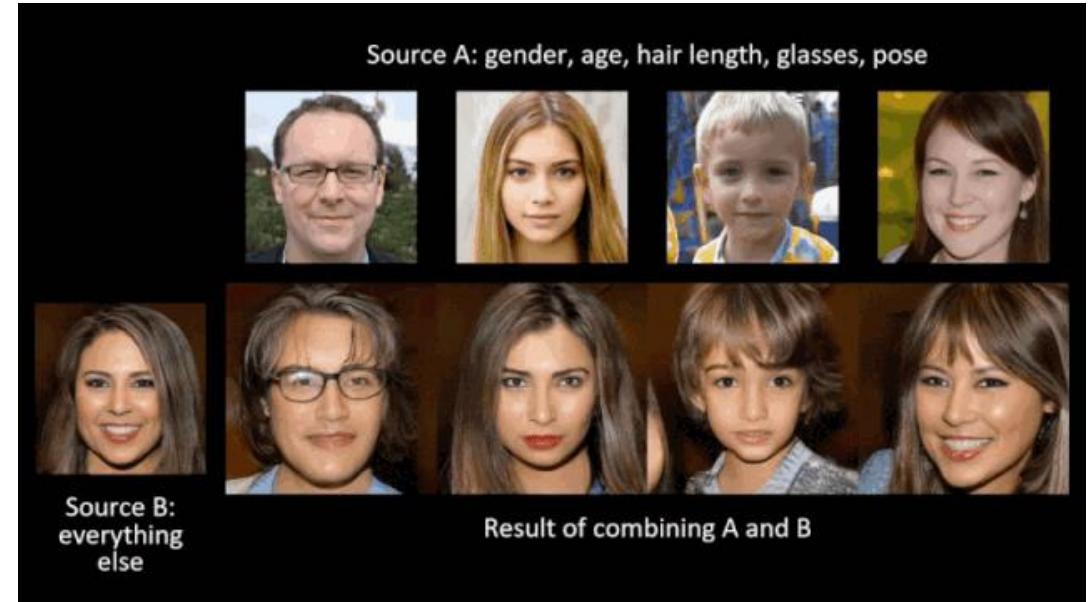


StyleGAN

Sempre Nvidia nel 2018 avanza ulteriormente le possibilità delle GAN presentano le StyleGAN.

Nelle StyleGAN il generatore si divide tra stile, che viene codificato come vettori «noise» da sommare all'immagine, e struttura che viene continuamente disturbata da noise.

In questo modo lo stile viene codificato a diversi livelli e sommato all'immagine mentre la struttura rimane invariata.

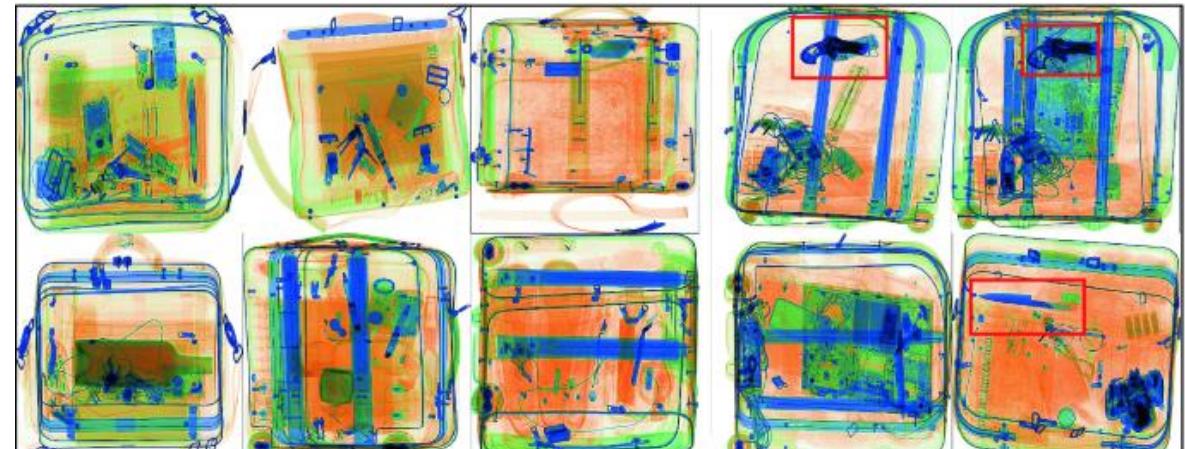
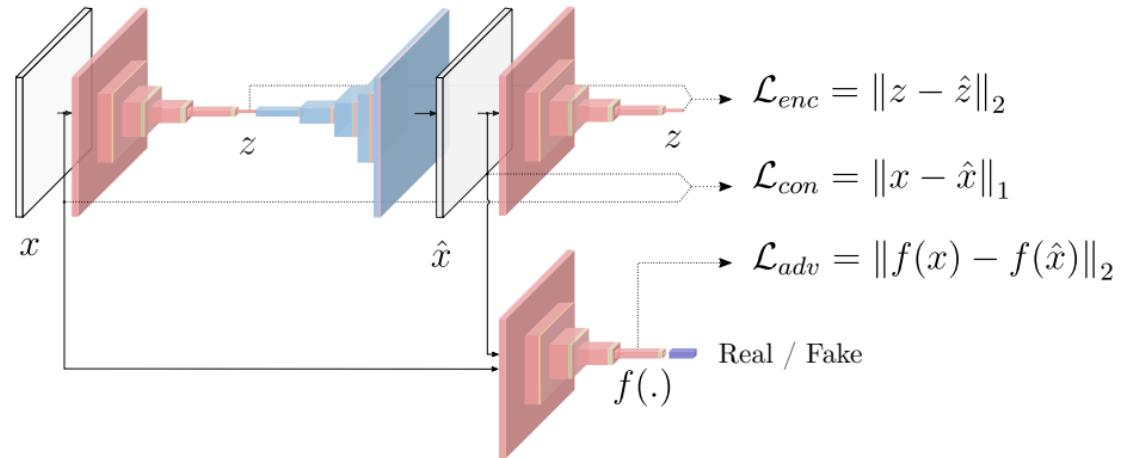


