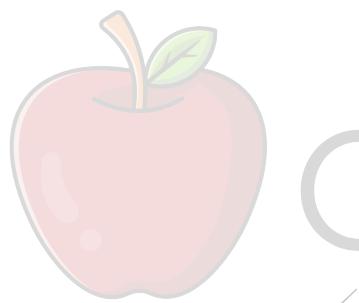


Unknown

2021/22

Appunti Data Science & Machine Learning



CoScienze
Associazione

LUCA FANZINI

Sommario

Big Data.....	0
Introduzione al ML.....	1
Supervised Learning	1
Unsupervised Learning.....	1
Semi Supervised Learning.....	2
Reinforcement Learning	2
Batch Learning.....	3
Online Learning.....	3
Instance-Based Learning.....	3
Model-Based Learning.....	3
Problems in Machine Learning.....	3
Data Profiling	5
Use Cases.....	6
Legge di Benford.....	6
Chiavi Univoco	6
Discovery Algoritm.....	8
Dipendenze Funzionali	8
HyFD	9
Inclusion Dependency	10
Dipendenze Funzionali	11
RFD Extent	11
AFD (Aproximate Functional Dependency)	12
PUD (Purity Dependency)	13
NUD (Numerical Dependency)	13
RFD Probabilità	13
CD (Constrained functional Dependency)	14
CFD (Conditional Functional Dependency).....	14
RFD Attribute Comparison	15
MFD (Metric Functional Dependency)	15
ND (Neighborhood Dependency)	16
SFD (Similarity Functional Dependency)	16
TMFD (Type-M Functional Dependency).....	16
MD (Matching Dependency)	17
CoD (Comparable Dependency)	17
DD (Differential Dependency)	17

Data Integration	19
Diversità di Prospettiva	19
Equivalenza dei Costrutti del Modello	19
Incompatibilità delle Specifiche	20
Concetti Comuni	20
Concetti Correlati	20
Architettura di Integrazione	21
Analisi dei Conflitti.....	21
Mapping.....	21
Schema Matching	22
Architettura Generale.....	22
Approcci di Schema Matching	23
Approcci basati sullo schema	24
Granularità del Matching.....	24
Approcci basati sul linguaggio	24
Approcci basati sui vincoli	25
Cardinalità del Matching	25
Approcci basati sull'istanza	25
Approcci Combinati	26
Principali Proposte in Letteratura	26
Principali Tool	26
Map Reduce.....	27
Esecuzione Map-Reduce.....	29
Quanti Map e Reduce?	30
Miglioramenti	30
Utilizzi di Map Reduce	31
Costo degli Algoritmi	31
Similar Items.....	32
Shingling	32
Jaccard Similarity	32
MinHashing.....	33
Locality-Sensitive Hashing	35
Data Mining	38
Regole di Associazione	38
Algoritmo Apriori.....	39
Regole di Classificazione e Regressione	41

Alberi di Decisione	41
Valutazione Classificatori.....	42
Clustering.....	43
K-Nearest Neighbor (KNN).....	44
Cross Validation	44
K-Means.....	44
Similarity Search	45
Classification.....	46
Cross Validation	47
Confusion Matrices.....	47
ROC Curve (Receiver Operating Characteristic)	49
Multiclass Classification.....	49
Multilabel Classification	50
Multioutput – Multiclass Classification	50
Training Models	51
Gradient Descent.....	52
Polynomial Regression	54
Learning Curves	55
Ridge Regression	56
Lasso Regression.....	57
Elastic Net.....	58
Early Stopping.....	58
Logistic Regression	59
Softmax Regression	60
SVM (Support Vector Machine).....	62
Non Linear Classification	63
SVM Regression	66
Decision Function and Predictions	66
Training Objective.....	66
Dual Problem	68
Online SVMs	68
Decision Tree	69
Decision Tree Learning	69
Split Predicates	71
Computational Complexity.....	71
Regularization Hyperparameters	71

Decision Trees Regression	71
Instability	72
Random Forest	73
Bagging and Pasting.....	73
Random Patches.....	74
Random Forest	74
Extra Trees.....	75
Boosting.....	75
Ada Boost.....	75
Gradient Boosting.....	76
Stacking.....	77
Dimensionality Reduction	78
Projection	78
Manifold Learning.....	79
PCA (Principal Component Analysis)	79
Incremental PCA.....	81
Randomized PCA	82
Kernel PCA	82
LLE (Locally Linear Embedding)	84
Other.....	85
Tensorflow.....	86
Dataflow Graph	87
TensorBoard	89
Clustering.....	90
Gerarchico	90
K-Means.....	91
BFR.....	92
CURE Alghoritm	93
ANN.....	95
Perceptron.....	95
Multioutput Classifier.....	97
Multi-Layer Perceptron	98
Backpropagation.....	99
Alternative MLP alle Step Function	99
Classification.....	100
CNN.....	101

Convolutional Layer.....	101
Pooling Layer	102
CNN Architectures	103
RNN.....	106
Encoder-Decoder.....	108



CoScienze
Associazione

Big Data

Data Science è un campo interdisciplinare su metodi, processi e sistemi scientifici per estrarre conoscenze o approfondimenti dai dati in varie forme.

Il 70% del TCO (*Total cost of ownership*) riguarda l'elaborazione dei dati, il restante 30% è suddiviso tra analisi e narrativa.

If data is too big, too fast, or too hard for existing tools to process, it is Big Data.

Le 3 "V" di Gartner riassumono le proprietà della Data Science:

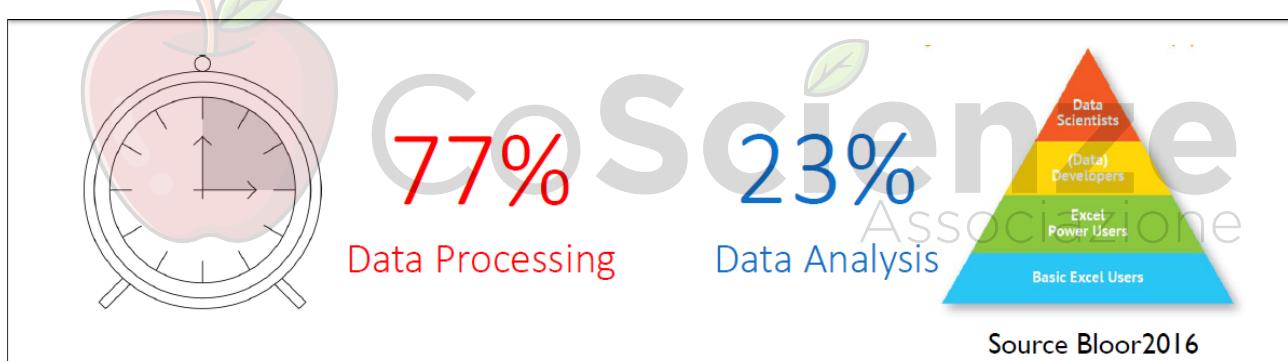
- **Volume:** esempio, 12 terabytes di Tweets per fare *sentiment analysis*;
- **Velocità:** esempio, 5 milioni di trade giornalieri usati per identificare potenziali frodi;
- **Varietà:** crescita dell'80% dei dati in immagini, video e documenti per migliorare la soddisfazione del cliente
- **(+1) Veridicità:** 1 leader aziendale su 3 non si fida delle informazioni che usano per eventuali decisioni.

Prima dell'avvento dei Big Data la **correlazione** non era intesa automaticamente come **causa** di un evento.

Oggi, invece, con i miliardi di dati che abbiamo a disposizione la correlazione è "abbastanza".

Ad esempio, se una popolazione di persone acquista un libro insieme ad un altro, Amazon ti suggerirà quel libro.

Probabilmente non sono legati causa-effetto, ma sono correlati e quindi è **utile**.



Le principali tecnologie nell'ambito dei Big Data:

- **Hadoop:** composto da quattro macro-progetti
 - *Hadoop Common* per la corretta esecuzione;
 - *HadoopDistributedFileSystem (HDFS)* un sistema di file distribuito che fornisce un rapido accesso ai dati;
 - *Hadoop MapReduce* per il processing parallelo;
 - *Hadoop YARN* per la gestione di cluster di risorse.
- **Apache Tika** per estrarre metadati dai documenti
- **Flume** per gestire massivamente i data streams
- **Sqoop** per la conversione di formati proprietari in formati open
- **Hive & Pig** per la gestione dei BigData in tabelle relazionali
- **Database NoSQL**

Introduzione al ML

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.

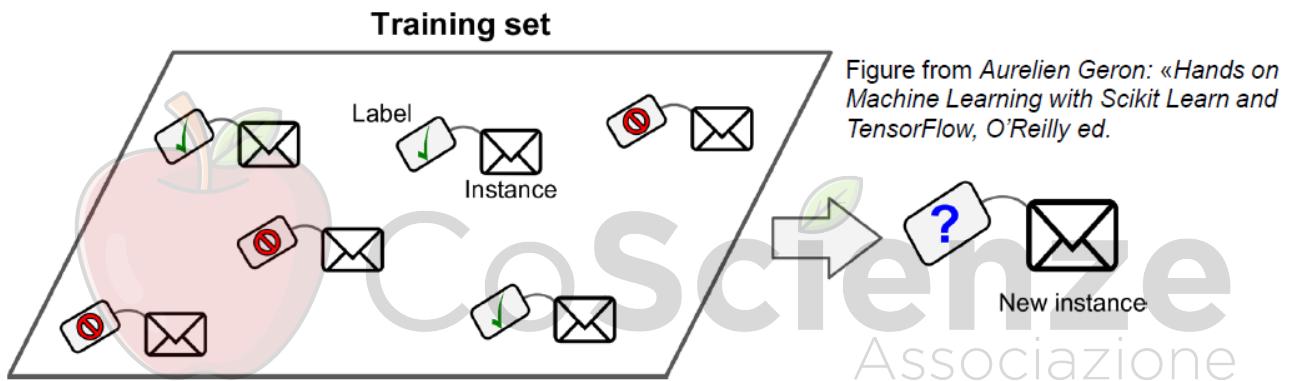
A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Facciamo un esempio con uno spam detector:

Qui, il compito T è classificare le nuove e-mail, l'esperienza E sono i dati di addestramento e la misura di prestazione P può essere ad esempio il rapporto di e-mail classificate correttamente.

Supervised Learning

Nell'apprendimento supervisionato, i dati di addestramento sono etichettati con la soluzione desiderata.



Solitamente in questa categoria rientrano task come la *classificazione* e la *regressione*.

Nella classificazione si predice una classe di output, mentre nella regressione l'output sarà un parametro numerico.

Classificazione e regressione possono anche essere combinati.

I principali algoritmi di apprendimento supervisionato:

- k Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks

Unsupervised Learning

Nell'apprendimento non supervisionato i dati di addestramento sono senza etichetta.

Un importante compito di apprendimento senza supervisione è il rilevamento delle anomalie.

L'apprendimento delle regole di associazione (*Association rule mining*) è un altro task non supervisionato, che scopre interessanti relazioni tra attributi.

I principali algoritmi di apprendimento non supervisionato:

- Clustering
 - k-Means
 - Hierarchical Cluster Analysis (HCA)
 - Expectation Maximization
- Visualization and dimensionality reduction
 - Principal Component Analysis (PCA)
 - Kernel PCA
 - Locally Linear Embedding (LLE)
 - t distributed Stochastic Neighbor Embedding (t SNE)
- Association rule learning
 - Apriori
 - Eclat

Semi Supervised Learning

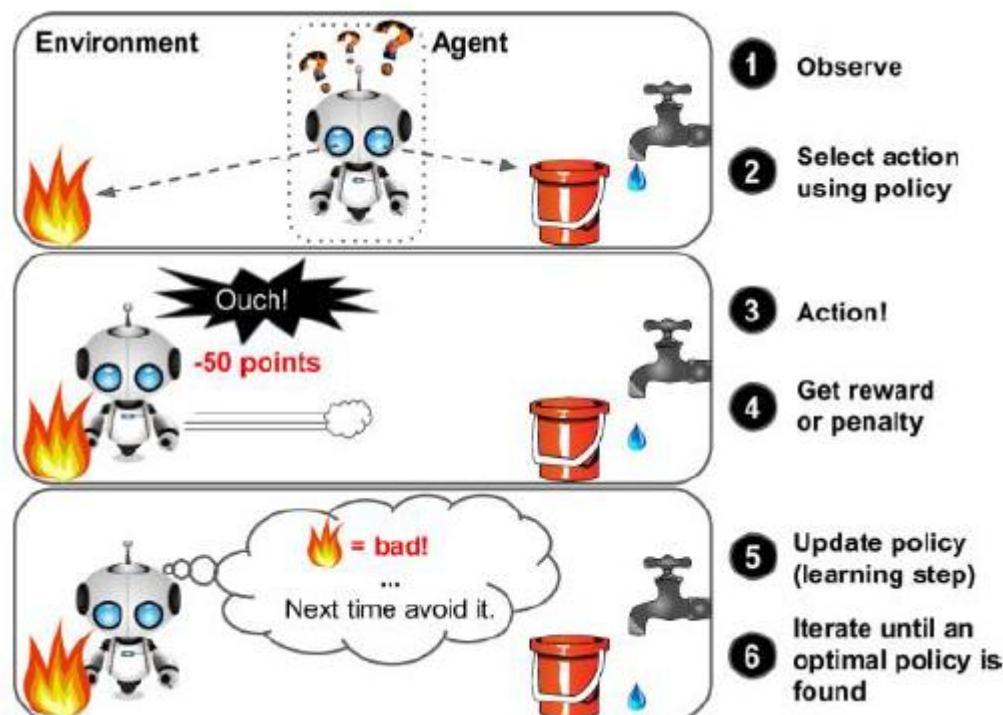
Gli algoritmi di apprendimento semisupervisionato si occupano di dati di addestramento parzialmente etichettati (di solito molti dati senza etichetta e pochi quelli etichettati).

La maggior parte degli algoritmi di apprendimento semisupervisionato sono combinazioni di algoritmi non supervisionati e supervisionati.

Reinforcement Learning

Nell'apprendimento per rinforzo un agente seleziona ed esegue azioni, ottenendo ricompense o penalità. Deve imparare la strategia migliore, chiamata **policy**, con l'obiettivo di ottenere la maggior parte della ricompensa nel tempo.

Molti robot implementano l'apprendimento per rinforzo per imparare a camminare.



Batch Learning

Il sistema deve essere addestrato utilizzando tutti i dati disponibili. Prima il sistema viene addestrato, poi applica ciò che ha appreso e funziona senza più imparare (*apprendimento offline*).

Online Learning

Il sistema viene addestrato in modo incrementale alimentandolo con istanze di dati sequenzialmente, individualmente o in piccoli gruppi. Ogni passaggio è veloce ed economico, quindi il sistema apprende i nuovi dati, man mano che arrivano.

Un parametro importante dei sistemi di apprendimento online è il **learning rate**. Con un rate elevato il sistema si adatterà rapidamente ai nuovi dati, ma dimenticherà rapidamente quelli vecchi; con un rate basso imparerà più lentamente, ma sarà meno sensibile al rumore dei nuovi dati.

Instance-Based Learning

Il sistema impara gli esempi, quindi li generalizza a nuovi casi utilizzando una misura di somiglianza.

Ad esempio, invece di contrassegnare come spam le email identiche a spam conosciuto, il filtro antispam potrebbe essere programmato a segnalare le email simili a quelle conosciute come spam. Ciò richiede una misura di somiglianza tra due e-mail (ad es. il numero di parole in comune)

Feature 2



!.

Model-Based Learning

Nell'apprendimento basato su modelli i sistemi costruiscono un modello dagli esempi a disposizione e lo usano per fare previsioni.

Una funzione di utilità (o funzione *fitness*) può essere utilizzata per misurare quanto è buono il modello. o equivalentemente, una funzione di costo può misura quanto si discosta dall'ideale.

Problems in Machine Learning

I seguenti sono tra i più rilevanti problemi nell'apprendimento automatico

- Quantità insufficiente di dati di training;
- Dati di training non rappresentativi;
- Dati di scarsa qualità;
- Caratteristiche irrilevanti;

- Overfitting dei dati di training;
- Underfitting dei dati di training;

Machine Learning significa imparare dai dati, invece di codificare esplicitamente le regole.

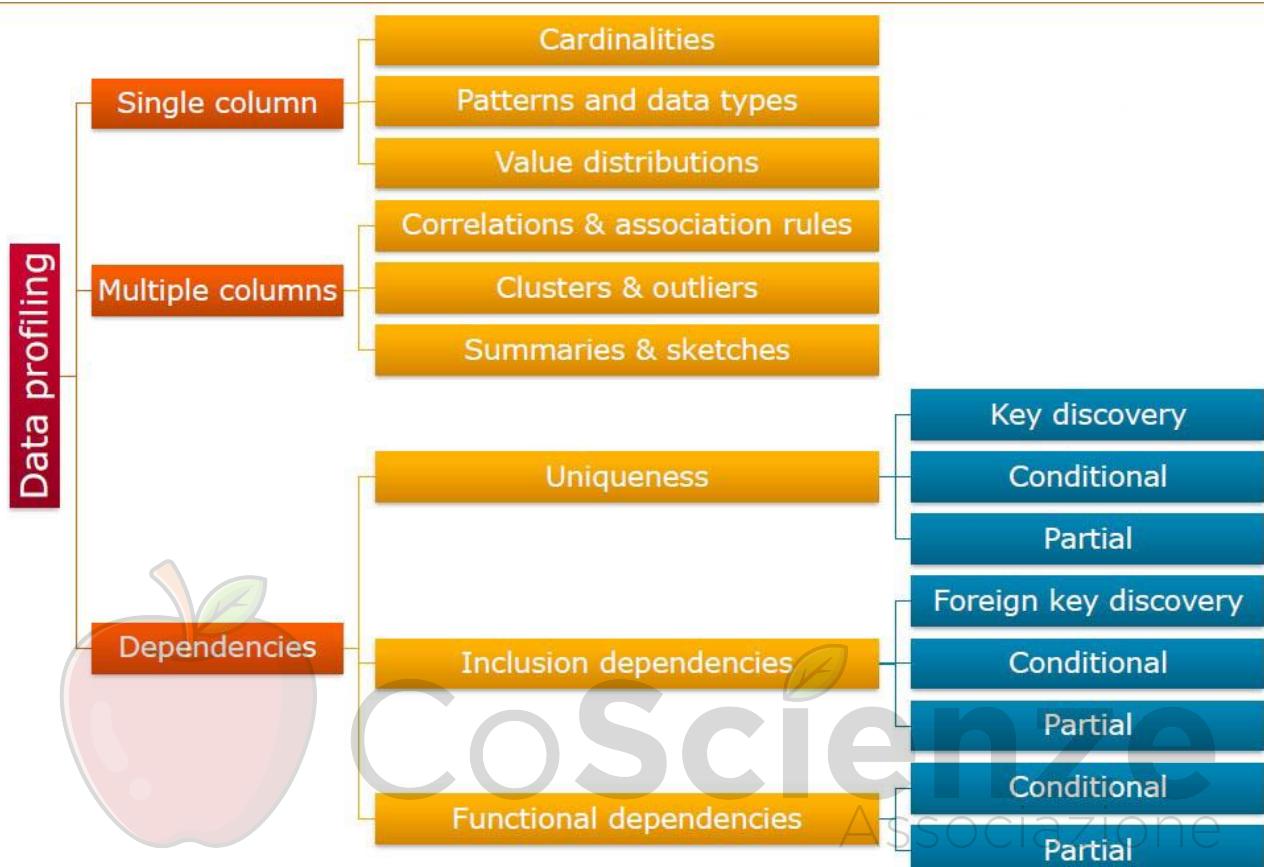
In un progetto di ML i dati vengono raccolti in un set di training e usati in un algoritmo di apprendimento. Il sistema non funzionerà bene se il training set è troppo piccolo, o se i dati non sono rappresentativi, rumorosi o inquinati da caratteristiche irrilevanti.

Infine, anche il modello non deve essere troppo semplice né troppo complesso. Una volta che un modello è stato addestrato, deve essere valutato e messo a punto se necessario.



Data Profiling

Data profiling refers to the activity of creating small but informative summaries of a database.



Category	Task	Description
Cardinalities	num-rows	Number of rows
	value length	Measurements of value lengths (min, max, median, and average)
	null values	Number or percentage of null values
	distinct	Number of distinct values; aka "cardinality"
	uniqueness	Number of distinct values divided by number of rows
Value distributions	histogram	Frequency histograms (equi-width, equi-depth, etc.)
	constancy	Frequency of most frequent value divided by number of rows
	quartiles	Three points that divide the (numeric) values into four equal groups
	soundex	Distribution of soundex codes
	first digit	Distribution of first digit in numeric values (Benford's law)
Patterns, data types, and domains	basic type	Generic data type: numeric, alphabetic, date, time
	data type	Concrete DBMS-specific data type: varchar, timestamp, etc.
	decimals	Maximum number of decimal places in numeric values
	precision	Maximum number of digits in numeric values
	patterns	Histogram of value patterns (Aa9...)
	data class	Semantic, generic data type: code, indicator, text, date/time, quantity, identifier, etc.
	domain	Classification of semantic domain: credit card, first name, city, phenotype, etc.

Use Cases

- Ottimizzazione Query: dipendenze funzionali, histogramma, counts...
- Data Cleaning: pattern, regole e vincoli
- Data Integration: dipendenze cross DB
- Gestione Dati Scientifica: analisi nuovi dataset
- Analisi dati dati: preparazione modelli
- Reverse Engineering

In sintesi **DATA PREPARATION**

Legge di Benford

Conosciuta anche come “legge del primo numero” è un’osservazione che afferma che la prima cifra di un numero in un qualsiasi caso reale è molto spesso piccola.