GANomaly

Come abbiamo visto VAE e GAN imparano la struttura del dataset e a generare nuovi esempi verosimili. Possiamo usarli anche per il processo inverso, ovvero capire se un esempio è verosimile o anomalo.

Uno dei modelli più interessanti è la GANomaly, in cui si cerca di unire le caratteristiche di VAE e GAN per identificare le parti anomale di un'immagine.

Possiamo sfruttare questo approccio quando abbiamo tanti esempi buoni e pochi o quasi nessuno anomalo.



(a) Normal Data (X-ray Scans)

(b) Normal + Abnormal Data (X-ray Scans)

Approfondimento - Anomaly Detection

Come abbiamo già citato diverse volte durante il corso, uno dei problemi che ricadono sotto il Machine Learning è l'Anomaly Detection.

Esistono due grandi categorie

- **Outlier** detection, ovvero trovare i dati sporchi all'interno del nostro dataset, ovvero gli outlier
- **Novelty** detection, ovvero una volta ottenuto un nuovo dato capire se è o meno un outlier

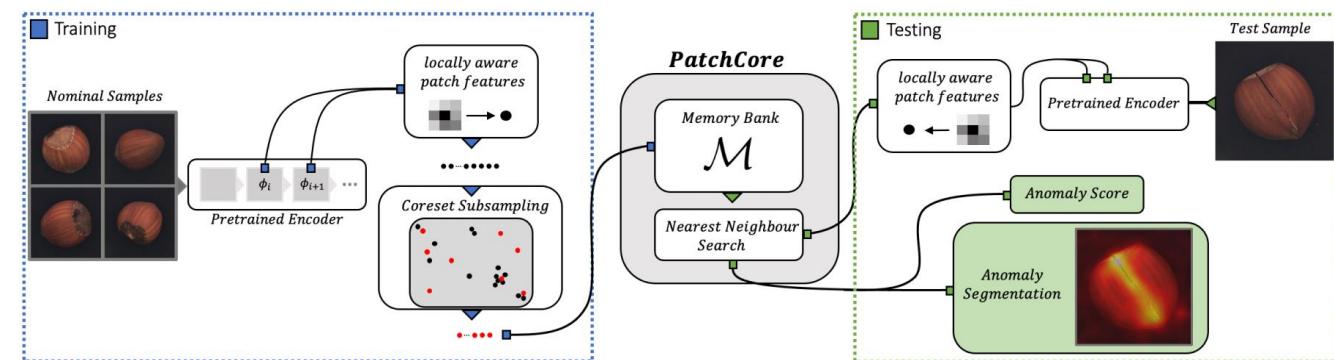
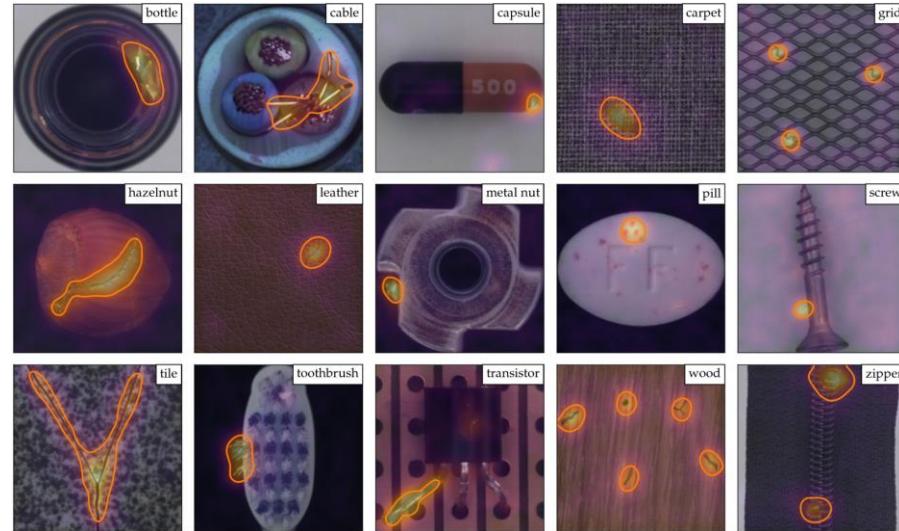
I primi spesso si trovano andando iterativamente a ripassare il dataset per valutare i casi in cui i nostri modelli sbagliano, andando a visualizzare i dati o andando ad applicare diverse tecniche.