Infatti statisticamente è stato studiato che il numero 1 appare il 30% delle volte come cifra iniziale, mentre il 9 solo il 5% delle volte.

Chiavi Univoci

L’unicità indica il fatto che una colonna (un parametro) non abbia valori nulli e che tutti i valori siano diversi tra loro.

Questa definizione può essere anche estesa ad una combinazione di colonne.

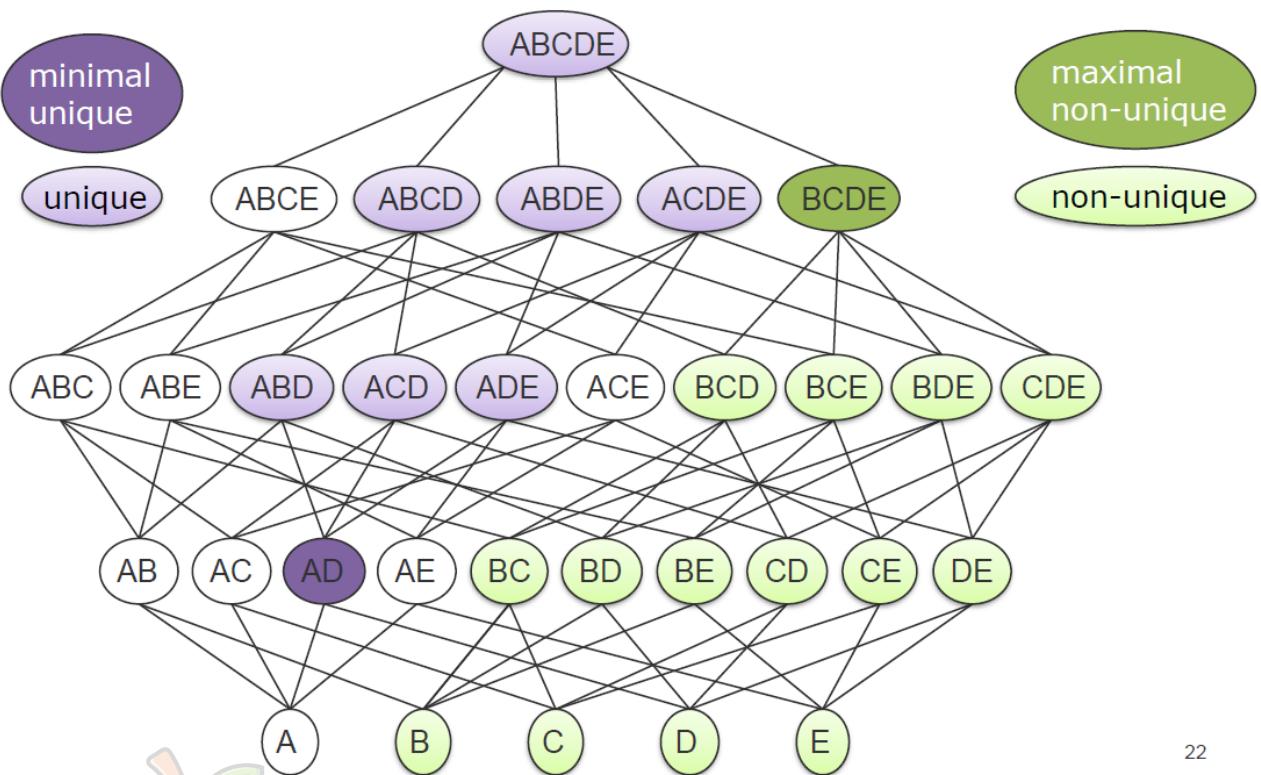
La minima unicità è il set di combinazioni di colonne **minimo** da cui è possibile generare delle chiavi, infatti aggiungendo ulteriori colonne a queste combinazioni avremo **sempre** delle chiavi univoci.

La massima non unicità è il set di combinazioni di colonne per le quali non ho unicità, ma alle quali aggiungendo **almeno** un attributo ottengo una chiave univoca.

Inoltre, qualsiasi sottoinsieme di questo set sarà NON univoco.

- Uniques: {A, AB, AC, BC, ABC}
- Minimal uniques: {A, BC}
- (Maximal) Non-uniques: {B, C}

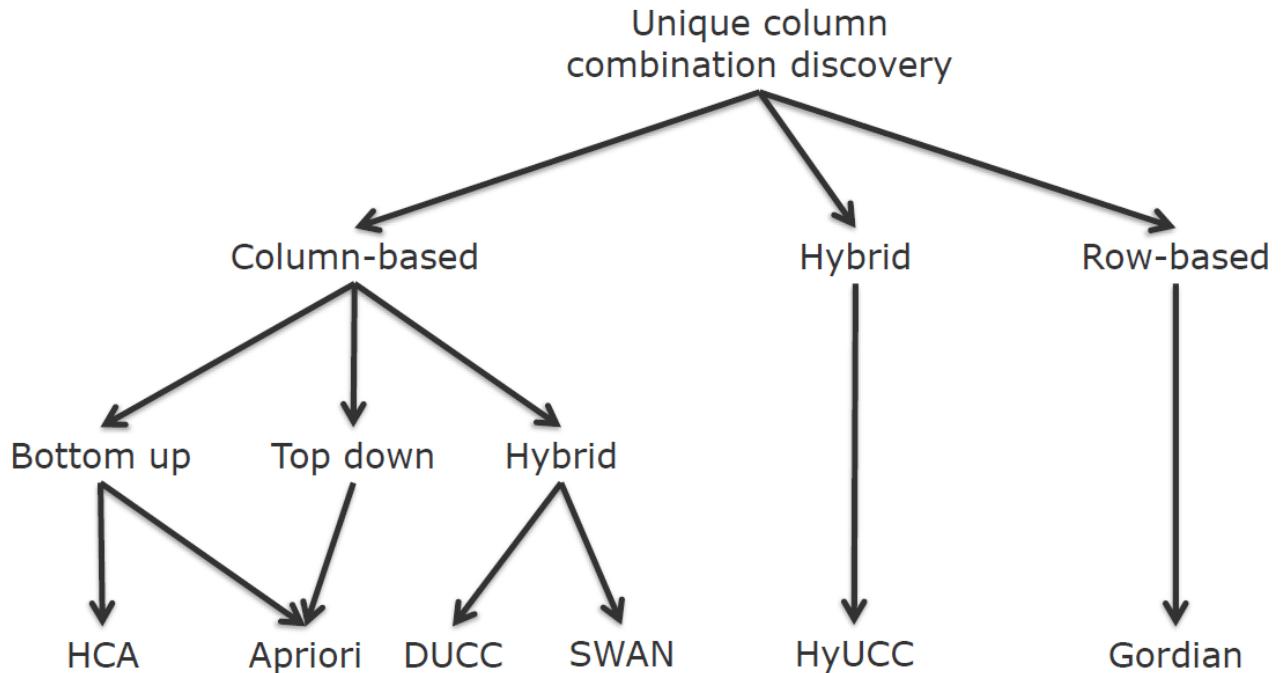
A	B	C
a	1	x
b	2	x
c	2	y



22

- Size of lattice: $2^n - 1$ (empty set not considered)
- Nodes at level 1: n
- Nodes at level n : 1
- Nodes at level k : $\binom{n}{k} = \frac{n!}{(n-k)!k!}$
- Largest level at $n/2$: $\binom{n}{n/2} = \frac{n!}{(\frac{n}{2})!^2}$

Discovery Algoritm



Dipendenze Funzionali

Una dipendenza funzionale è un particolare vincolo di integrità semantico per il modello relazionale che descrive legami di tipo funzionale tra gli attributi di una relazione.

$$\forall t_1, t_2 \in r, t_1[Y] = t_2[Y] \rightarrow t_1[Z] = t_2[Z]$$

Le dipendenze funzionali sono utili poiché vengono utilizzate per molteplici obiettivi:

- Schema Design
- Pulizia dei dati
- Ottimizzazione Query
- Normalizzazione dei dati
- Scoperta delle chiavi

Approccio Naive per la scoperta di dipendenze funzionali:

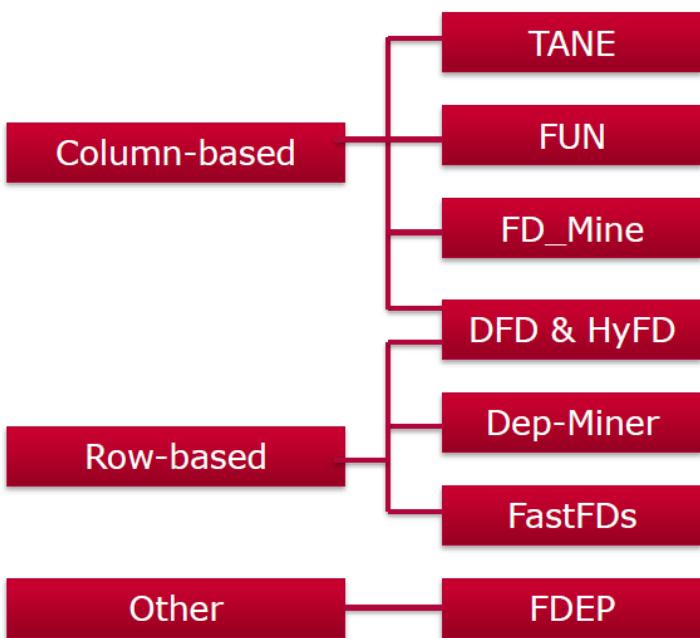
- Per ogni combinazione di colonne X
 - Per ogni coppia di tuple (t1,t2)
 - Se t1[X/A] == t2[X/A] e t1[A] ≠ t2[A]: Break

Esponenziale in (numero di attributi moltiplicato per numero di righe al quadrato).

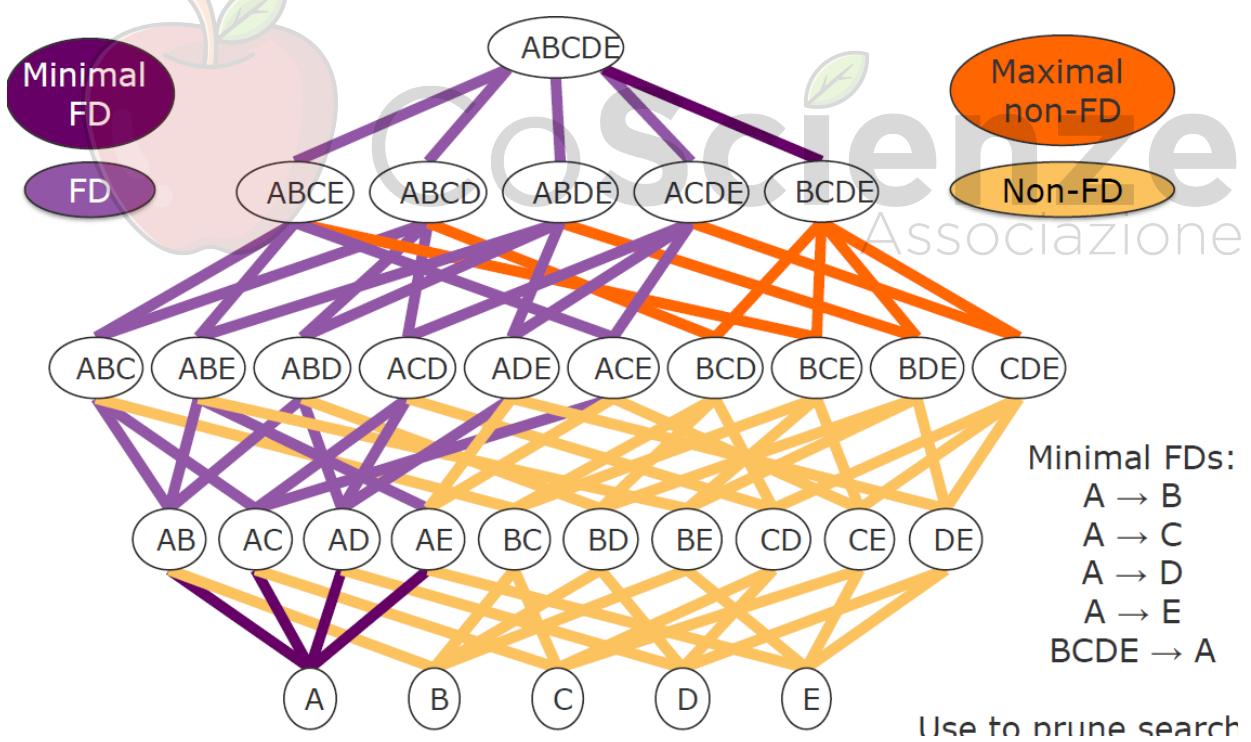
Esistono diversi algoritmi per la scoperta delle dipendenze funzionali, ma possono essere racchiusi in due categorie:

1. **Column based**, performante su dataset con poche colonne e molte righe;
2. **Row based**, performante su dataset con poche righe e molte colonne;

Un algoritmo row-based va a scansionare le righe a coppie, per trovare tutte le dipendenze funzionali NON valide, per estrarre poi quelle rimanenti come valide.



Gli archi tra i nodi all'interno del lattice rappresentano le dipendenze funzionali



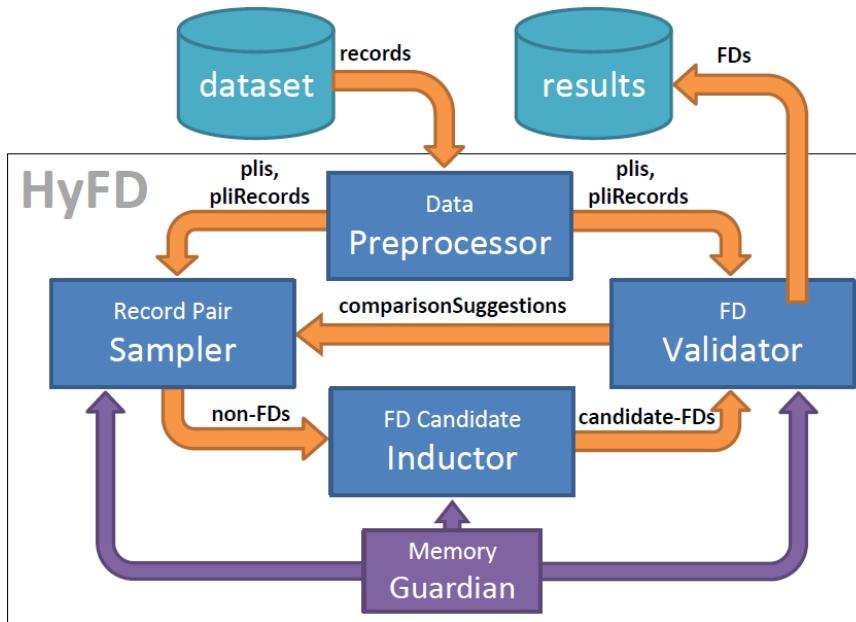
HyFD

Algoritmo ibrido: column e row based.

Il dataset viene suddiviso in *p/lis* (partizioni) di dati, ai quali viene prima applicato un algoritmo row based, che estrae delle dipendenze valide e non valide.

Le NON valide resteranno sempre non valide.

Successivamente applica un algoritmo column based a quelle valide.



Inclusion Dependency

Una dipendenza di inclusione si viene a creare tra attributi di tabelle diverse, normalmente conosciuta come **foreign key**.

Nella definizione di queste dipendenze va ricordato che la parte sinistra e la parte destra della dipendenza hanno sempre lo stesso numero di attributi.

$$R[A] \subseteq S[B] \text{ and } R[ABC] \subseteq S[DEF]$$

La complessità nella scoperta delle dipendenze di inclusione è $O(n^2)$ per le dipendenze di tipo unario, mentre è $O(2^n \times n!)$ per le dipendenze di tipo N-ennario.

Per le dipendenze di inclusione valgono le seguenti proprietà:

- Sotto sequenze di IND, sono IND.
 $R[A_1, \dots, A_n] \subseteq S[B_1, \dots, B_n] \Rightarrow R[A_{i1}, \dots, A_{im}] \subseteq S[B_{i1}, \dots, B_{im}]$
- Pruning Down
 $R[AB] \subseteq S[DE] \Rightarrow R[A] \subseteq S[D] \text{ and } R[B] \subseteq S[E]$
- Pruning Up
 $R[AB] \not\subseteq S[DE] \Rightarrow R[ABC] \not\subseteq S[DEF]$
- Pruning laterale
 $R[AB] \subseteq S[DE] \Rightarrow R[BA] \subseteq S[ED]$

Dipendenze Funzionali

Una dipendenza funzionale FD tra due insiemi di attributi X e Y , specifica un vincolo sulle tuple che possono formare uno stato di relazione r .

Il vincolo è che per ogni coppia di tuple t_1 e t_2 in r per le quali vale che $t_1[X] = t_2[X]$ allora deve valere anche $t_1[Y]=t_2[Y]$.

L'insieme di attributi di X è detto parte sinistra (*LHS Left Hand Side*) e Y è detto parte destra (*RHS Right Hand Side*).

CAP	Città	Indirizzo
84123	Salerno	Piazza Mazzini
84123	Salerno	Piazza Mazzini
84123	Salerno	via G.Vicinanza
84126	Salerno	Piazza Vicinanza

$$\{Città, Indirizzo\} \rightarrow CAP$$

L'interesse per le dipendenze funzionali è dovuto alla possibilità di poterle utilizzare in diverse operazioni avanzate sui database:

- Data cleaning
- Query relaxation
- Record matching



Sono state introdotte anche nuove definizioni di dipendenza, come la **dipendenza funzionale rilassata**. Questo perché è importante ammettere eccezioni o condizioni che restringono l'insieme delle tuple per cui deve valere una dipendenza.

Esistono due principali criteri di rilassamento:

1. **Grado di soddisfacibilità (extent)**, si vuole ammettere la possibilità che una RFD possa essere soddisfatta anche solo per un sottoinsieme di tuple di un'istanza di database.
Si usa quindi una misura di copertura oppure una condizione per restringere l'insieme delle tuple.
2. **Confronto tra attributi (attribute comparison)**, si vuole ammettere la possibilità che una RFD possa essere soddisfatta considerando anche coppie di tuple con valori simili.
Si usa quindi una misura di somiglianza.

RFD Extent

Vengono definite in base ad una **misura di copertura**

$$D_{\text{TRUE}}: X_{\text{EQ}} \xrightarrow{\Psi(X,Y)} Y_{\text{EQ}}$$

dove Ψ è definita in termini di

- Probabilità
- Cardinalità di dominio
- Impurità
- g3 error e/o confidenza

Oppure attraverso una **condizione**

$$D_c: X_{\text{EQ}} \xrightarrow{\Psi_{\text{err}(0)}} Y_{\text{EQ}}$$

dove D_c è un sottoinsieme identificato da una sequenza di predici in c che possono essere definiti in termini di

- Vincoli
- Pattern Tableau

AFD (Aproximate Functional Dependency)

Dipendenza funzionale che vale per quasi ogni tupla. Il "quasi" viene definito in base a delle misure tra le quali troviamo il **g3 error** che rappresenta il numero minimo di tuple che devono essere eliminate per far valere una dipendenza funzionale:

$$\Psi(X, Y) = \frac{\min\{|r_1| \text{ t.c. } r_1 \subseteq r \text{ e } X \rightarrow Y \text{ vale in } r \setminus r_1\}}{|r|}$$

Esempio:

SIN	Nome	Data di Nascita	Sesso	Gruppo Sanguigno
087-34-7789	Andrea White	1935-03-14	F	0+
087-11-3455	Mary Brown	1930-08-31	F	A+
089-65-3325	Bill Mc Gregor	1970-12-21	M	0+
091-87-9437	John Smith	2005-11-01	M	AB+
092-12-1439	Eric Ford	1929-10-19	M	A-
...				
097-19-7367	Mary Brown	1990-03-20	F	AB+

$$D_{\text{TRUE}}: \text{Nome}_{\text{EQ}} \xrightarrow{\Psi(X,Y) \leq 0,02} \text{GruppoSanguigno}_{\text{EQ}}$$

PUD (Purity Dependency)

Utilizza una *misura di impurità*: Date due partizioni Πs_1 e Πs_2 di un insieme S la misura di impurità avrà valore 0 se e soltanto se ogni blocco di Πs_1 è incluso in uno ed un solo blocco di Πs_2

Esempio:

Produttore	Nome	Categoria	...	Ingrediente Attivo	Prescrizione
Angelini	Aulin	NSAID		Nimesulide	Si
Dompè	Oki	NSAID		Ketoprofen	Si
Lisapharma	Arfen	NSAID		Ibuprofen	Si
...					
Zambon It	Spidifen	NSAID		Ibuprofen	No

► vale la seguente PuD

D_{TRUE}: IngredienteAttivo_{EQ}

$$\xrightarrow{\theta(\pi_{\text{IngredienteAttivo}}, \pi_{\text{Prescrizione}}) \leq 0.09} \text{Prescrizione}_{\text{EQ}}$$

NUD (Numerical Dependency)

Utilizza un vincolo sulla *cardinalità di dominio*, una NuD specifica che ogni proiezione di tupla $t[X]$ è associata al più a k differenti tuple su Y per qualche costante k .

Esempio: Se ad ogni reparto dell'ospedale possono essere assegnate al più 10 stanze

D_{TRUE}: StanzaCheckIn_{EQ}

$$\xrightarrow{\text{card}(\text{StanzaCheckIn}, \# \text{Stanza}) \leq 10} \#\text{Stanza}_{\text{EQ}}$$

RFD Probabilità

Esistono varie RFD che si basano su probabilità, ma che sfruttano teoricamente la stessa logica:

- PD (Partial Determination)
- soft FD (soft Functional Dependency)
- pFD (probabilistic Functional Dependency)

Esempio: La presenza di possibili errori nei dati sull'attributo Nome di medicine che hanno nomi simili come Daflon vs Deflan o Lanoxin vs Laroxyl fa sì che la FD canonica Nome → Produttore non valga

D_{TRUE}: Nome_{EQ} $\xrightarrow{p(\text{Nome}, \text{Produttore}) \geq 0.97}$ Produttore_{EQ}

CD (Constrained functional Dependency)

Una RFD che permette di specificare il sottoinsieme di tuple per il quale una dipendenza funzionale è valida attraverso un vincolo.

Esempio:

ID	Nome	Specializzazione	...	Esperienza	Stipendio
1	George Johnson	Neurologia		1 anno	\$118,000
2	Joe House	Cardiologia		10 anni	\$314,000
3	Derek Williams	Pediatria		2 anni	\$156,000
4	Henry Jones	Neurologia		1 anno	\$158,000
5	Robert White	Pediatria		2 anni	\$156,000
...					
30	Victor Sanchez	Radiologia		5 anni	\$225,000

$$D_c: \text{Esperienza}_{EQ} \xrightarrow{\Psi_{\text{err}(0)}} \text{Stipendio}_{EQ}$$

► dove

$$D_c = \{t \in \text{dom}(\text{Dottore}) \mid t[\text{Specializzazione}] = \text{'Pediatria'}\}$$

CFD (Conditional Functional Dependency)

Una RFD che permette di specificare il sottoinsieme di tuple per il quale una dipendenza funzionale è valida attraverso una condizione.

Il dominio di applicabilità viene definito mediante il concetto di **pattern tableau**.

Ad esempio, il seguente pattern tableau estrae tutte le tuple che come valore di *Specializzazione* = "Pediatria" e un qualsiasi valore in *Esperienza* e *Stipendio*.

Specializzazione	Esperienza	Stipendio
Pediatria	-	-

Alcune versioni di CFD:

- eCFD (extended Conditional Functional Dependency), estende le condizioni con la disunione e la disuguaglianza
- CFDp (CFD with built in predicates), permette la specifica di pattern di valori con prediciati come $<,>,\geq,\leq,$

Esempio:

ID	Nome	Specializzazione	...	Esperienza	Stipendio
1	George Johnson	Neurologia		1 anno	\$118,000
2	Joe House	Cardiologia		10 anni	\$314,000
3	Derek Williams	Pediatria		2 anni	\$156,000
4	Henry Jones	Neurologia		1 anno	\$158,000
5	Robert White	Pediatria		2 anni	\$156,000
...					
30	Victor Sanchez	Radiologia		5 anni	\$225,000

$$D_{T_r} : \text{Esperienza}_{EQ} \xrightarrow{\Psi_{err(0)}} \text{Stipendio}_{EQ}$$

► dove $T_r =$

Specializzazione	Esperienza	Stipendio
Pediatria	-	-

RFD Attribute Comparison

L'obiettivo è quello di catturare relazioni semantiche tra gruppi di valori che possono essere considerati simili piuttosto che identici.

MFD (Metric Functional Dependency)

Una RFD che generalizza le dipendenze funzionali ammettendo un confronto tra i valori delle tuple dell'RHS che tollera piccole differenze controllate da una metrica.

Esempio:

ID	Nome	Specializzazione	...	Esperienza	Stipendio
1	George Johnson	Neurologia		1 anno	\$218,000
2	Joe House	Cardiologia		10 anni	\$314,000
3	Derek Williams	Pediatria		2 anni	\$156,000
4	Henry Jones	Neurologia		1 anno	\$222,000
5	Robert White	Pediatria		2 anni	\$156,000
...					
30	Victor Sanchez	Radiologia		5 anni	\$225,000

$$D_{TRUE} : (\text{Specializzazione}, \text{Esperienza})_{EQ}$$

$$\xrightarrow{\Psi_{err(0)}} \text{Stipendio}_{\max_{s \in \pi_X} \Delta_\Phi(s[Y]) \leq 5,000}$$

ND (Neighborhood Dependency)

Essa sfrutta anche il concetto di predicato di vicinanza. Esso associa ogni attributo A ad una soglia α che permette di valutare una coppia di valori sulla base della funzione di vicinanza associata ad A.

Esempio: Si può ipotizzare che i pazienti che hanno un'età simile e diagnosi simili vengano ricoverati in reparti simili

$$D_{TRUE}: (Diagnosi, Anni)_{(\theta_{Diagnosi} \geq 0.85 \wedge \theta_{Anni} \geq 0.8)} \xrightarrow{\Psi_{err(0)}} RepartoCheckIn_{(\theta_{RepartoCheckIn} \geq 0.9)}$$

SFD (Similarity Functional Dependency)

Una RFD che sfrutta il concetto di relazione di tolleranza associata ad ogni attributo A.

Esempio:

ID	Nome	Specializzazione	...	Stipendio	Tasse
1	George Johnson	Neurologia		\$218,000	\$62,500
2	Joe House	Cardiologia		\$314,000	\$94,200
3	Derek Williams	Pediatria		\$156,000	\$39,500
4	Henry Jones	Neurologia		\$222,000	\$63,000
5	Robert White	Pediatria		\$156,000	\$39,500
...					

► vale la seguente SFD

$$D_{TRUE}: Stipendio_{\theta_{Stipendio}} \xrightarrow{\Psi_{err(0)}} Tasse_{\theta_{Tasse}}$$

TMFD (Type-M Functional Dependency)

Una RFD definita per i dati multimediali.

È parametrizzata attraverso funzioni di distanza, le quali vengono usate per confrontare gli attributi multimediali.

Siano t_1, t_2 due tuple di un'istanza di relazione r che sono simili entro una soglia ϵ_1 per l'insieme di attributi X e una soglia ϵ_2 per l'insieme di attributi Y.

Esempio: È possibile definire una dipendenza per correlare gli elettrocardiogrammi ECG con i battiti cardiaci Battiti. Tuttavia, è necessario utilizzare diverse funzioni di similarità sui singoli attributi FRATTALE e BC (Battiti cardiaci).

$$\mathbf{D}_{\text{TRUE}} : \mathbf{ECG}_{(\text{FRATTALE}, \varepsilon')} \xrightarrow{\Psi_{\text{err}(0)}} \mathbf{Battiti}_{(\text{BC}, \varepsilon'')}$$

MD (Matching Dependency)

Una RFD che mira a risolvere il problema di identificare quali record di istanze di database differenti rappresenta la stessa entità nel mondo reale.

Esempio:

$$\mathbf{D}_{\text{Paziente}} \times \mathbf{D}_{\text{Cliente}} :$$

$$(\{\text{Nome}, \text{DataDiNascita}\}, \{\text{Cliente}, \text{DataNascita}\})_{\approx} \xrightarrow{\Psi_{\text{err}(0)}} (\text{Indirizzo}, \text{Domicilio})_{\equiv}$$

CoD (Comparable Dependency)

Una RFD che generalizza il concetto di dipendenza funzionale nel contesto dei dataspace eterogenei.

Permette di gestire il confronto tra attributi con nomi differenti attraverso un operatore di confronto degli attributi.

Esempio: In un database clinico che contiene le cartelle cliniche di diversi ospedali vale la seguente CoD

$$\mathbf{D}_{\text{TRUE}} : (\#\text{Stanza}, \#\text{Letto})_{\theta(\#\text{Stanza}, \#\text{Letto})} \xrightarrow{\Psi_{\text{err}(0)}} (\text{Sesso}, \text{Genere})_{\theta(\text{Sesso}, \text{Genere})}$$

Dato che ogni stanza di un ospedale identifica univocamente il sesso del paziente che la occupa.

DD (Differential Dependency)

Una RFD che permette di esprimere vincoli di differenza sui valori degli attributi attraverso una funzione di distanza.

Esempio:

$$\mathbf{D_{TRUE}: DataDiPrescrizione}_{\phi_L} \xrightarrow{\Psi_{err(0)}} \mathbf{DataDiEsecuzione}_{\phi_R}$$

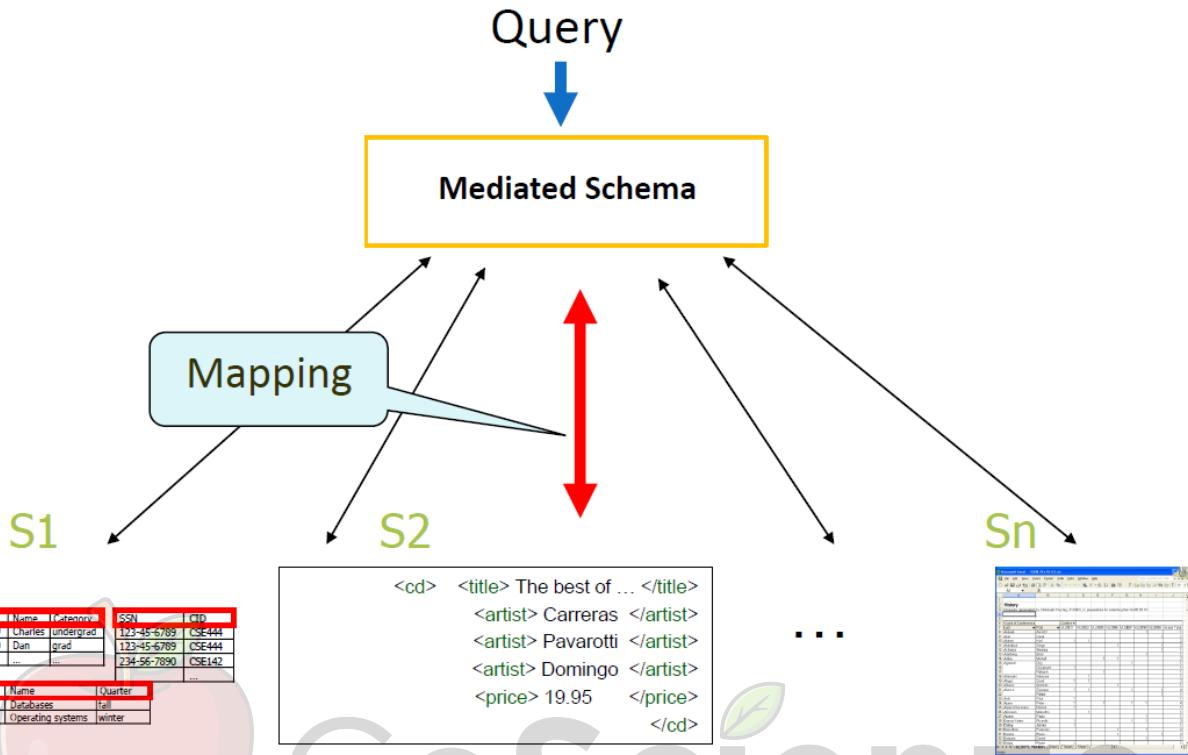
dove

- ▶ $\phi_L[\text{DataDiPrescrizione}] = [\text{DataDiPrescrizione}(=0)]$
- ▶ $\phi_R[\text{DataDiEsecuzione}] = [\text{DataDiEsecuzione } (\leq 5)]$



Data Integration

La data integration rappresenta il processo di integrazione di dati provenienti da sorgenti informative multiple, distribuite, autonome ed eterogenee, con lo scopo di fornire agli utenti l'accesso uniforme agli stessi.

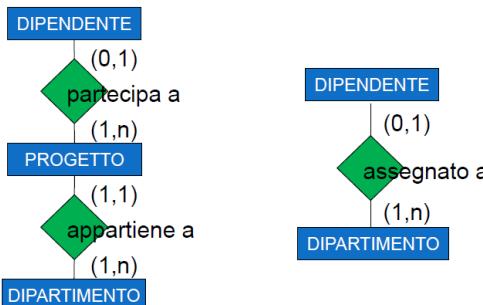


Nel processo di integrazione diventa necessario gestire problematiche su diversi livelli di astrazione:

- *Livello di sistema*: Gestire diverse piattaforme;
- *Livello logico*: Eterogeneità degli schemi (e dei dati);
- *Livello sociale*: localizzare e catturare solo i dati rilevanti nelle aziende

Diversità di Prospettiva

Uno stesso oggetto può essere tradotto differentemente in sistemi diversi



Equivalenza dei Costrutti del Modello

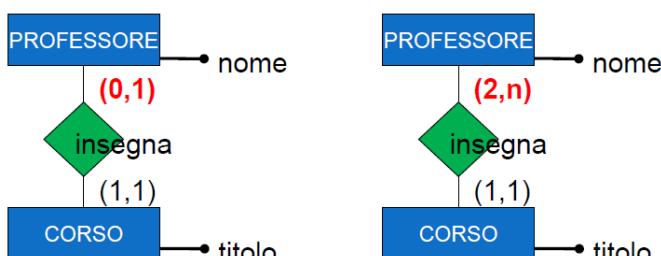
Possono essere usati diversi costrutti per descrivere lo stesso oggetto



Incompatibilità delle Specifiche

L'incompatibilità delle specifiche indica che schemi diversi che modellano una stessa porzione del dominio applicativo racchiudono concetti in contrasto tra loro.

Esempio: in un caso un professore non può tenere più di un corso, nell'altro deve tenerne almeno 2



Concetti Comuni

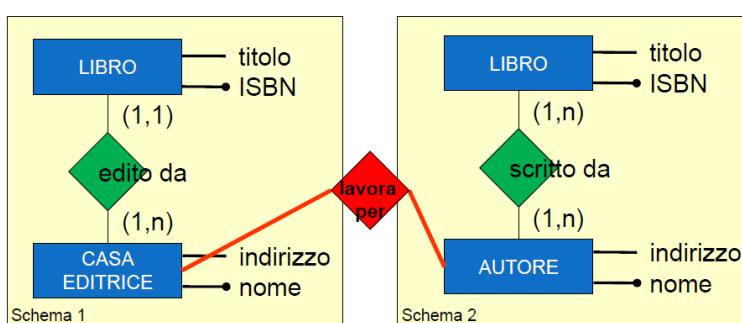
Sono possibili quattro relazioni tra due rappresentazioni R1 e R2:

- *Identità:* si verificano quando vengono utilizzati gli stessi costrutti e quindi le rappresentazioni coincidono;
- *Equivalenza:* sono stati utilizzati costrutti differenti ma equivalenti ai fini della rappresentazione;
- *Comparabilità:* le rappresentazioni non sono né identiche né equivalenti, ma comunque non sono in contrasto tra loro;
- *Incompatibilità:* quando la realtà modellata da R1 nega la realtà modellata da R2

Concetti Correlati

A seguito dell'integrazione, molti concetti diversi, ma correlati, verranno a trovarsi nello stesso schema, dando vita a nuove relazioni che non erano percepibili in precedenza.

Tali relazioni sono dette **proprietà inter-schema**.



Architettura di Integrazione

Esistono due approcci:

- **Database materializzato (Data Warehouse):** i dati vengono trasposti in un nuovo ambiente;
- **Database virtuale (Viste):** i dati restano nelle sorgenti e vengono uniti durante la lettura.

Analisi dei Conflitti

Inizialmente vengono analizzati i concetti correlati (*Schema Matching*), dei quali poi viene fatta un'analisi dei conflitti per integrare i sistemi a livello concettuale.

I conflitti possono essere:

- *Conflitti di eterogeneità:* indicano discrepanze dovute all'utilizzo di formalismi con diverso potere espressivo negli schemi sorgenti;
- *Conflitti semanticci:* si verificano quando due schemi sorgenti modellano la stessa porzione di mondo reale a un diverso livello di astrazione e dettaglio;
- *Conflitti sui nomi:* si verificano a causa delle differenze nelle terminologie utilizzate nei diversi schemi sorgenti;
- *Conflitti strutturali:* conflitti dovuti all'utilizzo di tipi differenti, di relazioni di dipendenza differente, di chiavi differenti e di politiche di cancellazione/modifica differenti.

Mapping

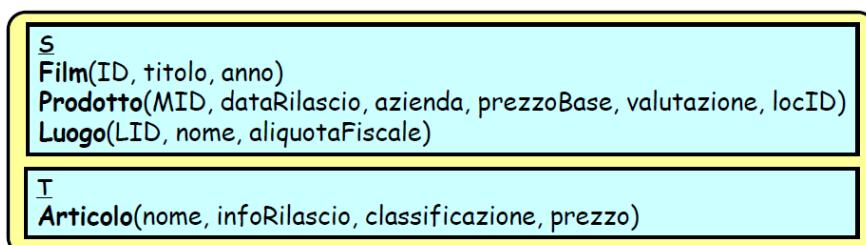
È necessario definire un *mapping* tra lo schema globale costruito a partire dalle sorgenti, e quello delle sorgenti stesse.

Esistono due approcci:

- **GAV (Global as View):** Ad ogni concetto dello schema globale deve essere associata una vista il cui significato è definito in base ai concetti che risiedono sugli schemi sorgenti.
Riduce l'estensibilità dello schema riconciliato poiché l'aggiunta di una nuova sorgente richiederà la modifica di tutti i concetti dello schema globale che la utilizzano.
Facilita la modalità di definizione delle interrogazioni poiché per capire quali concetti degli schemi sorgenti sono coinvolti sarà sufficiente sostituire a ogni concetto dello schema globale la definizione della vista che lo definisce rispetto ai concetti sugli schemi locali(**unfolding**).
- **LAV (Local as View):** Lo schema globale è espresso indipendentemente dalle sorgenti, i cui concetti saranno invece definiti come viste sullo schema globale.
Richiede trasformazioni complesse *query rewriting* per capire quali elementi degli schemi sorgente devono essere presi in considerazione per ricreare il concetto espresso nello schema globale.
Favorisce l'estensibilità dello schema riconciliato e la sua manutenzione.

Schema Matching

Un mapping semantico è una query che correla uno schema S con uno schema T



Il seguente mapping mostra come ottenere *Film.titolo*:

```
SELECT NOME AS TITOLO
FROM ARTICOLO;
```

I match semanticici possono essere di vario tipo:

- Uno a Uno
- Uno a Molti
- Molti a Uno
- Molti a Molti

Ritornando all'esempio precedente, ad esempio, abbiamo come corrispondenze Uno a Uno:

- Film.Titolo = Articolo.nome
- Prodotto.valutazione = Articolo.classificazione

Corrispondenze Uno a Molti:

- Articolo.prezzo = Prodotto.prezzoBase*(1+aliquotaFiscale)

Per creare la descrizione di una sorgente, spesso si creano prima i **match semanticici** e successivamente si trasformano i match in **mapping**.

Questo perché i sistemi di matching e mapping devono riconciliare l'eterogeneità semantica degli schemi.

L'eterogeneità semantica può avvenire in diversi modi:

- Stesso concetto con nomi differenti;
- Più attributi in uno schema correlati con un solo attributo nell'altro;
- Organizzazione tabellare differente;
- Differenze nel livello di dettaglio e copertura

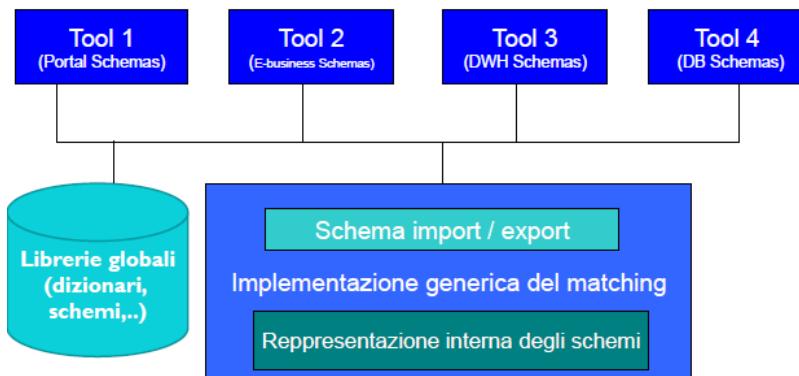
Solitamente la causa di queste differenze è dovuta al fatto che gli schemi vengono fatti da persone differenti e che l'obiettivo della rappresentazione è diverso pur coinvolgendo lo stesso oggetto.