Approfondimento - Anomaly Detection

Abbiamo visto, affrontando le SVM, che una delle tecniche è la OneClass SVM. Altrimenti possiamo usare anche l'encoder del nostro VAE per stimare se il nostro esempio è o meno simile agli altri disponibili.

Nell'elaborazione immagini invece dal 2021 una delle tecniche più promettenti si chiama **Patchcore**.

Si basa sull'estrarre da una backbone un set di features, poste in una **Memory Bank**, che vengono selezionate e confrontate tramite Nearest Neighbour con il nuovo esempio da valutare.

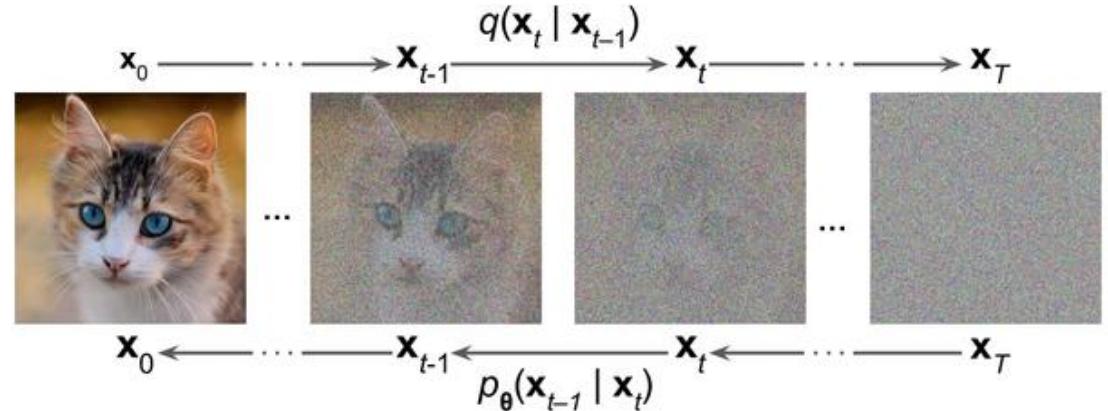


Diffusion Models

I Diffusion Model sono modelli generativi che provano iterativamente a ricostruire un output andando a rimuovere il noise presente.

Dal 2015 in poi, e in particolare dal 2020 e 2021 hanno acquisito grande importanza perché hanno battuto le GAN nella qualità dei risultati che riescono a produrre, con StableDiffusion come principale modello pubblico.

Il training è fatto fornendo ad ogni istante di tempo un'immagine con sempre più noise, mentre il modello deve continuamente provare a sottrarre questo noise per preservare l'informazione al suo interno.



16 – Reinforcement Learning

Daniele Gamba

2022/2023

RL

Il reinforcement learning, o apprendimento per rinforzo, è un altro dei modi con cui i modelli possono apprendere.

Si basa sul concetto di *agente* che tramite *osservazioni* decide di compiere *azioni* in un *ambiente* per massimizzare il suo *reward*.