Lo **schema matching** è il processo che permette di trovare le corrispondenze semantiche tra gli elementi di due schemi

Architettura Generale

Un'architettura generale per l'automatizzazione del processo di schema matching dovrà prevedere:

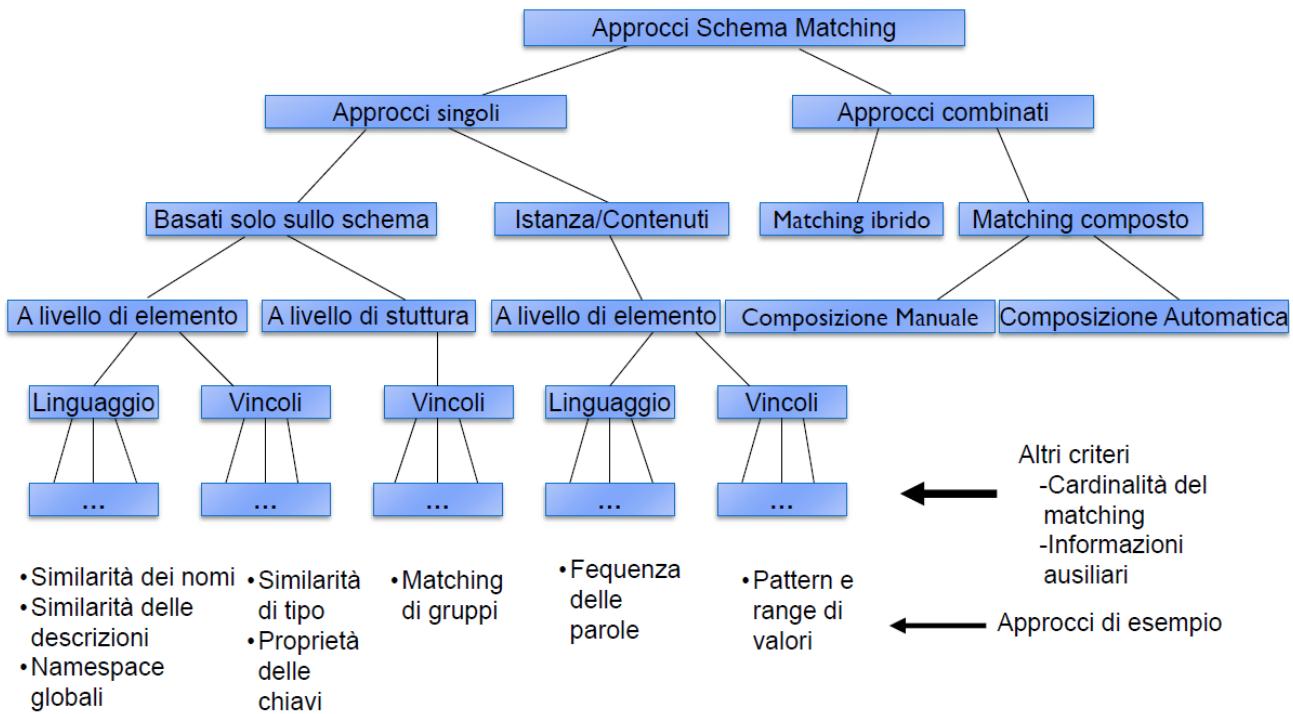
- Gli schemi che devono essere analizzati per il processo di matching devono avere necessariamente una rappresentazione interna uniforme
- Deve essere definita una componente che effettui l'import degli schemi, traducendoli dalla loro rappresentazione nativa alla rappresentazione interna
- Deve essere definita una componente che effettui l'export dei risultati di matching dalla rappresentazione interna alla rappresentazione richiesta da ogni tool



Approcci di Schema Matching

Esistono diversi approcci per lo schema matching:

- *Basato sull'istanza vs Basato sullo schema*: possono essere considerati i dati piuttosto che lo schema;
- *Matching di elementi vs Matching di strutture*: possono essere considerati i singoli elementi piuttosto che una combinazione di essi;
- *Basato sul linguaggio vs Basato sui vincoli*: possono essere considerati i nomi piuttosto che i vincoli dello schema;
- *Cardinalità del matching*: un matching può correlare uno o più elementi a uno o più schemi;
- *Informazioni ausiliari*: possono essere sfruttati dizionari, input utente..



Approcci basati sullo schema

Considerano informazioni come:

- Nome
- Descrizione
- Tipi di dati
- Relazioni (is a, part of)
- Vincoli
- Strutture



Granularità del Matching

Determina per ogni elemento del primo schema, quello che corrispondono nell'altro.

A livello di struttura può esserci un matching **completo**, se tutti gli elementi sono mappati tra gli schemi, o un matching **parziale**.

Approcci basati sul linguaggio

Considerano informazioni come:

- Nomi degli attributi
- Descrizione degli attributi

A supporto di ciò, per effettuare un matching, vengono utilizzati dizionari.

Esempio

S1	S2	Matching basato su
CarID	TruckID	Car \cong Truck (Iperonimi) Car è un automobile e truck is un automobile.
Brand	Make	Brand \cong Make (Sinonimi)
Price	Price	Price \cong Price (Uguaglianza dei nomi)
SoldTO	Sold2	SoldTo \cong Sold2 (Sottostringhe)
CAddress	CustomerAddress	Caddress \cong CustomerAddress (Pre-processing)

Approcci basati sui vincoli

Considerano informazioni come:

- Vincoli sui tipi di dati
- Vincoli di dominio
- Caratteristiche delle chiavi
- Cardinalità delle relazioni
- Chiavi esterne

Esempio

S1	S2	Matching
Employee	Personal	
EmpNo {int, PK}	Pno {int, PK}	Born \cong Birthdate {Tipo} Pno \cong EmpNo DeptNo {Chiave}
EmpName {String}	Pname {string}	Pname \cong DeptName {Tipo} Pname \cong EmpName {Tipo} Dept \cong DeptName {Tipo} Dept \cong EmpName {Tipo}
DeptNo {int, ref dep}	Dept {String}	S2 Personal {Pno, Pname, Dept, born} Select S1.Employee.EmpNo, S1.Employee.EmpName, S1.Department.DeptName, S1.Employee.BirthDate
salary {single}	Born {date}	From S1.Employee, S1.Department Where (S1.Employee.DeptNo = S1.Department.DeptNo)
BirthDate {date}		
Department		
DeptNo {int,PK}		
DeptName {String}		Nota: Matching strutturale

Potrebbero esserci diverse corrispondenze candidate per cui questo approccio potrebbe essere utilizzato per limitare il numero di candidati.

Cardinalità del Matching

Uno o più elementi dello schema S1 potrebbero corrispondere ad uno o più elementi dello schema S2

Esempio

	Cardinalità del matching	S1	S2	Matching
1	1:1 a livello di elemento	Price	Amount	Amount = Price
2	n:1 a livello di elemento	Price,Tax	Cost	Cost = Price * (1 + Tax/100)
3	1:n a livello di elemento	Name	FirstName LastName	FirstName, LastName = Extract(Name,...)
4	n:1 a livello di struttura (n:m a livello di elemento)	B.Title B.PuNo P.PuNo P.Name	A.Book A.Publisher	A.Book, A.Publisher = Select B.Title, P.Name From B,P Where B.PuNo = P.PuNo

Approcci basati sull'istanza

Considerano informazioni come:

- Nomi degli elementi
- Range di valori

- Tipi di dato
- Sottoelementi
- Regole

EmpNo	Dept
234	Marketing
235	Accounting
236	Marketing

SSN	Works for
230	Accounting
229	Marketing
228	Marketing

{Dept ≈ works for} (sulla base della frequenza di "Marketing")

{EmpNo ≈ SSN} (sulla base del range di valori)

Approcci Combinati

Un sistema ideale deve sfruttare più tipologie di matching.

Le due categorie principali sono:

- **Matching ibrido:** combina diversi approcci (nomi, tipi, vincoli..) e risulta più efficace perché i candidati mediocri vengono presto eliminati
- **Matching composto:** combina i risultati di diversi approcci eseguiti in maniera indipendente e risulta più flessibile perché permette di selezionare tra più *matcher*

Principali Proposte in Letteratura

- **LSD (Learning Source Descriptions):** dell'Università di Washington, sfrutta tecniche di machine learning per mappare una nuova sorgente rispetto ad uno schema definito
- **SKAT (Semantic Knowledge Articulation Tool):** dell'Università di Stanford, si basa su regole per determinare in maniera semi-automatica le corrispondenze tra schemi
- **TransScm:** dell'Università di Tel Aviv, utilizza le corrispondenze tra schemi per derivare una traduzione automatica dei dati tra le istanze degli stessi
- **DIKE (Database Intensional Knowledge Extractor):** dell'Università di Calabria, supporta la costruzione semi-automatica di sistemi informativi cooperativi (CIS) partendo da DB eterogenei
- **Cupid:** di Microsoft, è un matcher ibrido basato sulla ricerca delle corrispondenze sia a livello di elemento che a livello di struttura

Principali Tool

Tool semi-automatici che assistono l'utente nella definizione dei match e dei mapping tra due schemi

- Clio (IBM Almaden e Università di Toronto):
- COMA++ (Università di Lipsia):
- Harmony (MITRE Corporation)
- CoDIT (Università di Salerno)

Map Reduce

Un'implementazione MapReduce permette di gestire molti calcoli su larga scala in modo tollerante ai guasti hardware.

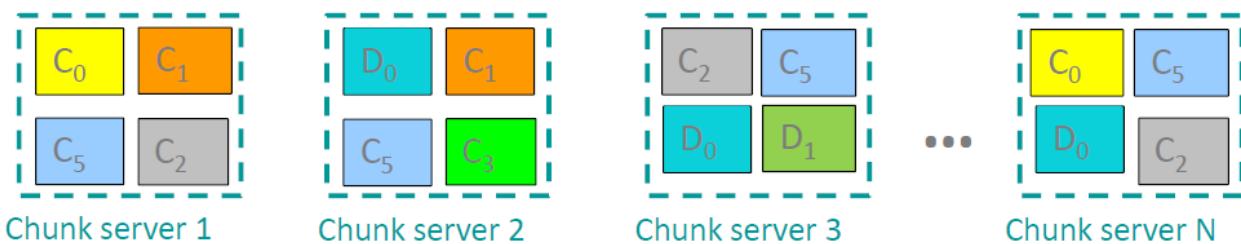
I dati devono però essere aggiornati raramente, mentre possono essere comuni eventuali *read* e *append*.

Per l'utilizzo di questo paradigma è necessaria un file system distribuito.

I dati vengono suddivisi in “*chunks*” che vengono replicati all'interno di macchine distribuite.

Deve essere presente un nodo *master* nel quale vengono storicizzati i metadati che contengono informazioni sulla locazione delle risorse.

I *client*, invece, usano una libreria che consente di conoscere la locazione dei *chunk servers* (mediante il nodo master) e di connettersi a loro.

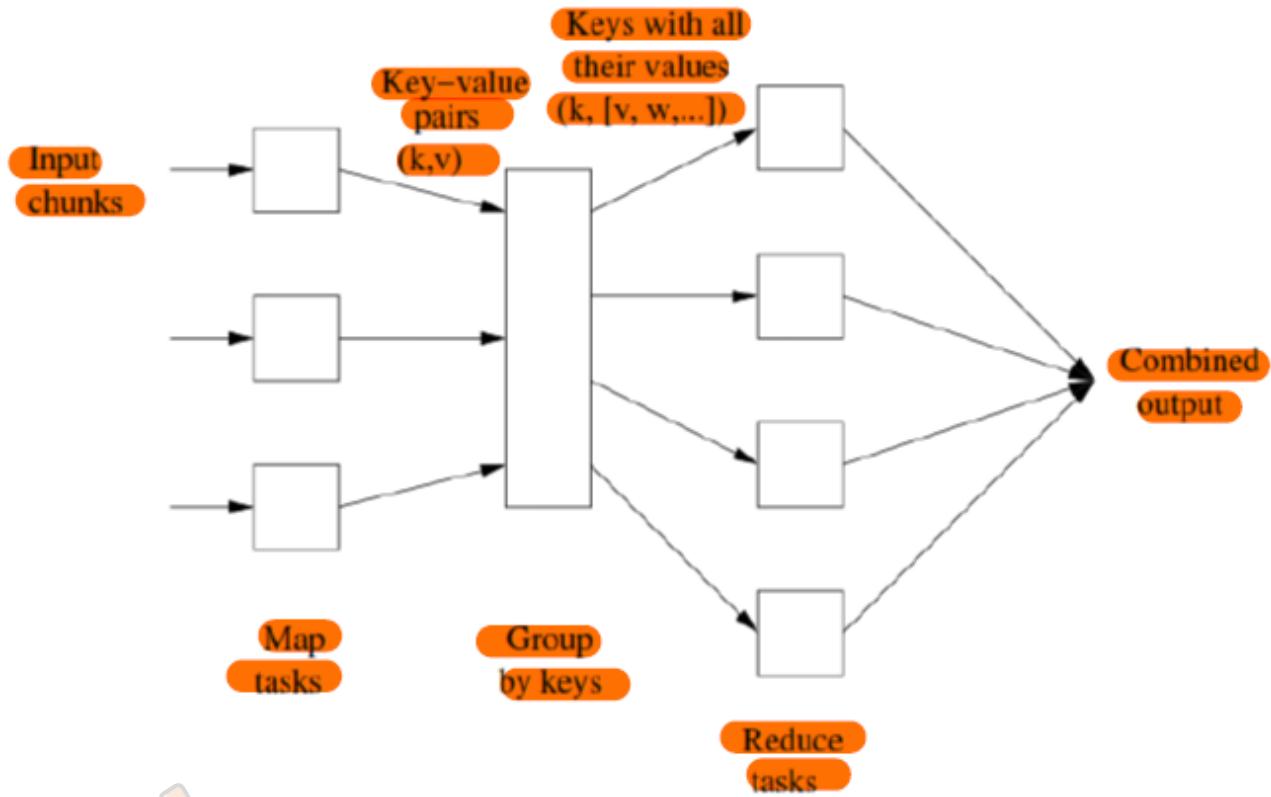


A questo punto tutto quello che serve scrivere sono due funzioni: ***Map*** e ***Reduce***.

Mentre il sistema gestisce l'esecuzione parallela, coordinando le attività che eseguono i task di *Map* e *Reduce*, affronta anche la possibilità che alcuni task falliscano.

Le fasi di un sistema di questo tipo sono principalmente tre:

1. Ad ogni task ***Map*** vengono dati in input uno o più *chunks* (parte di dati), eseguendo il codice progettato nel task questi vengono trasformati in una sequenza di *chiave-valore*;
2. Le sequenze *chiave-valore* di ogni task *Map* vengono collezionate da un ***master controller***, ordinate per chiave e suddivise tra i task ***Reduce***, in modo che tutte le sequenze con la stessa chiave arrivino allo stesso task *Reduce*.
Questo è possibile poiché il controller conosce quanti *Reduce Task* sono disponibili, ad esempio “r”, e quindi esegue una funzione di hash sulle chiavi che produce un risultato da 0 a r-1.
3. Il task *Reduce* elabora i valori arrivati con la stessa chiave eseguendo il codice progettato nella ***Reduce function***. L'applicazione di questa funzione è detta ***reducer***.



Un esempio può essere un programma che debba contare le occorrenze delle parole in un file di testo. E' possibile eseguire questo compito con il paradigma Map-Reduce, utilizzando queste due funzioni:

map(key, value):

```
// key: document name; value: text of the document
```

for each word w in value:

```
emit(w, 1)
```

reduce(key, values):

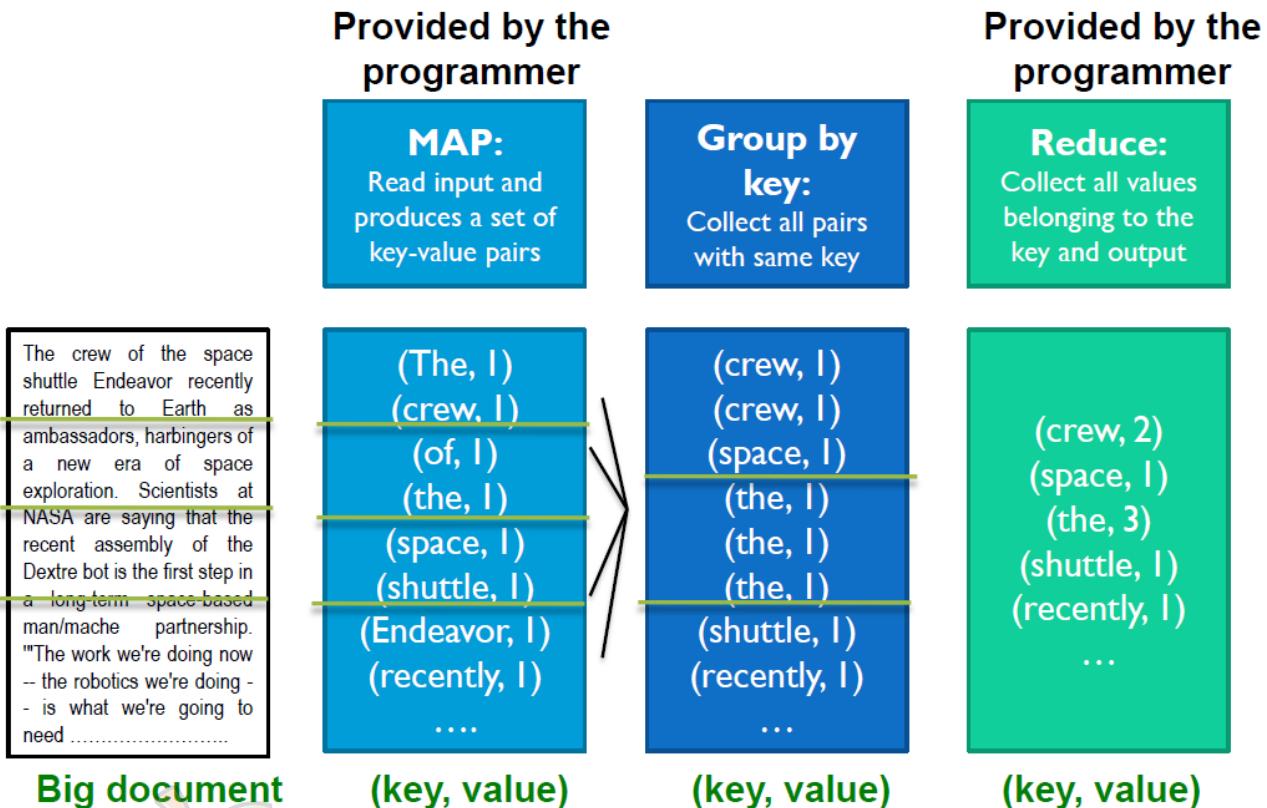
```
// key: a word; value: an iterator over counts
```

$result = 0$

for each count v in values:

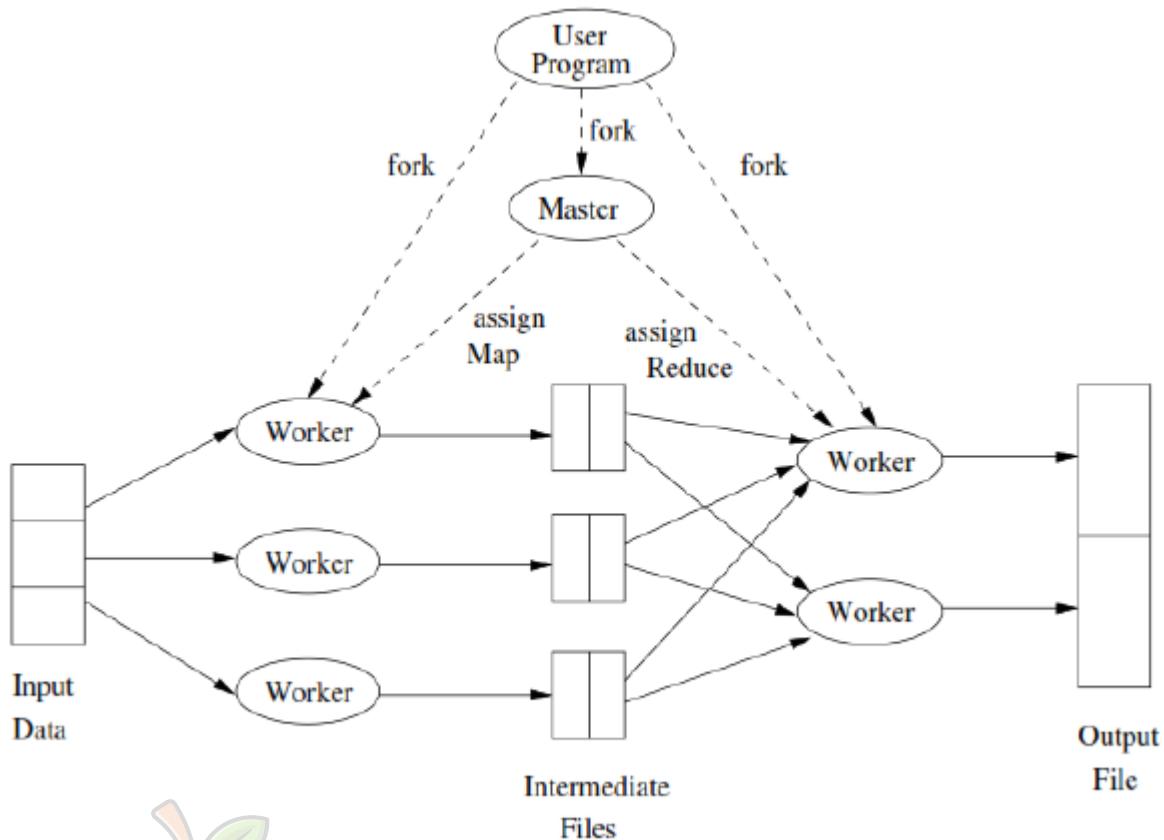
```
result += v
```

emit(key, result)



Esecuzione Map-Reduce

- Il programma utente esegue il *fork* di un processo *Master Controller* e di alcuni *Worker* in nodi differenti
- Un *Worker* gestisce un *Map Task* oppure un *Reduce Task*, ma non entrambi
- Il *Master* crea alcuni *Map Task* e *Reduce Task* e li assegna ai processi *Worker*
- È ragionevole pensare di creare un *Map Task* per ogni *chunk*, mentre un numero di *Reduce Task* minore
- Il *Master* continua a monitorare lo stato di ogni *Map* e *Reduce Task* (attesa, esecuzione su un *Worker*, completato)
- Un *Worker* notifica il *Master* quando termina e può essere quindi schedulato di nuovo
- Un *Map Task* crea un file per ogni *Reduce Task* sul disco locale del *Worker* e informa il *Master* sulla locazione e la grandezza di questi, oltre al *Reduce Task* al quale è destinato
- Il *Master* periodicamente pinga i *Worker* per verificarne il funzionamento o l'eventuale fallimento
- Se il *Master* rileva un fallimento di un nodo in cui risiede un *Map Worker*, tutti i task di quel worker devono essere rifatti anche se già completati. Questo perché l'output è all'interno del nodo fallito e quindi non più disponibile per il *Reduce Task*.
Il Master, quindi, setta lo stato di questi task "in attesa" e li schedulerà nuovamente su un altro *Worker*.
Il Master inoltre informerà il *Reduce Task* della nuova locazione del file di output del *Map Task* cambiato
- Se fallisce un *Reduce Task* il *Master* setta lo stato "in attesa" e lo rischedula nuovamente su un altro *Worker* successivamente.