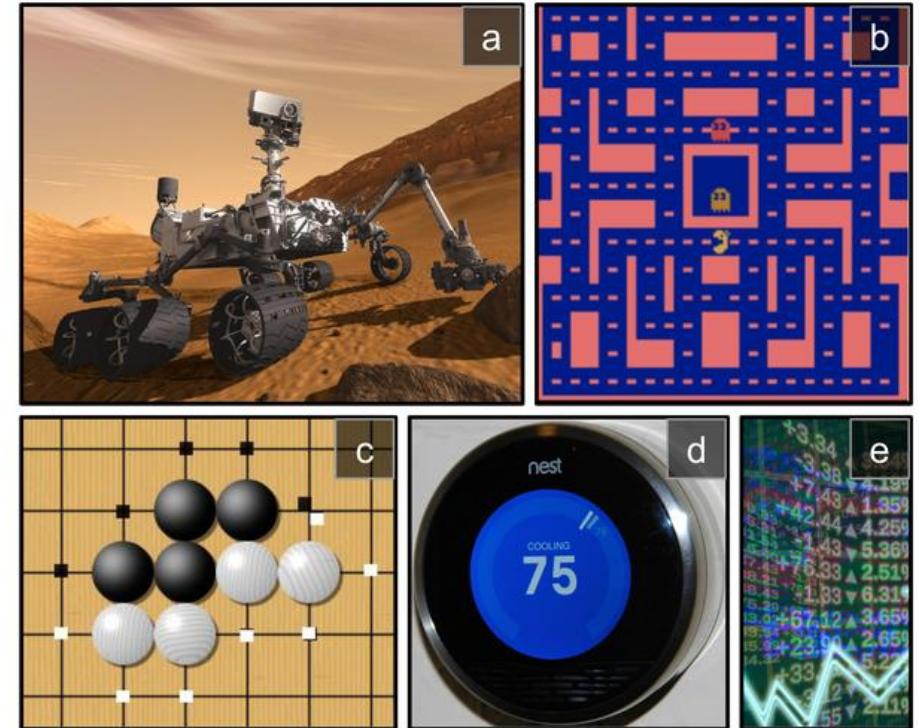
Dovendo interagire con l'ambiente, il RL ha avuto grande successo nel gaming, inanellando un successo dietro l'altro e riuscendo a battere campioni di Atari, Go, Dota, Starcraft, ecc.



Esempi di RL

Alcuni esempi di RL

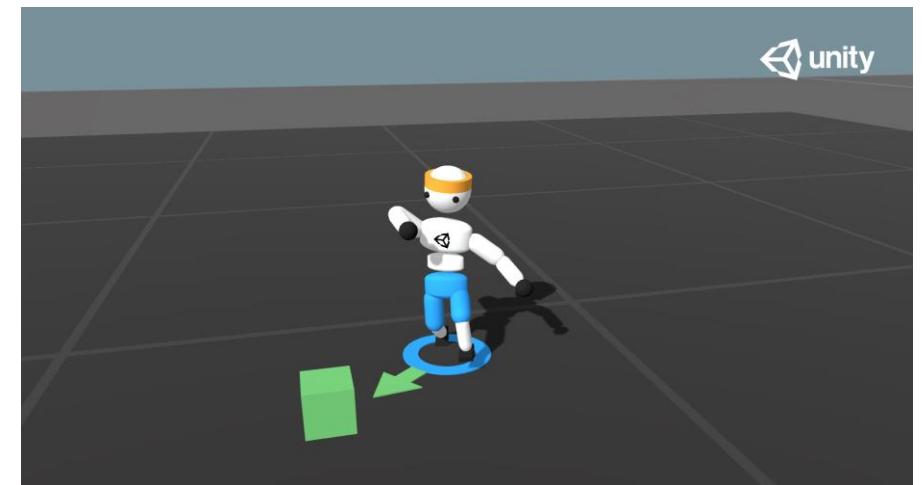
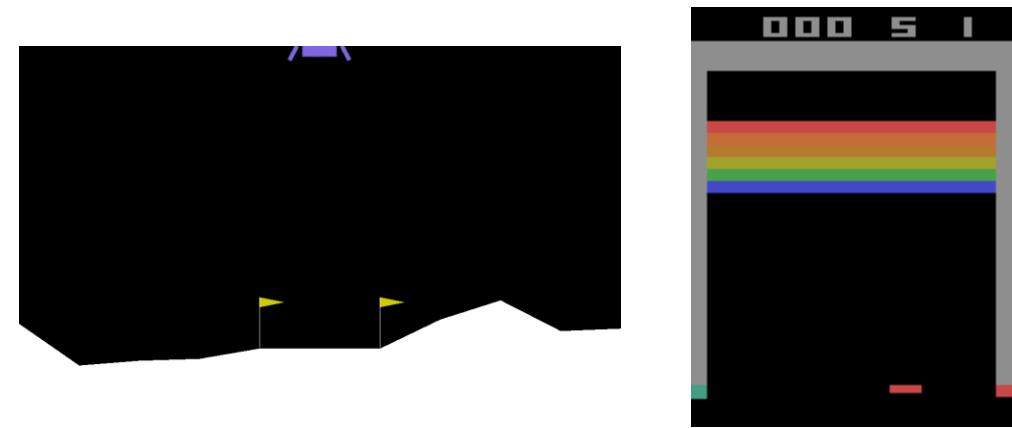
- Un robot controllato da un agente, che osserva il mondo tramite camere e sensori, compie azioni comandando i motori e riceve un reward avvicinandosi al suo obiettivo, mentre viene penalizzato nel caso si allontani
- Un agente gioca a PacMan, l'ambiente è la simulazione dell'Atari e le azioni sono i possibili tasti da premere, le osservazioni sono screenshot del gioco e il reward è il punteggio raggiunto
- Un termostato smart riceve reward positivo quando si avvicina alla temperatura target risparmiando energia, mentre viene penalizzato ogni volta che l'umano deve modificare manualmente la temperatura
- Un trader, ..