Quanti Map e Reduce?

Consideriamo M Map Task e R Reduce Task.

La regola empirica afferma che M deve essere più grande del numero di nodi nel cluster, mentre R sarà più piccolo di M. Questo comporta un miglioramento nel *load balancing* e nel recupero da possibili fallimenti dei workers.

Inoltre, c'è un overhead per ogni task che viene creato, quindi conviene mantenere il numero di *Reduce Task* più piccolo delle possibili chiavi dei *Map Task*.

Dato che il numero delle liste di elementi per ogni chiave può essere molto differente, passando da una lista di pochi elementi ad una lista molto lunga, questo può comportare una grossa differenza nel tempo di esecuzione per ogni *Reduce Task*.

Questa differenza si può ridurre usando meno *Reduce Task* del numero di *reducer* (*una chiave con la lista di valori associata*), ma mantenendo il numero maggiore del numero di nodi del cluster, così che un nodo possa gestire un task lungo e un altro molteplici task brevi.

Miglioramenti

- **Backup Task:** Workers lenti possono comportare un aumento nel completamento di un job, quindi una soluzione può essere spawnare copie di quel task, così che il primo che finisce vince.
- **Combiners:** un Map Task solitamente produce coppie di chiave-valori con chiavi diverse, ma anche con la stessa chiave, quindi per salvare un tempo di comunicazione in rete, si possono utilizzare dei costrutti (*combiners*) che combinano le stesse chiavi prima di mandare l'output
- **Funzione di partizione:** si vuole controllare quante chiavi vengono partizionate, e solitamente si usa la funzione $hash(k)modR$ per questo. A volte è meglio effettuare l'override di questa, ed usare $hash(hostname(URL))modR$

Utilizzi di Map Reduce

Il paradigma MapReduce NON è una soluzione ad ogni problema.

Infatti ha senso nel momento in cui stiamo usando una grossa mole di dati che raramente viene aggiornata.

L'obiettivo principale per cui Google ha implementato MapReduce era per eseguire una grossa elaborazione matrice-vettore per il calcolo di *PageRank*.

Il prodotto matrice-vettore è un vettore di lunghezza "n", dove l'iesimo elemento è dato da:

$$x_i = \sum_{j=1}^n m_{ij} v_j$$

Quindi i compiti erano così suddivisi tra i task:

- **Map Function:** elaborava un chunk di M (matrix). Produceva le chiavi-valori (i, mij, vj), dove "i" rappresenta l'indice iesimo dell'elemento del vettore risultante, e gli altri due l'elemento della matrice e il vettore di partenza.
- **Reduce Function:** effettua la somma di tutti i valori associati alla stessa chiave "i". Il risultato sarà la coppia (i,xi)

Costo degli Algoritmi

Abbiamo diversi tipi di costi:

- **Communication Cost:** numero totale di operazioni I/O di tutti i processi
- **Elapsed Communication Cost:** il massimo di I/O lungo ogni percorso
- **Elapsed Computation Cost:** il massimo tempo di elaborazione dei processi

Per l'algoritmo Map-Reduce abbiamo:

- **Communication Cost** = input size + 2*(somma di tutti i size dei file passati da Map a Reduce) + somma dei file di output di Reduce (ogni file è contato due volte, una prima volta per lo *shuffling* e una seconda per il reduce task)
- **Elapsed Communcation Cost** = somma dell'input più grande + output per ogni processo Map + la somma del file più grande per ogni processo Reduce
- **Elapsed Computation Cost** = si tende ad ignorarlo, dato che le funzioni sono semplici e quindi i costi dominanti sono i precedenti.

Il costo totale di comunicazione del Join del Map-Reduce invece è $O(|R|+|S|+|R \bowtie S|)$, mentre l'elapsed è $O(s)$

Similar Items

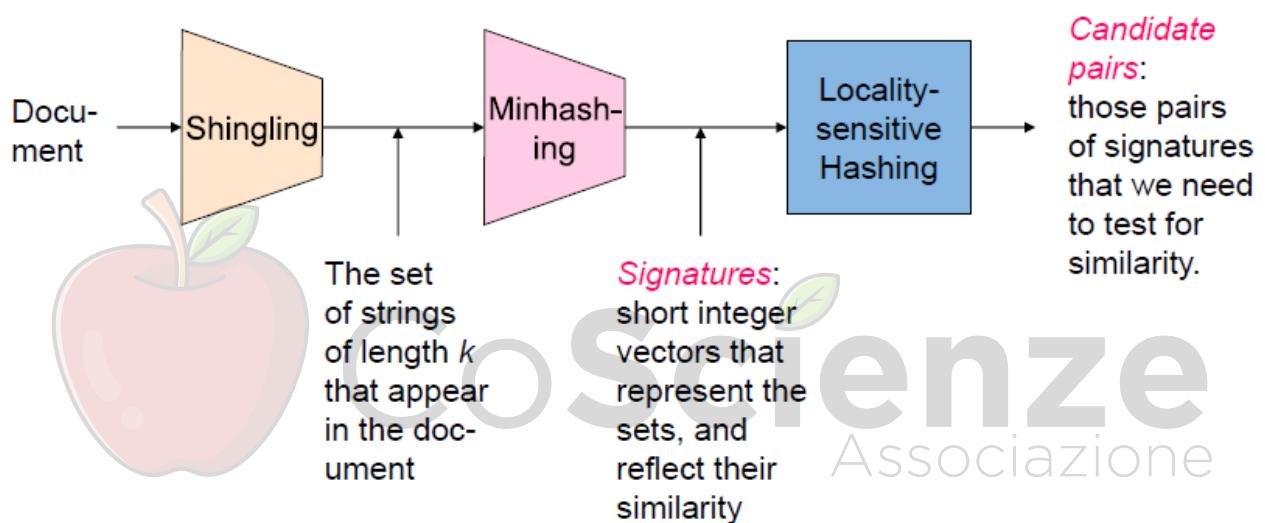
Da sempre uno dei problemi principali del *data mining* è trovare elementi “simili”.

Concetto che può essere applicato per trovare, ad esempio, documenti simili. Per documenti simili possiamo pensare a pagine web duplicate oppure articoli copiati da altri articoli e che quindi rappresentano un plagio.

Per ottenere questo risultato abbiamo bisogno di utilizzare tre tecniche:

1. **Shingling**: convertire documenti in sets, così che possa essere utilizzata una funzione di similarità per set, come quella di Jaccard;
2. **Minhashing**: convertire grossi set in piccole firme preservandone la similarità;
3. **Locality-sensitive hashing**: invece che comparare tutte le coppie di elementi, concentrarsi solo su quelle che possono essere più simili.

In generale, si tratta di mettere all’interno di *buckets* gli elementi utilizzando delle funzioni di hash, così che solo gli elementi che hanno condiviso un bucket almeno una volta vengano presi in considerazione.



Shingling

Uno *k-shingle* (*k-gram*) di un documento è una sequenza di *k* caratteri che appaiono nel documento.

Example: $k = 2$; doc = abcab. Set of 2-shingles = {ab, bc, ca}.

Infatti, documenti che sono intuitivamente simili avranno più shingles in comune.

Tuttavia scegliere il valore di *k* non è semplice, infatti scegliendo un valore troppo basso avremo troppe similarità tra documenti, mentre scegliendone uno troppo alto ne avremo poche.

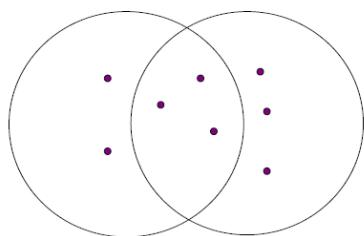
Solitamente il valore viene scelto in base alla tipologia di documento, per le e-mail 5 è un buon numero, mentre per i documenti lunghi solitamente si utilizza 9.

Per salvare spazio ma mantenendo l’informazione degli shingles, si possono utilizzare le funzioni di hash, così che ogni shingle sia mappata da un numero che rappresenta un bucket. Questi numeri vengono chiamati **tokens**.

Jaccard Similarity

La similarità di Jaccard di due sets è una misura della loro intersezione diviso il numero totale di elementi.

$$\text{Sim}(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|.$$



3 in intersection.
8 in union.
Jaccard similarity = 3/8

MinHashing

L'obiettivo è sostituire i sets di shingle poiché troppo ingombranti, anche se già sottoposti ad una funzione di hash, in *signatures* che preservino la loro natura e quindi anche la similarità.

È necessario rifarsi ad un costrutto, ossia la ***characteristic matrix***, tipicamente sparsa. Questa è suddivisa in colonne che corrispondono ai sets, mentre le righe corrispondono ad elementi del set globale.

Quindi avremo un 1 alla riga “e” colonna “S” se “e” è un membro di “S”.

Per effettuare il *minhashing* di questa matrice, si effettua la permutazione di questa per righe, e il valore di minhash della colonna S sarà il numero della riga con il primo 1 nell'ordine permuto

Element	S ₁	S ₂	S ₃	S ₄
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Permutation

Element	S ₁	S ₂	S ₃	S ₄
b	0	0	1	0
e	0	0	1	0
a	1	0	0	1
d	1	0	1	1
c	0	1	0	1

$$h(S_1) = h(S_4) = a; h(S_2) = c; h(S_3) = b;$$

La probabilità che la funzione di minash per una permutazione produca lo stesso valore per due sets è equivalente alla similarità di Jaccard di quei sets. Quindi viene preservata la natura dei sets.

Columns 1 & 2:
Jaccard similarity 1/4.
Signature similarity 1/3

Columns 2 & 3:
Jaccard similarity 1/5.
Signature similarity 1/3

Columns 3 & 4:
Jaccard similarity 1/5.
Signature similarity 0

0	1	1	0
0	0	1	1
1	0	0	0
0	1	0	1
0	0	0	1
1	1	0	0
0	0	1	0

3	1	1	2
2	2	1	3
1	5	3	2

Signature Matrix

Input Matrix

La similarità delle signatures è la frazione di minhash functions (righe) in cui sono uguali.

Ci si aspetta che questo valore sia approssimativamente uguale alla similarità di Jaccard dei sets.

Ovviamente, se consideriamo un miliardo di righe, sarà complicato effettuare la permutazione ciclica di queste, e quindi per risolvere questa problematica si utilizzano delle funzioni di hash random che mappano le righe in "k" buckets.

Quindi piuttosto che scegliere "n" permutazioni random, utilizziamo "n" funzioni di hash.

Si crea una matrice di *signatures*, di righe pari al numero di funzioni di hash e colonne pari al numero di colonne della *characteristic matrix*.

Inizialmente tutti gli elementi vengono impostati ad infinito e poi si utilizza l'algoritmo seguente:

```

for each row r do begin
    for each hash function hi do
        compute hi(r);
        for each column c
            if c has 1 in row r
                for each hash function hi do
                    if hi(r) is smaller than SIG(i, C) then
                        SIG(i, C) := hi(r);
    end;

```

Important: so you hash r only once per hash function, not once per 1 in row r.

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

- (1) Row 1 has 1 in C1
 $h(1)=1 < \text{SIG}(h, C1)=\infty$
 $g(1)=3 < \text{SIG}(g, C1)=\infty$

	C1	C2
h	1	∞
g	3	∞

- (3) Row 3 has 1 in C1 and C2
 $h(3)=3 > \text{SIG}(h, C1)=1$
 $h(3) > \text{SIG}(h, C2)=2$
 $g(3)=2 < \text{SIG}(g, C1)=3$
 $g(3) > \text{SIG}(g, C2)=0$

	C1	C2
h	1	2
g	2	0

- (2) Row 2 has 1 in C2
 $h(2)=2 < \text{SIG}(h, C2)=\infty$
 $g(2)=0 < \text{SIG}(g, C2)=\infty$

	C1	C2
h	1	2
g	3	0

- (4) Row 4 has 1 in C1
 $h(4)=4 > \text{SIG}(h, C1)=1$
 $g(4)=4 > \text{SIG}(g, C1)=2$
Signature Matrix Unchanged

- (5) Row 5 has 1 in C2
 $h(5)=0 < \text{SIG}(h, C2)=2$
 $g(5)=1 > \text{SIG}(g, C2)=0$

	C1	C2
h	1	0
g	2	0

Initial Signature Matrix:

Row	C1	C2
h	∞	∞
g	∞	∞

SIM(C1,C2)=1/5 whereas SIM(SIG(C1),SIG(C2))=0
For bigger matrices the estimation gets closer !!

Locality-Sensitive Hashing

Per ridurre il numero di coppie da comparare per la similarità viene utilizzata LSH.

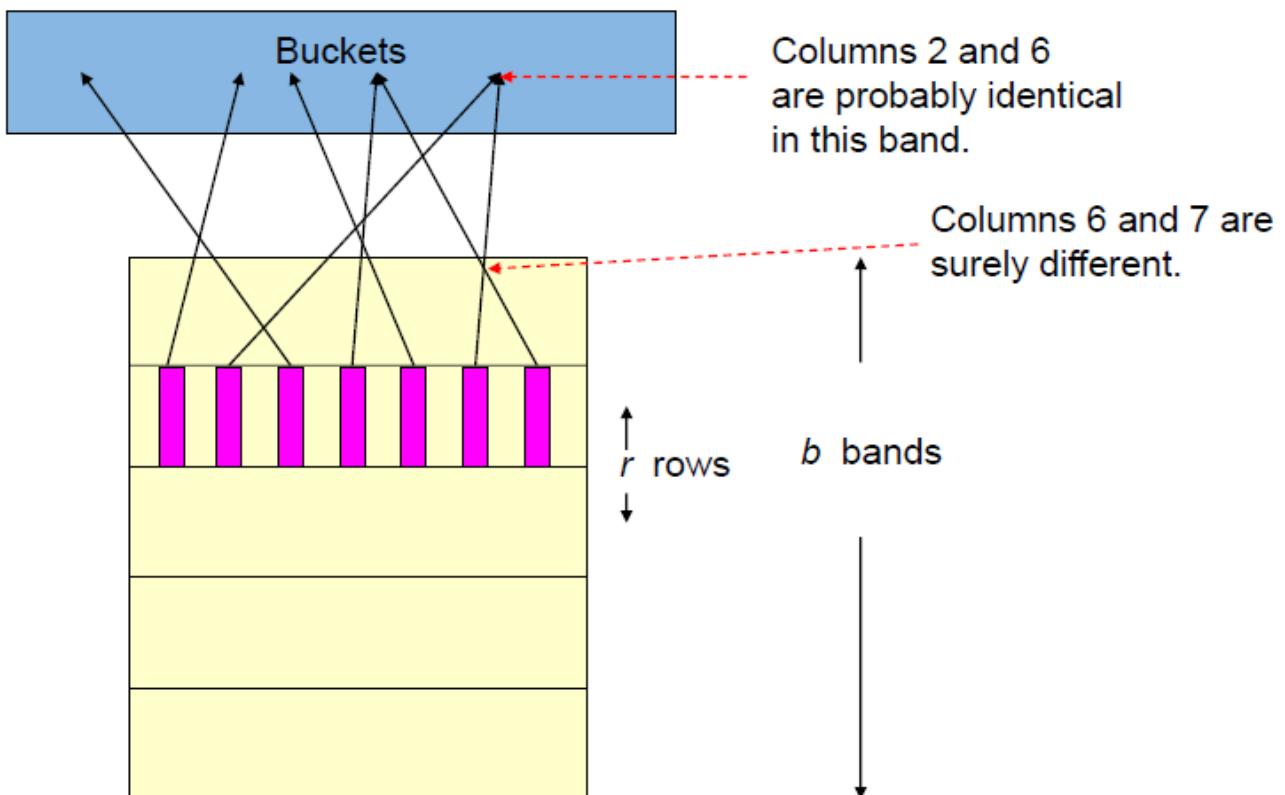
Consiste nell'effettuare l'hash degli elementi molte volte, così che gli elementi simili è più probabile ritrovarli all'interno dello stesso bucket.

La similarità viene definita in base ad una soglia "t" che rappresenterà il numero minimo di righe in cui le firme devono essere uguali.

Ovviamente, gli elementi dissimili che finiscono nello stesso bucket saranno falsi positivi, mentre quelli simili che non finiranno nello stesso bucket rappresentano i falsi negativi.

La matrice M viene suddivisa in "b" bande e "r" righe. Per ogni banda, si effettua l'hash di ogni colonna in una hash table di "k" buckets.

I candidati saranno le colonne che finiranno nello stesso bucket per almeno una banda.

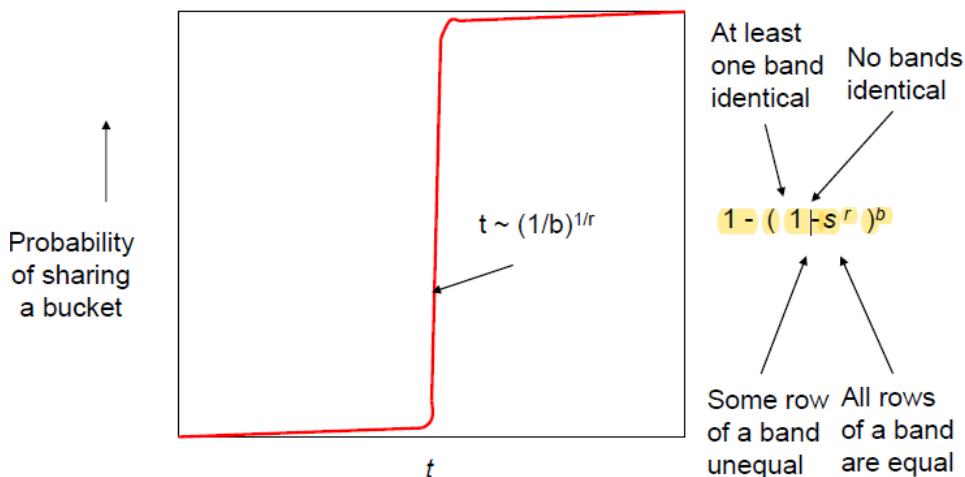


Consideriamo "b" numero di bande, "r" numero di righe di una banda e "s" la similarità di Jaccard di SIG(C1) e SIG(C2).

La probabilità che C1 e C2 siano candidati da comparare è la seguente:

- ▶ probability their signatures agree in all the rows of a band is s^r ;
- ▶ probability they disagree in at least one row of a band is $1-s^r$;
- ▶ probability they disagree in at least one row of each of the bands is $(1-s^r)^b$;
- ▶ probability they agree in all the rows of at least one band and therefore **become a candidate pair**, is $1-(1-s^r)^b$;

Il valore di soglia "t" approssimativamente sarà uguale a $\left(\frac{1}{b}\right)^{\frac{1}{r}}$.



Ricapitolando, quindi, si scelgono *k-shingle* per ciascun documento. Facoltativamente, si può calcolare una funzione di hash per renderli più piccoli.

Si ordinano le coppie di *shingle* dei documenti.

Si sceglie una lunghezza "n" per le firme *minhash* e si calcolano le firme *minhash* per tutti i documenti.

Si sceglie una soglia per definire quando documenti simili dovrebbero essere considerati come una "coppia di candidati".

Si sceglie un numero di bande "b" e un numero di righe "r" tale che $b * r = n$, e "t" è approssimativamente

$$\left(\frac{1}{b}\right)^{\frac{1}{r}}$$

Per evitare falsi negativi, scegli "b" e "r" per abbassare "t".

Per limitare i falsi positivi e migliorare l'efficienza scegliere "b" e "r" per aumentare "t".

Costruire coppie di candidati attraverso la tecnica LSH.

Esaminare le firme di ciascuna coppia di candidati e determinare se la frazione di componenti nella quale sono simili è almeno "t".

Facoltativamente, se le firme sono sufficientemente simili, verificare che i documenti siano veramente simili.

Data Mining

La maggior parte delle aziende dispone di enormi basi di dati contenenti dati di tipo operativo.

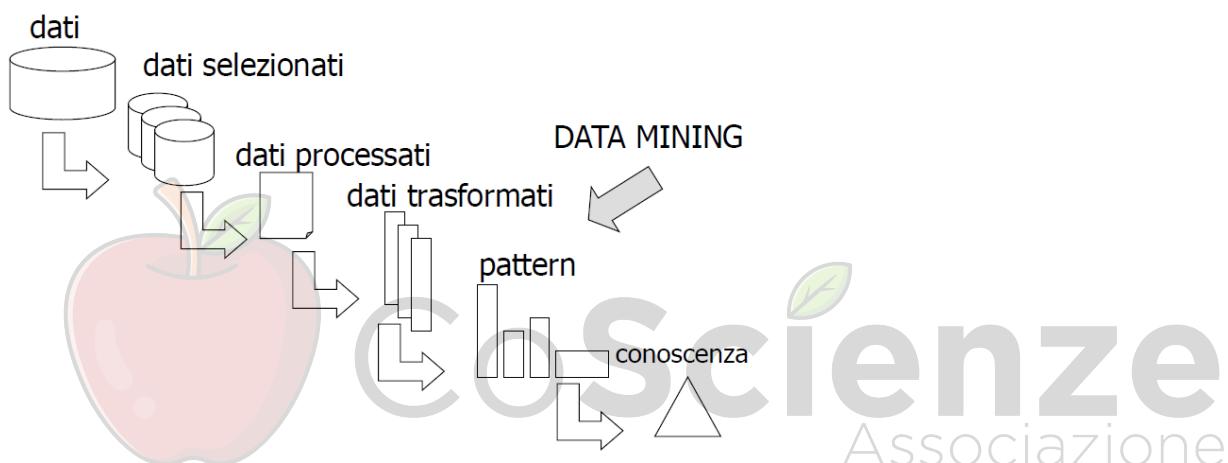
Un processo di Knowledge Discovery si basa sui seguenti elementi:

- *Dati*: insieme di informazioni contenute in una base di dati o data warehouse
- *Pattern*: espressione in un linguaggio opportuno che descrive in modo succinto le informazioni estratte dai dati

I pattern, inoltre, devono rispettare le seguenti caratteristiche:

- *Validità*: i pattern scoperti devono essere validi su nuovi dati con un certo grado di certezza
- *Novità*: misurata rispetto a variazioni dei dati o della conoscenza estratta
- *Utilità*: come, ad esempio, aumento di profitto atteso dalla banca associato alla regola estratta
- *Comprendibilità*

Processo di estrazione



Regole di Associazione

Solitamente abbiamo a disposizione i seguenti dati:

- “**I**” insieme di items;
- “**T**” transazioni che contengono un sottoinsieme di I;
- “**D**” base dati che è l’insieme delle transazioni.

Usiamo le seguenti definizioni:

- Una regola di associazione si definisce così $X \Rightarrow Y$
- **Supporto(X)** = numero di transazioni che contengono X in D
- **Supporto(X \Rightarrow Y)** = Supporto(X U Y)
- **Confidenza(X \Rightarrow Y)** = Supporto(X U Y) / Supporto(X)

*** \Rightarrow uova** : significa che si sta facendo l’analisi su cosa si deve promuovere per aumentare le vendite di uova

Latte \Rightarrow * : significa che si sta facendo l’analisi su cosa deve essere venduto da un negozio che vende latte

TRANSACTION ID	OGGETTI ACQUISTATI
1	<u>A,B,C</u>
2	<u>A,C</u>
3	<u>A,D</u>
4	B,E,F

1	<u>A,B,C</u>
2	<u>A,C</u>
3	<u>A,D</u>
4	B,E,F

- ▶ Regole ottenute:
 - ▶ A \Rightarrow C supporto 50% confidenza 66.6
 - ▶ C \Rightarrow A supporto 50% confidenza 100%

Algoritmo Apriori

Ha come obiettivo quello di trovare un insieme di items che hanno un supporto minimo, cioè i *frequent itemset*.

Si parte quindi col decidere una soglia minima di supporto, ad esempio il 50% delle transazioni, e si parte con l'algoritmo definito nel seguente modo:

- Ad ogni passo:
 - costruisce un insieme di itemset candidati
 - conta il numero di occorrenze di ogni candidato (accedendo alla base di dati)
 - determina i candidati che sono frequent itemset
- Al passo "k" avremo:
 - $C(k)$: insieme di itemset candidati di dimensione k (potenzialmente frequent itemset)
 - $L(k)$: insieme di frequent itemset di dimensione k

$$L_1 = \{\text{singoli items frequenti}\}$$

```

for (k=1,  $L_k \neq \{\}$  , k++)
  begin
     $C_{k+1}$  = nuovi candidati generati da  $L_k$ 
    foreach transazione t in D do
      incrementa il conteggio di tutti i candidati in  $C_{k+1}$ 
      che sono contenuti in t
     $L_{k+1}$  = candidati in  $C_{k+1}$  con supporto minimo
  end
  frequent itemsets =  $U_k L_k$ 

```

Una delle proprietà dell'algoritmo Apriori è che un sottoinsieme di un frequent itemset è un frequent itemset.

Questo porta a due soluzioni:

- *Soluzione A*: dato $L(k)$, $C(k+1)$ si genera aggiungendo ogni item agli itemset $L(k)$
- *Soluzione B (ottimizzata)*: da $C(k)$ si possono cancellare tutti i candidati che contengono un sottoinsieme non frequent

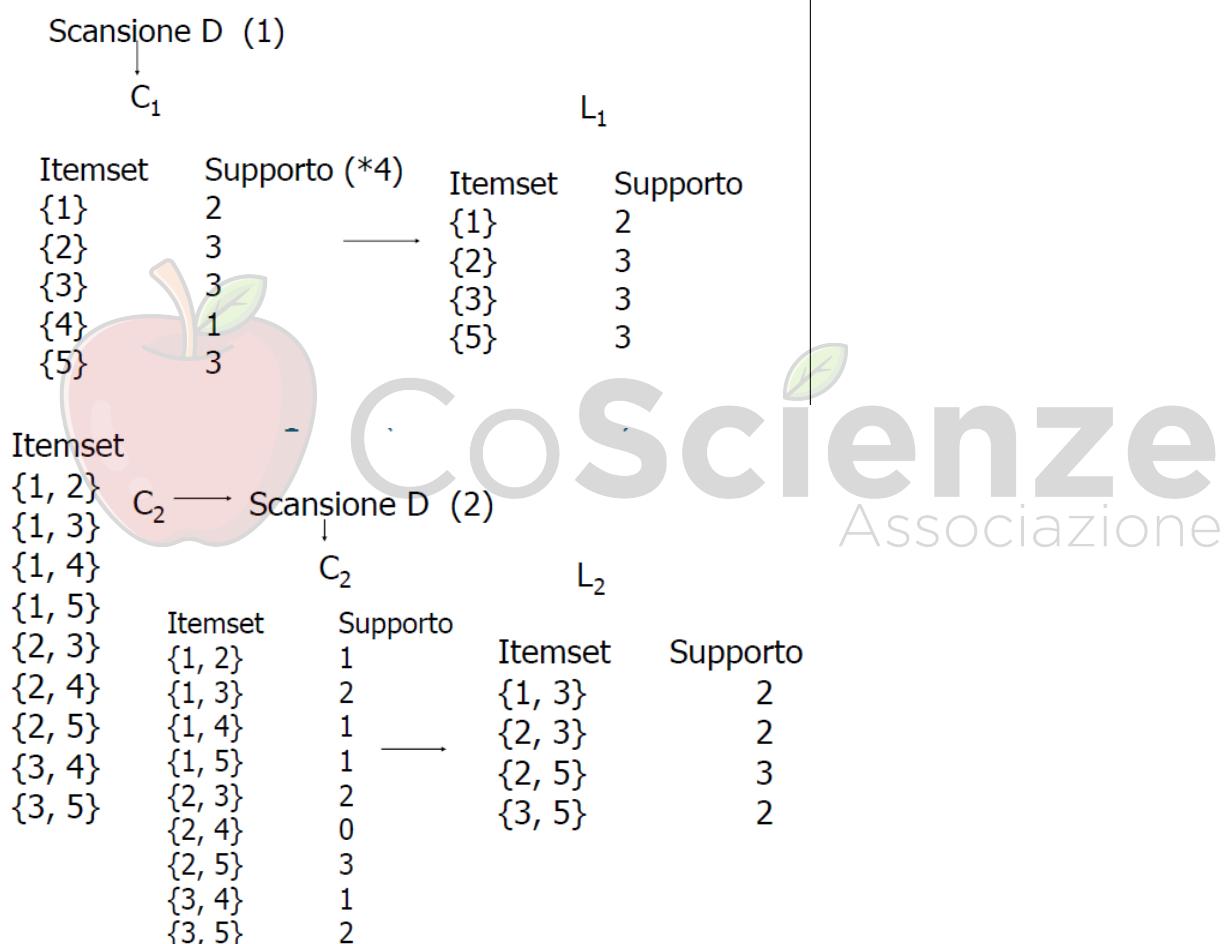
Esempio:

Base di dati D

TID	Items
100	1, 3, 4
200	2, 3, 5
300	1, 2, 3, 5
400	2, 5

Soluzione A

Esempio: Soluzione A



Soluzione B**Esempio: Soluzione B**

Scansione D (1)

C₁L₁

Itemset	Supporto (*4)	Itemset	Supporto
{1}	2	{1}	2
{2}	3	{2}	3
{3}	3	{3}	3
{4}	1	{5}	3
{5}	3		

Itemset

{1, 2} C₂ → Scansione D (2){1, 3} C₂L₂

{2, 3}

Itemset	Supporto	Itemset	Supporto
{1, 2}	1	{1, 3}	2
{1, 3}	2	{2, 3}	2
{1, 5}	1	{2, 5}	3
{2, 3}	2	{3, 5}	2
{2, 5}	3		
{3, 5}	2		

Regole di Classificazione e Regressione

$$P_1(X_1) \wedge \dots \wedge P_k(X_k) \Rightarrow Y = c$$

Y attributo *dipendente*X_i attributi *predittivi*P_i(X_i) condizione su attributo X_iSe l'attributo dipendente è categorico, si ottiene una regola di **classificazione**.Se l'attributo dipendente è numerico, si ottiene una regola di **regressione**.

Esempio:

Regola di classificazione

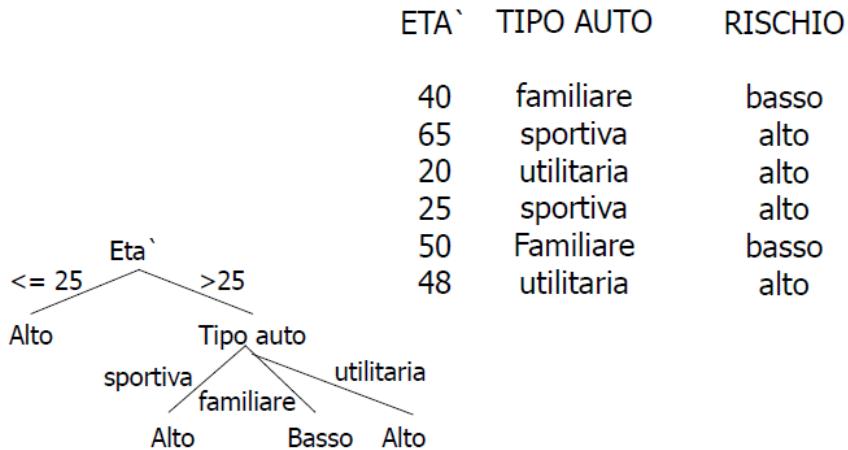
$$\text{età} > 25 \wedge \text{tipo_auto} \in \{\text{Sportiva, utilitaria}\} \Rightarrow \text{rischio} = \text{alto}$$

Solitamente i dati a disposizione vengono splittati in due insiemi:

- *Training Set (70%)*
- *Test Set (30%)*

Alberi di Decisione

Gli alberi di decisione permettono di rappresentare le regole di classificazione come un albero.



Hanno due fasi:

1. **Build**: si costruisce l'albero iniziale, partizionando ripetutamente il training set sul valore di un attributo, fino a quando tutti gli esempi in ogni partizione appartengono ad una sola classe;
2. **Pruning**: si pota l'albero, eliminando rami dovuti a rumore o fluttuazioni statistiche

Builtree(training set T)

```
{Partition(T)}
```

Partition(Data set S)

```
{
```

```
if (tutti i punti in S sono nella stessa classe) then
    return
```

```
foreach attributo A do
```

```
    valuta gli splits su A (usando un algoritmo di split)
```

```
    usa split "migliore" per partizionare S in S1 e S2
```

```
    Partition(S1)
```

```
    Partition(S2)
```

Gli split possono riguardare attributi

- numerici
- categorici

E possono essere:

- Binari
- Multipli

Valutazione Classificatori

Nella valutazione di un classificatore si considerano metriche come:

- Accuratezza
- Velocità di classificazione
- Robustezza in presenza di valori mancanti o errori

- Scalabilità
- Facilità di Interpretazione

La fonte primaria per la stima dell'accuratezza è la **confusion matrix**.

		True Class	
		Positive	Negative
Predicted Class	Positive	True Positive Count (TP)	False Positive Count (FP)
	Negative	False Negative Count (FN)	True Negative Count (TN)

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

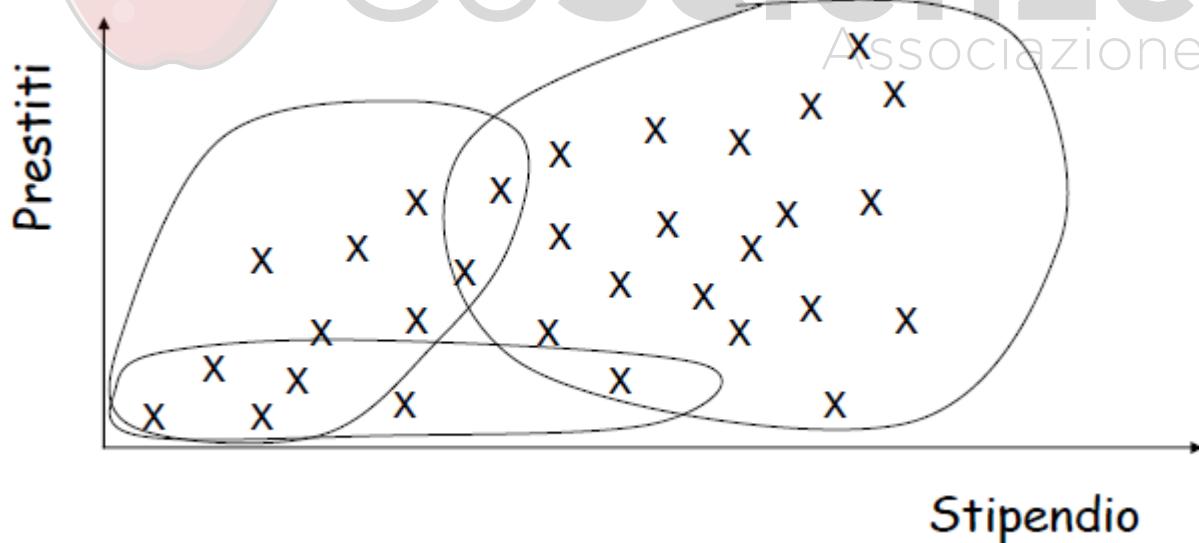
$$\text{Tasso True Positive} = \frac{TP}{TP + FN}$$

$$\text{Tasso True Negative} = \frac{TN}{TN + FP}$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad \text{Recall} = \frac{TP}{TP + FN}$$

Clustering

Ha l'obiettivo di raggruppare all'interno dello stesso insieme gli elementi che risultano più simili tra loro e di avere non similarità tra elementi di gruppi diversi.



La similitudine tra due oggetti si ottiene applicando una **funzione di distanza**.

Esistono due approcci principali:

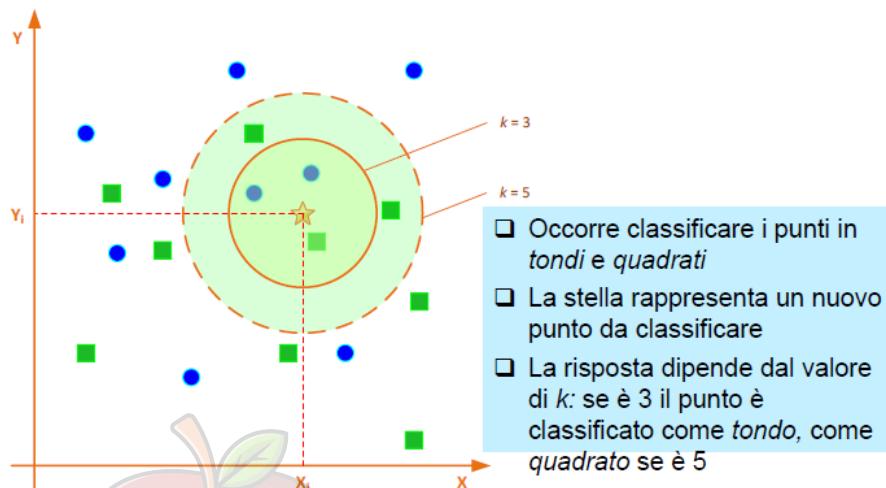
- **Algoritmi di partizionamento:** dato un numero di cluster "k", partiziona i dati nei "k" cluster mantenendo criteri ottimali

- **Algoritmi Gerarchici:** si considera un cluster per elemento e si prosegue combinando passo dopo passo i cluster più simili, oppure, si parte da un singolo cluster con tutti gli elementi e si continua a suddividerlo fino ad ottenere "k" cluster

K-Nearest Neighbor (KNN)

È utilizzabile sia per la classificazione che per la regressione in cui "k" è il parametro che rappresenta il numero di "vicini" da considerare per l'algoritmo.

Ogni nuovo oggetto viene classificato usando la maggioranza dei suoi "k" oggetti più vicini, misurandone la distanza mediante opportuna funzione di distanza scelta in fase di progettazione.



Valori più grandi di "k" riducono gli effetti distorsivi (errori, valori mancanti) nei dati, ma rendono anche meno marcati i confini tra le classi.

La **Cross Validation** è una delle tecniche più frequentemente utilizzate per trovare il valore ottimale per "k" e per la misura di distanza

Cross Validation

Tecnica sperimentale per inferire valori ottimali per parametri di modelli predittivi.

Prevede le seguenti fasi:

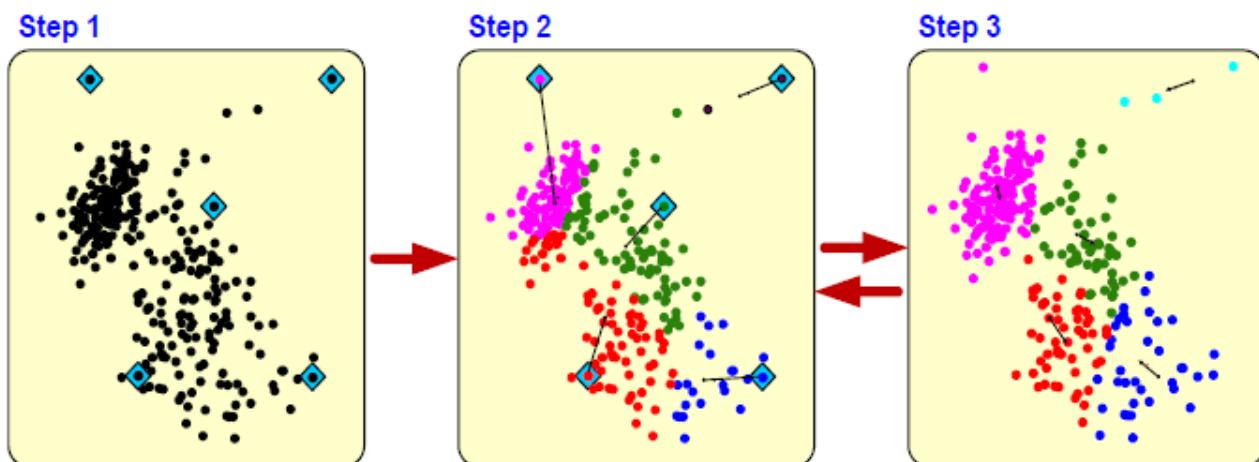
1. si sceglie un numero v in modo casuale e, per ogni valore potenziale di k , si classifica, a turno, uno dei v insiemi usando gli altri $v-1$ come training set
2. per ognuno di tali v esperimenti si calcola l'errore quadratico metrico, aggregando tali valori
3. si sceglie il valore di k che minimizza l'errore totale

K-Means

Si determina inizialmente il numero di cluster da costruire "k" e la funzione di distanza da utilizzare.

Successivamente:

1. Seleziona in modo casuale k punti come centri iniziali dei cluster.
2. Assegna ciascun nuovo punto al cluster il cui centro è più vicino.
3. Ricalcola i centri dei cluster aggiornati
4. *Step di ripetizione:* Ripetere gli step 2 e 3 finché si raggiungerà un dato criterio di convergenza sostanzialmente, l'assegnamento di punti che rende stabili le classi



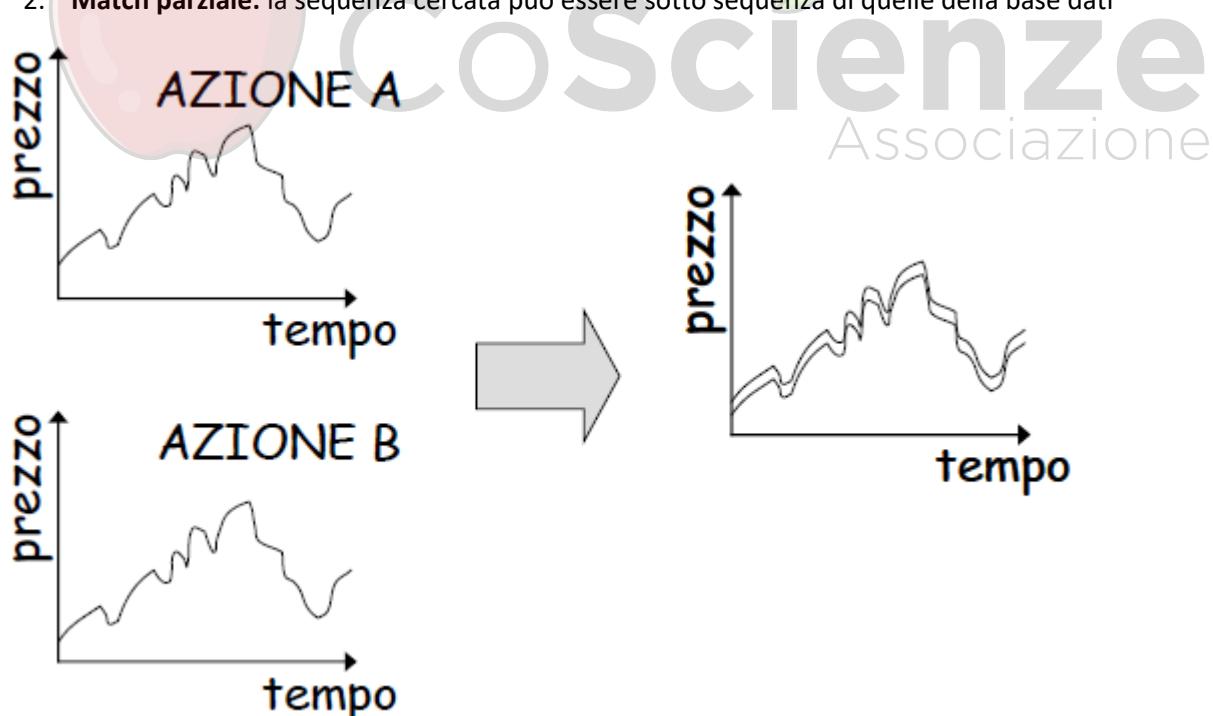
Similarity Search

L'obiettivo è determinare sequenze simili ad una sequenza data.