OpenAI/Farama Gym

Gymnasium (ex OpenAI Gym) è una raccolta di environment in cui poter addestrare diversi agenti.

Dispone di API in Python per poter esser facilmente integrato con il codice degli agenti ed eseguire ripetutamente azioni e training nei diversi ambienti.

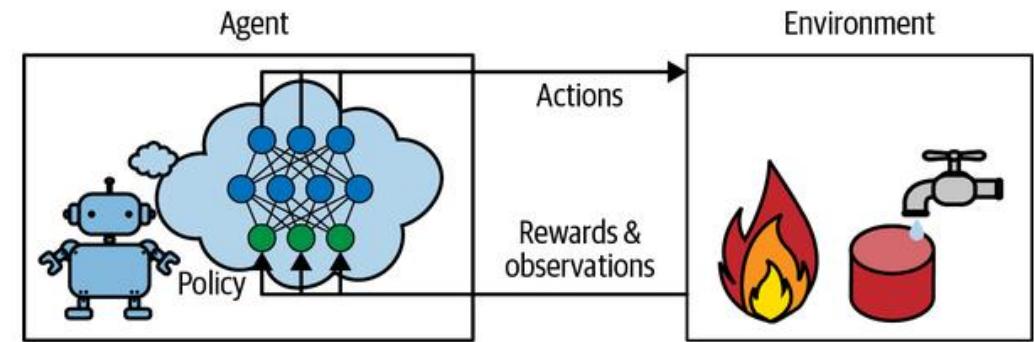


Policy search

L'algoritmo che l'agente usa per determinare le sue azioni è la sua *policy*.

La policy potrebbe essere una NN o un altro algoritmo, ogni policy può avere uno o più *parametri* che vengono man mano aggiustati per trovare la policy ottima in un processo chiamato *policy search*.

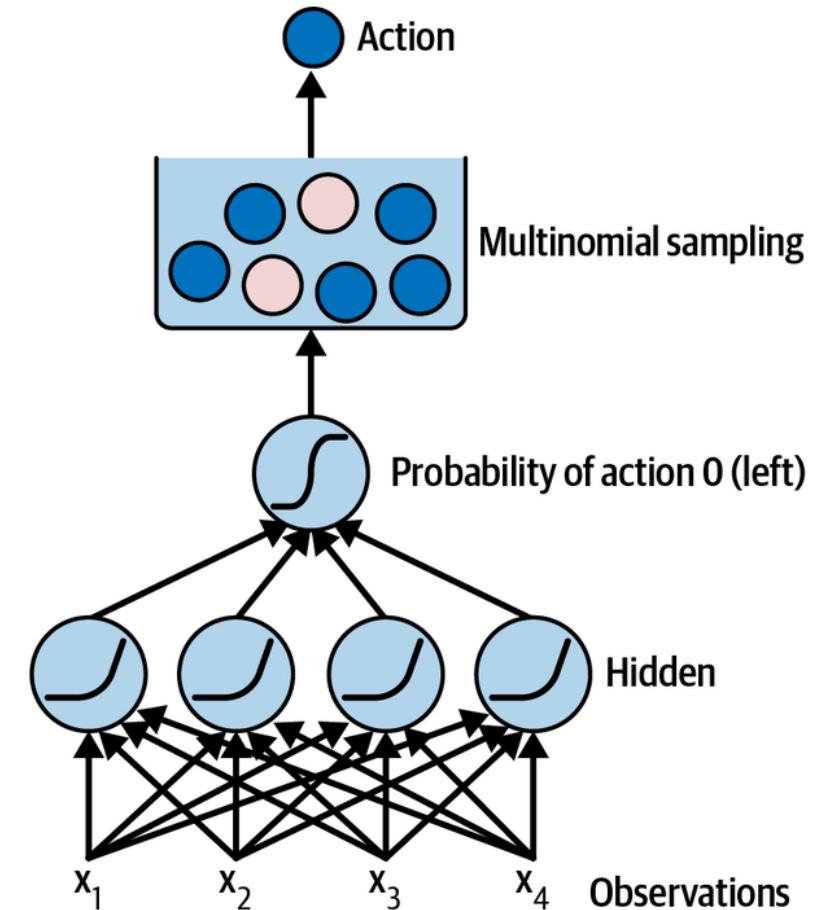
Potremmo scegliere di tirare a random l'azione successiva, o usare algoritmi genetici per randomizzare solo una parte dei parametri della policy.



Exploration / Exploitation

Nel caso decidessimo di affrontare un ambiente, se ottimizzassimo direttamente il nostro modello, tenderemmo a fare exploitation della policy corrente scegliendo sempre l'azione più probabile senza mai porci il problema che possano esisterne di altre più efficienti.

Per questo va sempre bilanciata una ricerca di soluzioni migliori, l'**exploration**, con l'ottimizzazione poi rispetto a quelle che riteniamo più performanti, l'**exploitation**.



Punteggi a lungo termine

Una volta definita la nostra rete dobbiamo scegliere ad ogni istante qual è la mossa migliore per massimizzare i nostri reward.

Se conoscessimo ad ogni istante la mossa migliore da eseguire il tutto sarebbe un semplice problema di apprendimento supervisionato, non dovremmo far altro che addestrare la rete a fronte di certi input a prendere determinati output.

Purtroppo non conosciamo ad ogni istante quale sia la mossa migliore ma dobbiamo per forza arrivare alla fine del gioco per valutare il nostro punteggio. Da qui la necessità di scegliere delle policy che ci consentano di prendere decisioni senza aver visibilità sul risultato immediato.

Dobbiamo quindi assegnare ad ogni mossa una probabilità che ci porti in futuro a massimizzare il reward.

Punteggi a lungo termine

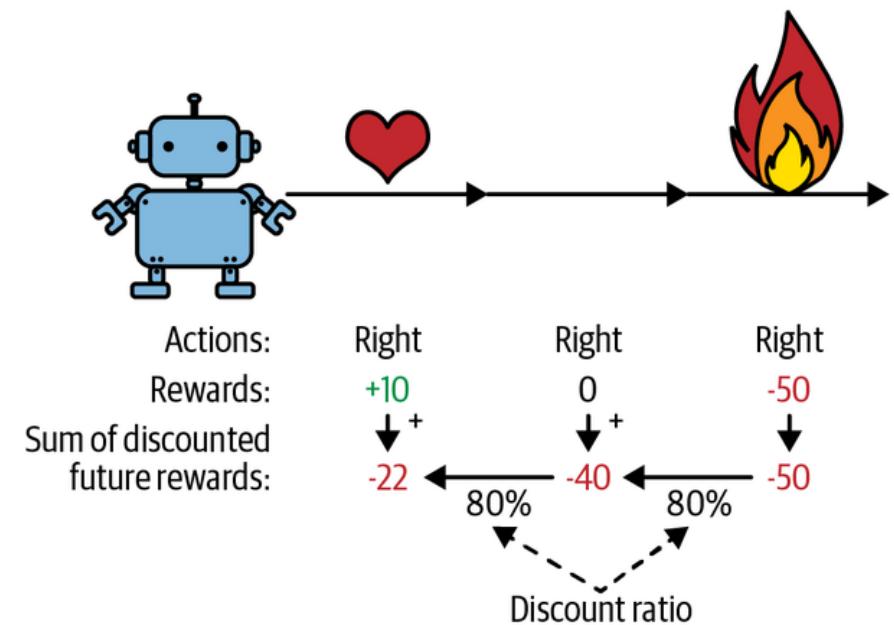
Un problema è quello di assegnare un punteggio ad ogni mossa, infatti una volta raggiunto il nostro reward non è detto che siamo in grado di assegnare a ciascuna di esse se ha avuto un impatto positivo o negativo.

Questo problema è noto come *credit assignment problem*.

Discount factor

Un modo di assegnare il reward alle diverse azioni è quella di pesare il reward futuro a ritroso tramite un *discount factor* gamma che va a scontare nel tempo l'impatto delle diverse azioni.

Tanto più il discount factor è vicino a zero, tanto più impatteranno solamente le decisioni immediate, tanto più è tendente ad 1, tanto più anche le decisioni più lontane contribuiscono al raggiungimento dell'obiettivo.



Policy Gradients

Un'idea più interessante è quella di valutare il gradiente della policy rispetto al reward per massimizzare il nostro guadagno.