Ad esempio, identificazione delle società con comportamento simile di crescita, determinazione di prodotti con profilo simile di vendita oppure identificazione di azioni con andamento simile.

Utilizza due tipi di interrogazione:

1. **Match completo:** la sequenza cercata e le sequenze della base dati hanno la stessa lunghezza;
2. **Match parziale:** la sequenza cercata può essere sotto sequenza di quelle della base dati

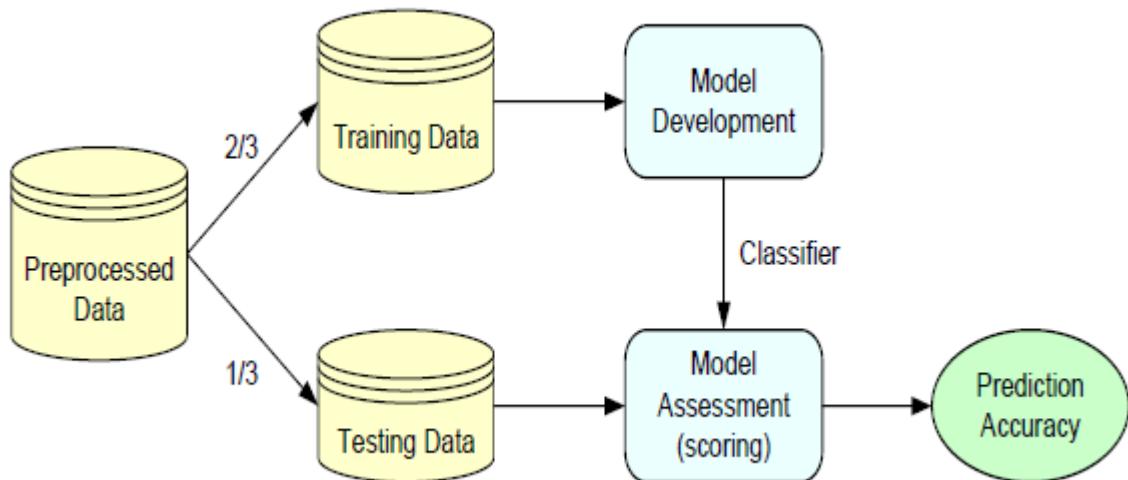


Classification

Nei dati a disposizione, solitamente, è presente un set di classi con cui ogni elemento è etichettato. L'obiettivo è etichettare un nuovo elemento sulla base della conoscenza acquisita dal training set.

Di base, si effettua una suddivisione dei dati a disposizione in training-test set.

Il motivo per cui suddividiamo i nostri dati in set di training e test è che siamo interessati a misurare quanto bene il nostro modello si generalizza a dati nuovi, mai visti prima.



Esistono diversi tipi di classificatori:

- Decision trees
- Bayesian
- Statistic
- Neural net
- Knn
- ...

CoScienze
Associazione

Per valutare un modello dobbiamo:

1. Splittare il dataset in training e test set
2. Costruire un modello sul training set
3. Valutarlo utilizzando il test set

```

from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# create a synthetic dataset with data (X) and labels (y)
X, y = make_blobs(random_state=0)
# split data and labels into training and test sets [70-30%]
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
# instantiate a model and fit it to the training set
logreg = LogisticRegression().fit(X_train, y_train)
# evaluate the model on the test set
print("Test set score: {:.2f}".format(logreg.score(X_test,
y_test)))
  
```

Test set score: 0.88
Process finished with exit code 0

Cross Validation

Come già detto nei capitoli precedenti, la *cross validation* è un metodo statistico che permette di valutare la generalizzazione del nostro modello

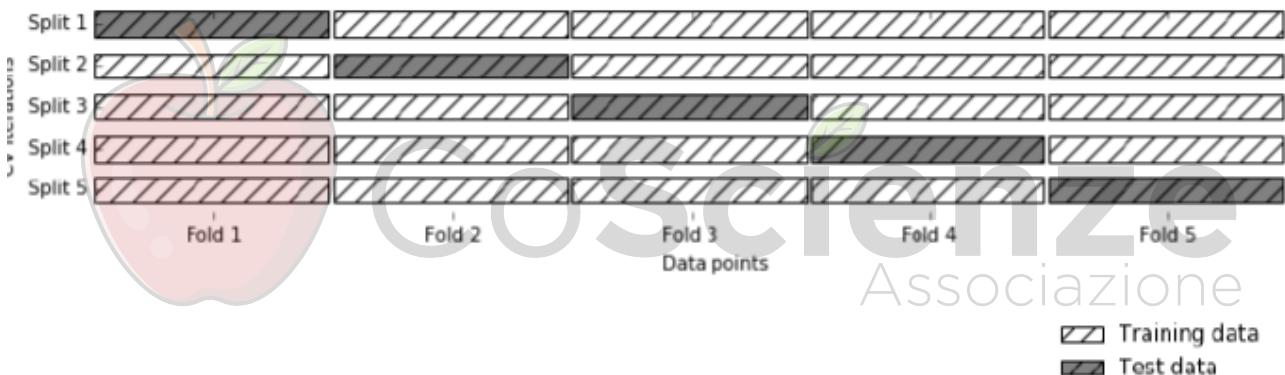
"The data is split repeatedly, and multiple models are trained"

La versione più comunemente utilizzata è la **k-fold cross validation**, nella quale i dati vengono prima partizionati in parti di uguale dimensione, chiamate *folds*, e ad ogni iterazione viene scelta una di queste parti come test set e le altre fungono da training set. Quindi, quando usiamo la cross validation ogni esempio viene utilizzato come test set esattamente una volta.

Solitamente, senza l'uso di questa tecnica, se siamo fortunati tutti gli esempi difficili da classificare finiranno nel training set e quindi la valutazione del nostro modello sarà buona. Contrariamente, se tutti gli esempi difficili finiscono nel test set il nostro modello avrebbe un risultato finale non adeguato.

Inoltre, utilizzando la cross-validation, utilizziamo dati a disposizione in maniera efficiente, poiché ad ogni split utilizziamo come dati di training alcuni di quelli che non avevamo precedentemente usato.

D'altra parte, però, lo svantaggio principale è dato dall'incremento del costo di computazione del nostro algoritmo.



Di seguito un'implementazione della cross validation:

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
logreg = LogisticRegression()
scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

Average cross-validation score: 0.96

Confusion Matrices

All'interno della matrice di confusione abbiamo quattro categorie:

- **True Positives (TP):** i casi per cui il classificatore correttamente ha predetto positivamente un esempio;

- **True Negatives (TN):** i casi per cui il classificatore correttamente ha predetto negativamente un esempio;
- **False Positives (FP):** i casi per cui il classificatore in maniera non corretta ha predetto positivamente un esempio negativo;
- **False Negatives (FN):** i casi per cui il classificatore in maniera non corretta ha predetto negativamente un esempio positivo;

	TN	FP
negative class		
positive class	FN	TP
predicted negative		predicted positive

Di seguito un esempio di implementazione:

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

digits = load_digits()
y = digits.target == 9
X_train, X_test, y_train, y_test = train_test_split(digits.data,
y, random_state=0)
logreg = LogisticRegression().fit(X_train, y_train)
pred_logreg = logreg.predict(X_test)
print("logreg score: {:.2f}".format(logreg.score(X_test, y_test)))
confusion = confusion_matrix(y_test, pred_logreg)
print("Confusion matrix:\n{}".format(confusion))
```

```
logreg score: 0.98
Confusion matrix:
[[401  2]
 [ 8 39]]
```

I numeri lungo la diagonale principale corrispondono alle classificazione **corrette**.

L'altra diagonale rappresenta gli esempi mal classificati.

		True Class	
		Positive	Negative
Predicted Class	Positive	True Positive Count (TP)	False Positive Count (FP)
	Negative	False Negative Count (FN)	True Negative Count (TN)

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

$$\text{True Positive Rate} = \frac{TP}{TP + FN}$$

$$\text{True Negative Rate} = \frac{TN}{TN + FP}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

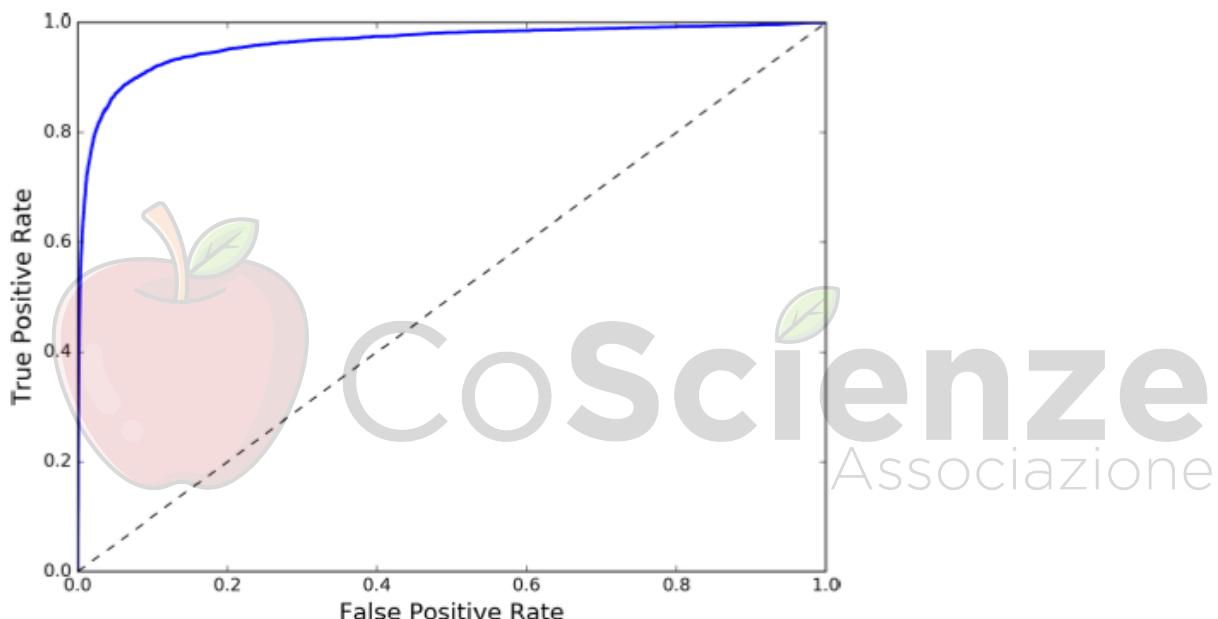
- **Accuracy:** È un modo per riassumere il risultato nella matrice di confusione;
- **Precision:** misura quanti campioni previsti come positivi sono in realtà positivi;
- **Recall:** misura quanti sono i campioni positivi indovinati dalle previsioni positive. Sfortunatamente, l'aumento della *precision* riduce la *recall*, e viceversa (**precision/recall tradeoff**)
- **F1 Score:** È un modo per riassumere tutti i risultati, essendo una media armonica di *precision* e *recall*

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

ROC Curve (Receiver Operating Characteristic)

È molto simile alla curva *precision/recall*, ma in questo caso, la curva ROC traccia il tasso positivo reale (*recall*) contro il tasso di falsi positivi.

Anche in questo caso c'è un tradeoff, infatti più alto è il *recall* più falsi positivi il classificatore produrrà.



Un buon classificatore rimane il più lontano possibile da quella linea tratteggiata.

Soltanmente si preferisce la curva *precision/recall* alla ROC quando:

- La classe positiva è rara
- Siamo più interessati ai falsi positivi che ai falsi negativi

Multiclass Classification

Riesce a distinguere tra più di due classi.

Per ottenere questo, esistono due strategie:

- **One Versus All (OvA):** si utilizza un sistema nel quale vengono allenati N classificatori diversi, ognuno per classe di output, e quindi per ogni elemento da classificare si effettua la *predict* in ogni classificatore, scegliendo quello che ha dato il punteggio maggiore.
- **One Versus One (OvO):** si utilizza un sistema di classificatori binari, ognuno per ogni coppia di classe di output. Quindi avremo bisogno di $N*(N-1)/2$ classificatori.

Il vantaggio principale è che ogni classificatore ha bisogno solo di essere allenato sulla parte del training set che contiene le classi che deve distinguere.

Per molti algoritmi di classificazione binaria *OvA* viene preferito, mentre *OvO* viene preferito quando c'è poca scalabilità e quindi è meglio suddividere il lavoro ed i dati tra i vari classificatori.

Multilabel Classification

Utile quando è necessario restituire in output più classi per ogni istanza.

Un esempio può essere quello in cui abbiamo necessità di classificare la presenza all'interno di una foto di più soggetti, e quindi in output avremo un array di booleani che indica la presenza o meno di una classe.

Per valutare l'accuratezza di un sistema del genere, un approccio può essere utilizzare l'F1 score per ogni label e fare la media dei risultati.

Multioutput – Multiclass Classification

È una semplice generalizzazione della multilabel dove ogni label può essere una multiclass.



Training Models

Un modello lineare effettua una previsione semplicemente calcolando la somma ponderata delle caratteristiche di input, più una costante chiamata termine **bias** (chiamato anche termine di intercettazione)

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

- ▶ \hat{y} is the predicted value
- ▶ n is the number of features
- ▶ x_i is the i^{th} feature value
- ▶ θ_j is the j^{th} model parameter
 - ▶ including the bias term θ_0 and the feature weights $\theta_1, \dots, \theta_n$

Per trovare il valore di θ che riduca al minimo la funzione di costo, esiste una soluzione in forma chiusa. Un'equazione matematica che fornisce direttamente il risultato.

Questa è chiamata **equazione normale**

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

- ▶ $\hat{\theta}$ is the value of θ that minimizes the cost function
- ▶ y is the vector of target values containing $y^{(1)}$ to $y^{(m)}$

L'equazione normale calcola l'inverso di $X(T) \cdot X$ che è una matrice $n \times n$.

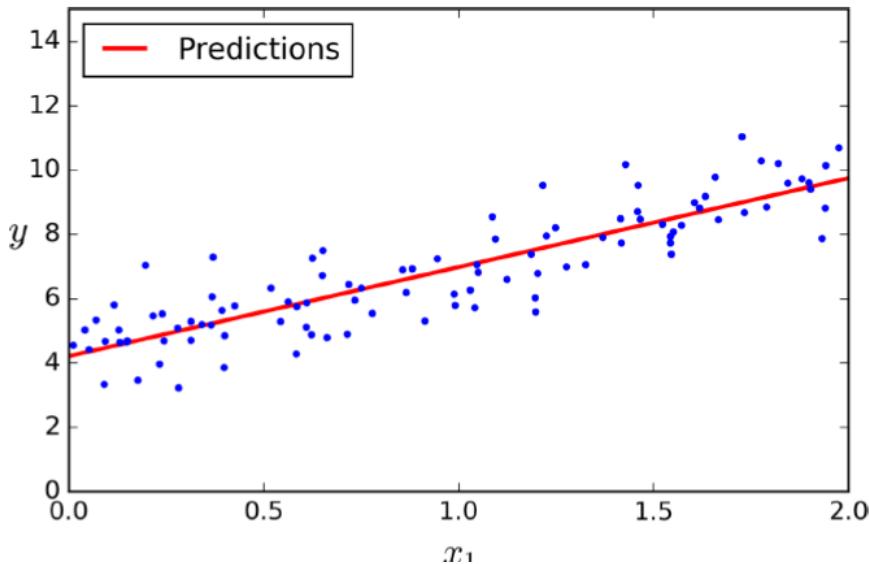
La complessità computazionale dell'inversione di tale matrice varia da $O(n^{2.4})$ a $O(n^3)$ a seconda dell'implementazione.

L'equazione normale diventa molto lenta quando il numero delle caratteristiche aumenta (ad es. 100.000)

Di seguito un esempio in codice:

```
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
print(theta_best)
array([[ 4.21509616],
       [ 2.77011339]])

#Apply regression
X_new = np.array([[0, 2]])
X_new_b = np.c_[np.ones((2, 1)), X_new] # add x0 = 1 to each instance
y_predict = X_new_b.dot(theta_best)
```



Il lato positivo è che questa equazione è lineare per quanto riguarda il numero di istanze nel training set $O(m)$.

Dopo aver addestrato il modello di regressione lineare (usando l'equazione normale o qualsiasi altro algoritmo), le previsioni sono molto veloci.

Esistono diversi modi per addestrare un modello di regressione lineare più adatti per i casi in cui ci sono un gran numero caratteristiche.

Tra questi il **Gradient Descent**.

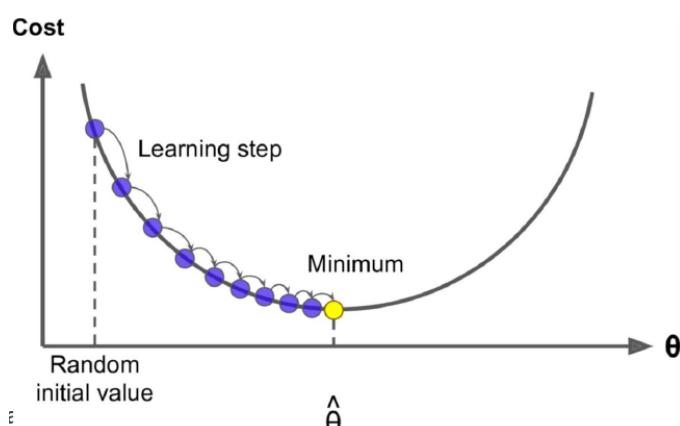
Gradient Descent

L'idea generale del Gradient Descent è di modificare parametri in modo iterativo al fine di ridurre al minimo una funzione di costo.

Misura il gradiente locale della funzione di errore per quanto riguarda il vettore dei parametri θ e si muove in direzione della pendenza discendente.

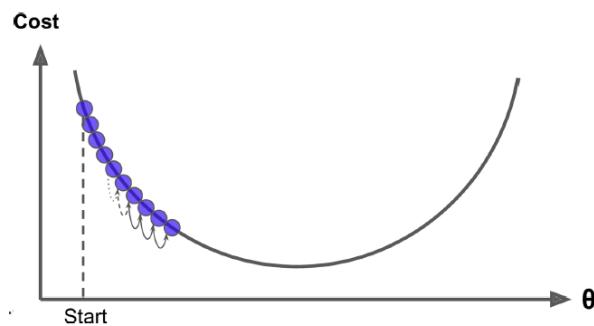
Una volta che la pendenza è zero, ha raggiunto il minimo.

Inizialmente i pesi del modello sono settati in modo casuale, questa fase è chiamata **random initialization**.



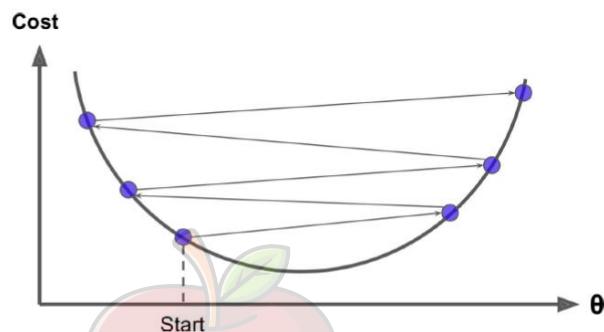
Un parametro importante nel Gradient Descent è la dimensione dei passi determinata dall'iperparametro **"learning rate"**.

Se il tasso di apprendimento è troppo piccolo, allora l'algoritmo dovrà passare attraverso molte iterazioni per convergere.



Se il tasso di apprendimento fosse troppo alto potrebbe saltare e finire sull'altro lato, forse anche più in alto di prima.

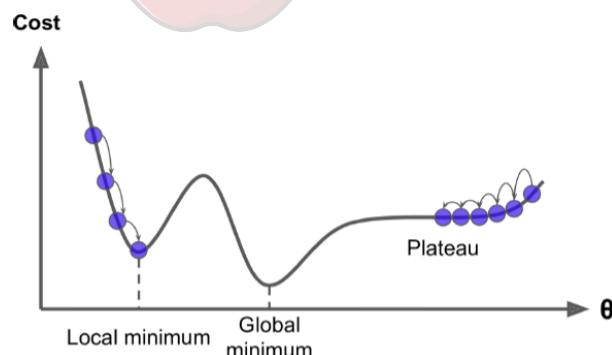
Ciò potrebbe far divergere l'algoritmo, con valori più grandi, non riuscendo a trovare una buona soluzione.



La seguente figura mostra le due sfide principali con l'uso del Gradient Descent.

Se l'inizializzazione casuale avvia l'algoritmo a sinistra, questi convergerà a un minimo locale.

Se inizia a destra, ci vorrà molto tempo e se si ferma troppo presto non lo raggiungerà il minimo globale.



La funzione di costo MSE per un modello di regressione lineare è una funzione convessa.

Ciò implica che non ci sono minimi locali, solo uno globale.

Di conseguenza per il Gradient Descent è garantito avvicinarsi al minimo globale.

Addestrare un modello significa cercare una combinazione di parametri che riducano al minimo una funzione di costo. Più parametri ha un modello, più dimensioni ha questo spazio, e più difficile è la ricerca. La funzione di costo è convessa nel caso di Regressione Lineare.

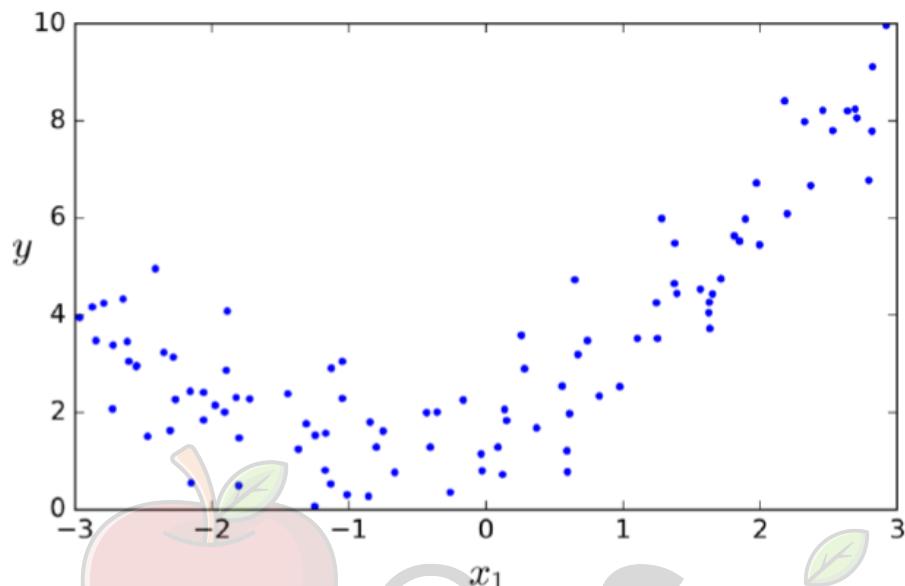
Polynomial Regression

Sorprendentemente, si può effettivamente utilizzare un modello lineare per adattarlo a dati non lineari. Un modo semplice per farlo è aggiungere potenze di ciascuna funzione come nuove *features*, quindi addestrare un modello lineare su questo set esteso di caratteristiche.

Questa tecnica è chiamata **regressione polinomiale**.

Consideriamo alcuni dati non lineari, basati su un semplice equazione quadratica (più un po' di rumore)

$$y = ax^2 + bx + c$$

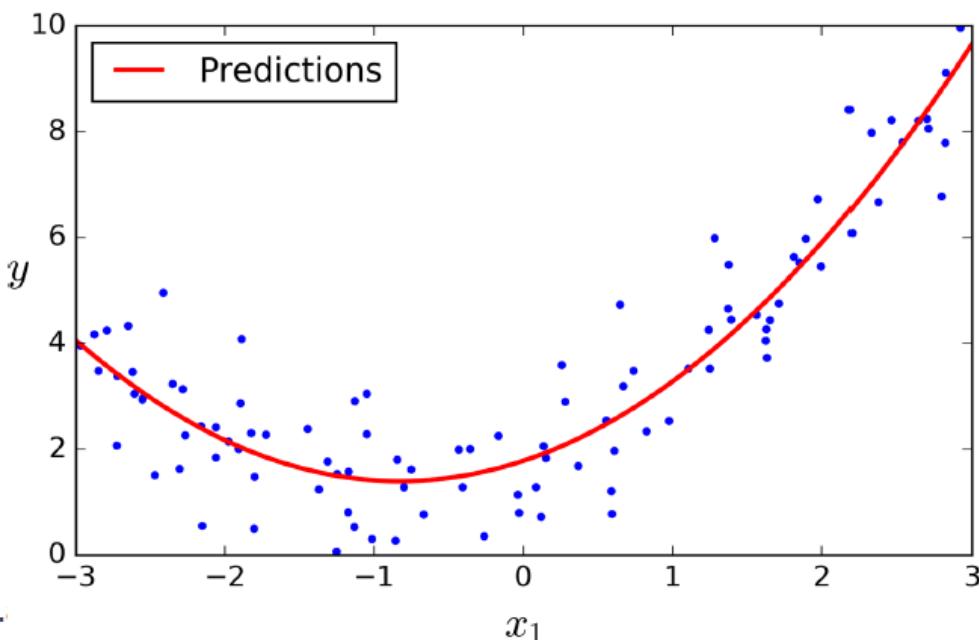


Una linea retta non si adatterà mai correttamente a questi dati.

La classe *PolyomialFeatures* di Scikit-Learn trasforma i nostri dati di addestramento, aggiungendo il quadrato (polinomio di 2° grado) di ogni funzionalità nel set di training come nuove funzionalità