Un modo è descritto dall'algoritmo REINFORCE

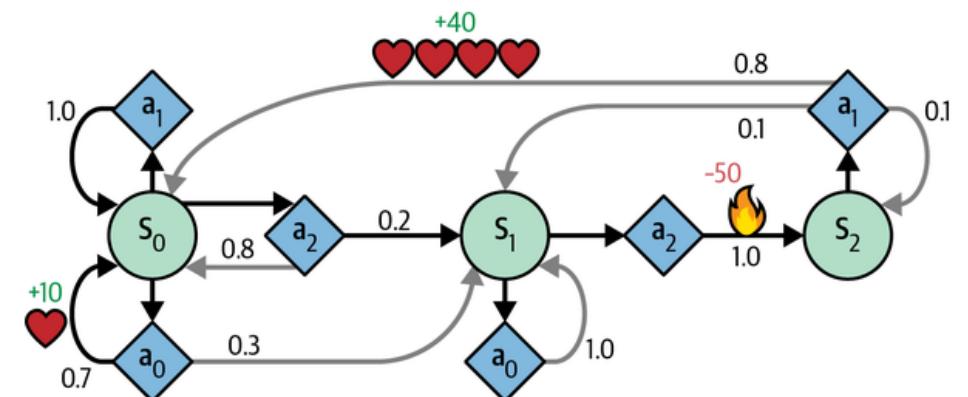
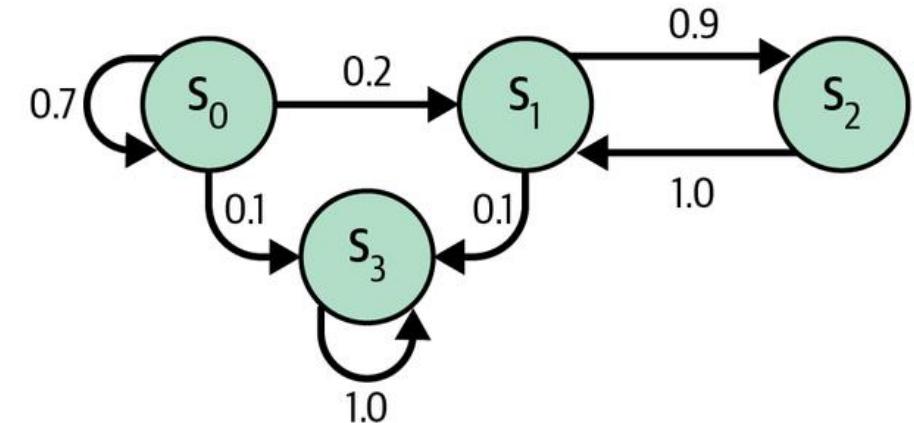
- Per prima cosa giochiamo molte volte contro l'ambiente e calcoliamo i gradienti che rendono più probabile le scelte prese senza applicarli
- Valutiamo poi a ritroso l'impatto delle azioni andando a scontare i punteggi ottenuti
- Se un'azione è stata positiva allora terremo il gradiente positivo, altrimenti applicheremo l'opposto del gradiente
- Calcoliamo la media di tutti i gradienti e facciamo uno step di ottimizzazione

Markov Decision Process

Nel trattare il RL non possiamo non citare i Markov Decision Process, o MDP.

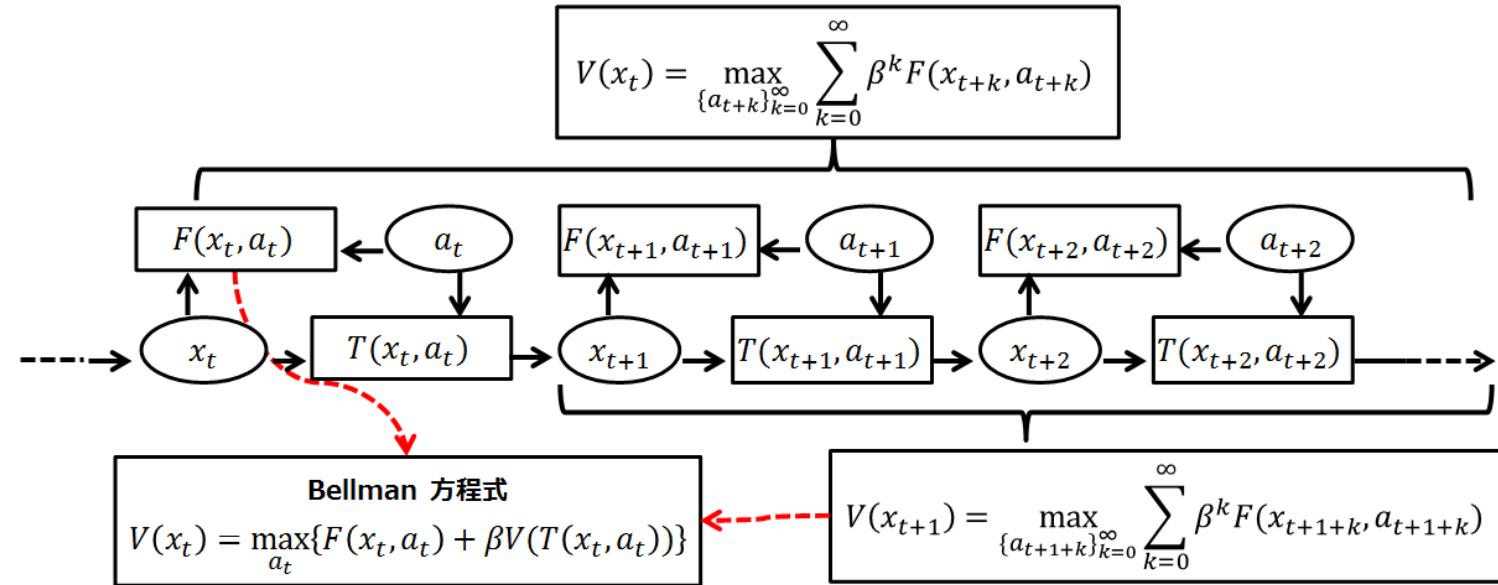
Una Markov Chain è un processo stocastico stabile e senza memoria in cui si passa da uno stato all'altro con una certa probabilità.

Un MDP a differenza della MC è un processo in cui in ogni stato posso scegliere di compiere delle azioni, il cui outcome è definito da delle probabilità.



Ottimalità di Bellman

Bellman ha dimostrato il valore di ogni stato nel caso in cui l'agente agisca in modo ottimo tramite la *Bellman optimality equation* e un algoritmo iterativo che stima lo stato ottimo, *Value iteration algorithm*, che trovate qui sotto per conoscenza.



Q-value

Le due formule precedenti però non ci danno un'indicazione su qual è la policy ottima per l'agente. Per questo sempre Bellman ha introdotto i Quality-values come output di un algoritmo per stimare l'ottima scelta stato-azione.

Il Q-value è calcolato come la somma scontata di tutti i valori futuri di reward associati al raggiungimento dello stato s e all'esecuzione dello stato a .

Q-value iteration would use the previous iteration of the Q-value on the right hand side of the equation

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right)$$

to update the Q value estimate of the current step. Hence, the Q value update for k^{th} step would look like:

$$Q_{k+1}^*(s, a) = \sum_{s'} T(s, a, s') (R(s, a, s') + \gamma \max_{a'} Q_k^*(s', a')) .$$

Temporal Differences

Una delle assunzioni dell'approccio tradizionale di Bellman è la conoscenza dell'MDP e delle probabilità di transizione. Questo chiaramente non è vero nella maggior parte dei casi in cui l'agente ha solo una conoscenza parziale dell'MDP del suo ambiente.

L'algoritmo chiamato Temporal Differences, o TD, tiene conto del fatto che non conosce l'ambiente in cui si trova e di conseguenza all'inizio dell'apprendimento tenderà a fare *exploration* per stimare stati e probabilità di transizione prima di andare a stimare i valori di reward.

Q-learning

Allo stesso modo del TD, il Q-learning è un algoritmo che osserverà un agente giocare quasi a random contro l'ambiente per provare a fare una stima dei Q-value.

Quando la stima si avvicinerà a quelli effettivamente ricevuti, a quel punto inizierà a fare exploitation della policy che massimizza il ritorno in modo greedy.

Durante tutta la fase di addestramento è spesso usata una ε -greedy policy, quindi con una probabilità piccola ε di non scegliere l'azione migliore ma di randomizzarla per continuare ad esplorare.

Deep Q-learning

Fin'ora abbiamo ipotizzato ambienti con un numero finito di stati, ma se pensiamo anche solo ad un videogioco 2D, con premi e nemici generati random, il numero di «stati» da esplorare sarebbe enorme e l'algoritmo impiegherebbe troppo tempo per stimare tutte le probabilità.

DeepMind nel 2013 ha suggerito invece di usare una deep neural network per stimare i Q-value, ovvero una DQN.

Andremo ad ottimizzarla andando iterativamente ad interagire con l'ambiente, osservare i reward, scontarli per le azioni e dando il valore come obiettivo di apprendimento alla DQN.

Altri algoritmi di apprendimento

Sono stati creati negli altri numerosi altri algoritmi di apprendimento, tra cui

- Double DQN
- Prioritized Experience Replay
- Duelling DQN
- Monte Carlo Tree Search
- Actor Critic (tra cui anche A3C, A2C, Soft actor-critic)
- Proximal Policy Optimization
- Curiosity Based-exploration

Hide & Seed



RL in Colab

<https://colab.research.google.com/drive/1g7doH2zk8vKXACoe4R0HPFX7N48ehe01?usp=sharing>