```
#Add the square
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
print(X[0])
print(X_poly[0])
...
array([-0.75275929])
array([-0.75275929,  0.56664654])

#Apply regression
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
```



Se si esegue la regressione polinomiale di grado molto alto, probabilmente il modello si adatterà ai dati di addestramento molto meglio rispetto ad una semplice regressione lineare.

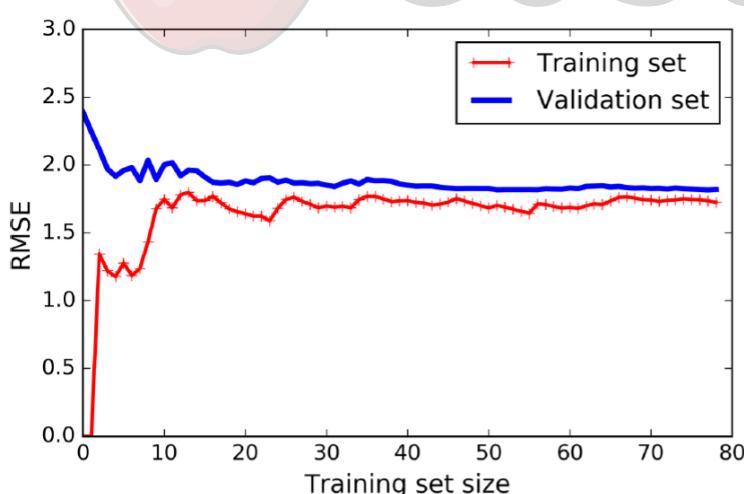
Il modello di regressione polinomiale di grado alto potrebbe però essere in *overfitting*.

Invece, un modello lineare potrebbe non adattarsi ai dati di addestramento, quindi essere in *underfitting*.

Learning Curves

Per ottenere il valore ottimale del grado del polinomio, una soluzione è la *cross-validation*.

Un'altra è studiare la **curva di apprendimento**, grafico delle prestazioni del modello sul set di allenamento e sul set di validazione in funzione della dimensione del training set.



Inizialmente, con le prime istanze, il modello riesce ad adattarsi perfettamente ai dati di training.

Tuttavia, man mano che nuove istanze vengono aggiunte al set di formazione è impossibile per il modello adattare perfettamente i dati di addestramento.

L'errore sale fino a raggiungere un **plateau**, punto in cui aggiungere nuove istanze al training set non rende l'errore medio migliore o peggiore.

Le curve di apprendimento discusse sono tipiche di un modello sottodimensionato (*underfitting*).

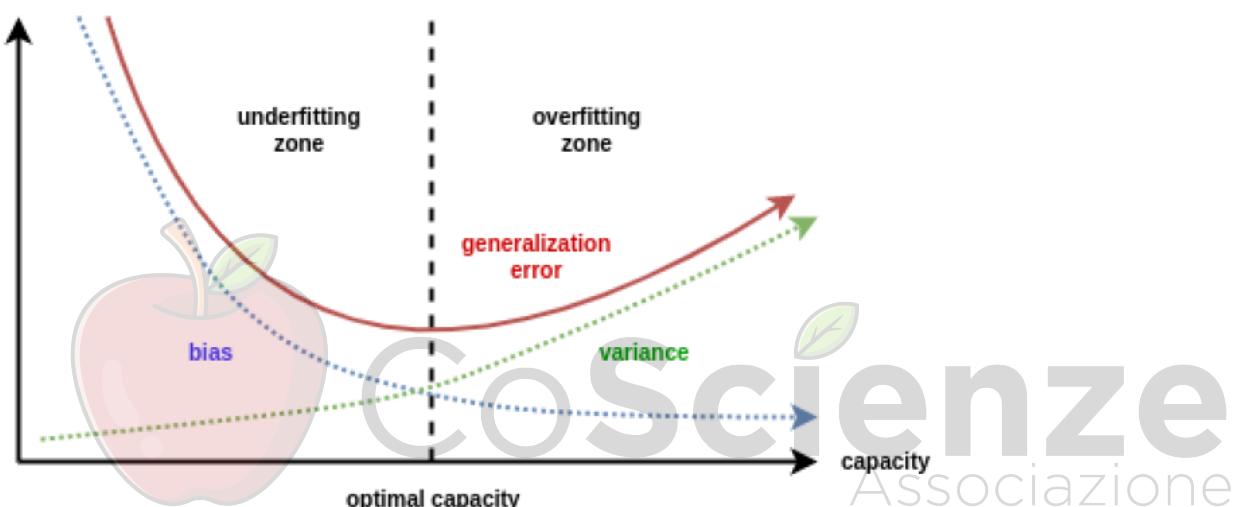
Se un modello non si adatta ai dati di addestramento, aggiungere più istanze di training non aiuta.

È necessario usare un modello più complesso, o trovare feature migliori.

Un importante risultato teorico della statistica e del ML è che l'errore di generalizzazione di un modello può essere espresso come la somma di tre errori diversi:

- **Bias:** Questa parte dell'errore di generalizzazione è dovuta a un errore di ipotesi, come ipotizzare che i dati siano lineari mentre invece sono quadratici.
Un valore alto di questo errore può significare che molto probabilmente il modello sia in *underfitting*;
- **Varianza:** Questa parte è dovuta all'eccessiva sensibilità del modello a piccole variazioni nei dati di allenamento. È probabile che un modello con molti gradi di libertà abbia un valore elevato varianza e sia in *overfitting*;
- **Irreducible Error:** Questa parte è dovuta alla rumorosità dei dati stessi. L'unico modo per ridurre questa parte dell'errore è pulire i dati.

L'aumento della complessità di un modello in genere aumenterà la sua varianza e ridurrà il bias.
Di conseguenza, ridurre la complessità di un modello aumenta il suo bias e riduce la sua varianza.



Ridge Regression

Ridge Regression (chiamata anche *regolarizzazione di Tikhonov*) è una versione regolarizzata della regressione lineare.

Alla funzione di costo viene aggiunto un termine di regolarizzazione pari a $\alpha \sum_1^n \theta_i^2$.

Il termine di regolarizzazione dovrebbe essere solo aggiunto alla funzione di costo durante il training.

Ciò costringe l'algoritmo di apprendimento non solo ad adattare i dati ma anche a mantenere i pesi del modello il più piccoli possibile.

La seguente equazione presenta la funzione di costo della Ridge Regression:

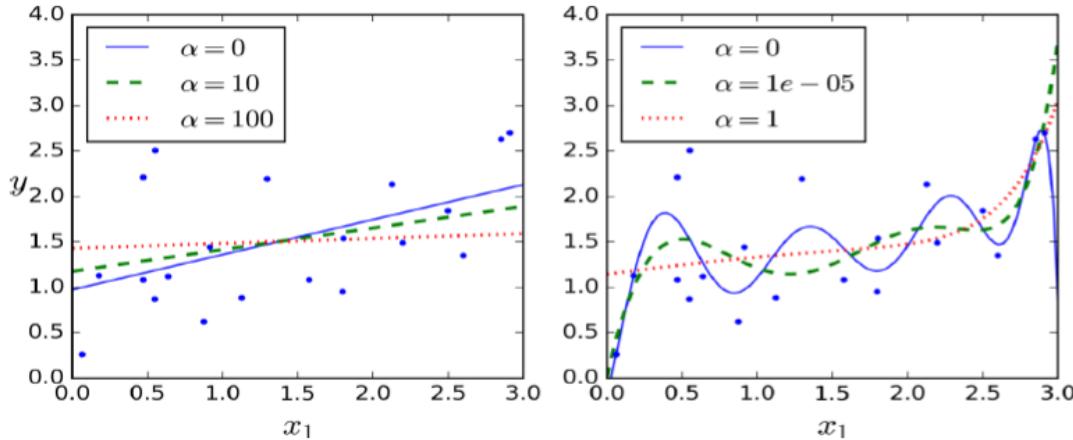
$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

L'iperparametro α controlla quanto si vuole regolarizzare il modello:

- Se $\alpha=0$ allora diventerà una regressione lineare

- Se α è molto grande, tutti i pesi finiscono molto vicini allo zero ed il risultato sarà una linea piatta che attraversa la media dei dati

È importante scalare i dati prima di eseguire Ridge Regression ed ogni altro modello.



Possiamo eseguire la Ridge Regression calcolando l'equazione in forma chiusa o eseguendo Gradient Descent.

La seguente equazione mostra la soluzione in forma chiusa:

$$\hat{\theta} = (X^T \cdot X + \alpha A)^{-1} \cdot X^T \cdot y$$

Dove A è la matrice di identità $n \times n$ eccetto con uno 0 nella cella in alto a sinistra, corrispondente al termine di bias.

Lasso Regression

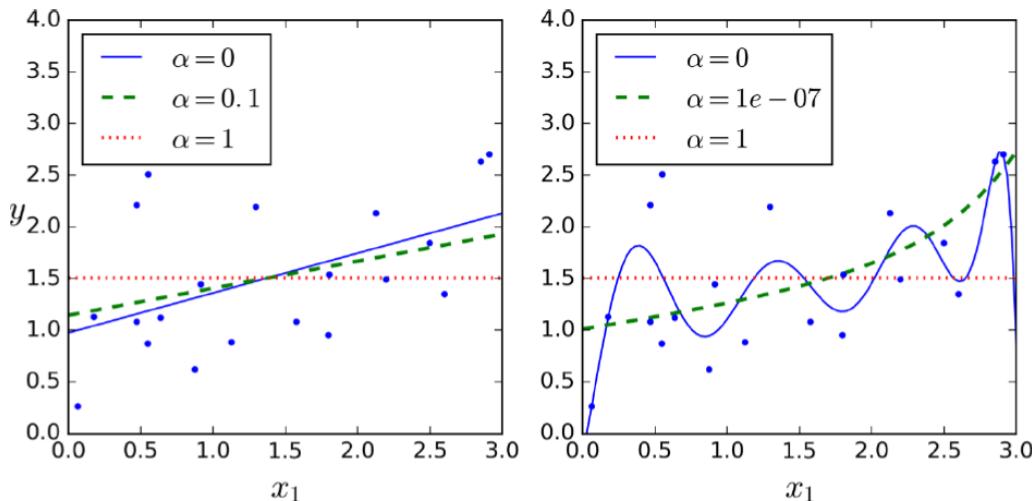
È un'altra versione regolarizzata della regressione lineare.

Aggiunge un termine di regolarizzazione alla funzione di costo, ma utilizza la ℓ_1 norm del vettore peso invece di metà del quadrato della ℓ_2 norm

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n |\theta_i|$$

La *Lasso Regression* tende ad eliminare completamente i pesi delle caratteristiche meno importanti.

Di seguito l'applicazione della Lasso Regression agli esempi precedenti:



Elastic Net

Elastic Net è una via di mezzo tra Ridge Regression e Lasso Regression.

Si può controllare il rapporto di combinazione “ r ”:

- quando $r=0$ Elastic Net equivale a Ridge regression
- quando $r=1$ è equivalente a Lasso Regression

$$J(\theta) = \text{MSE}(\theta) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2$$

In genere andrebbe evitata la semplice regressione lineare.

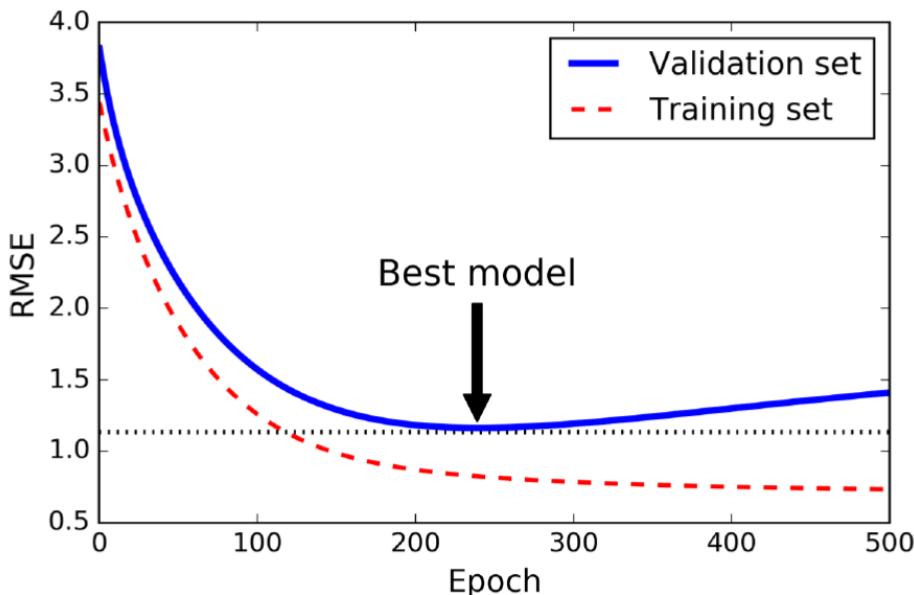
Ridge è una buona impostazione predefinita.

Lasso o Elastic Net se c'è il sospetto che solo alcune funzionalità siano effettivamente utili.

In generale, Elastic Net è preferita a Lazo, poiché Lazo può comportarsi in modo irregolare quando il numero di funzionalità è maggiore del numero di istanze di addestramento o quando più caratteristiche sono fortemente correlate.

Early Stopping

Un modo molto diverso per regolarizzare l'apprendimento di algoritmi come Gradient Descent è interrompere l'allenamento non appena l'errore di convalida raggiunge il minimo.



Dopo un po' l'errore di convalida smette di diminuire e inizia effettivamente a risalire.
Ciò indica che il modello ha iniziato un *overfitting* dei dati di training.

Logistic Regression

Alcuni algoritmi di regressione possono essere utilizzati per la classificazione e viceversa

La *Logistic Regression* è comunemente usata per stimare la probabilità che un'istanza appartenga a una classe particolare.

Se la probabilità stimata è maggiore di 50% allora il modello prevede che l'istanza appartiene a quella classe.

Questo lo rende un classificatore binario.

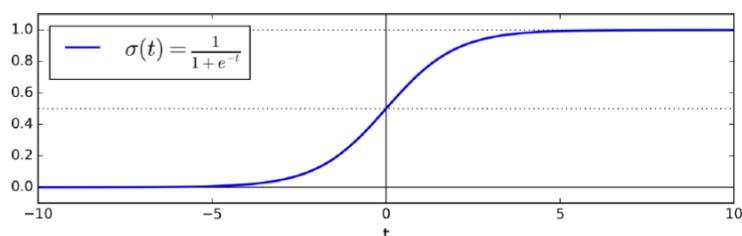
Il modello di regressione logistica calcola una somma ponderata delle caratteristiche di input più un termine di bias.

$$\hat{p} = h_{\theta}(x) = \sigma(\theta^T \cdot x)$$

La logistica detta anche *logit*, annotata con “ σ ” è una funzione sigmoidale che restituisce un numero compreso tra 0 e 1.

È definita come mostrato di seguito

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Un modello di regressione logistica prevede 1 se $\theta(T) \cdot x$ è positivo, e 0 se è negativo.

La funzione di costo sull'intero training set è semplicemente il costo medio su tutte le istanze di formazione
Può essere scritto in una singola formula chiamata **log loss**:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)})]$$

Non esiste un'equazione in forma chiusa nota per calcolare il valore di θ che riduca al minimo questa funzione di costo.

Ma questa funzione di costo è convessa, quindi per il Gradient Descent (o qualsiasi altro algoritmo di ottimizzazione) è garantito trovare il minimo globale.

Softmax Regression

Il modello di regressione logistica può essere generalizzato per supportare più classi.

Quando viene fornita un'istanza "x", il modello di regressione Softmax calcola un punteggio $s(x)$ per ogni classe "K" quindi stima la probabilità di ciascuna classe applicando la funzione softmax ai punteggi.

L'equazione per calcolare $s(x)$ è:

$$s_k(x) = (\theta^{(k)})^T \cdot x$$

Una volta calcolato il punteggio di ogni classe per l'istanza x può stimare la probabilità $p(k)$ che l'istanza appartenga alla classe "k" eseguendo i punteggi attraverso la funzione softmax:

$$\hat{p}_k = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_{j=1}^K \exp(s_j(x))}$$

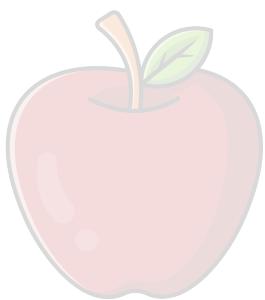
- ▶ K is the number of classes
- ▶ $s(x)$ is a vector containing the scores of each class for the instance x
- ▶ $\sigma(s(x))_k$ is the estimated probability that the instance x belongs to class k given the scores of each class for the instance

Il classificatore *Softmax Regression* prevede la classe con la più alta probabilità stimata:

$$\hat{y} = \operatorname{argmax}_k \sigma(s(x))_k = \operatorname{argmax}_k s_k(x) = \operatorname{argmax}_k ((\theta^{(k)})^T \cdot x)$$

La funzione di costo è la **cross entropy** che penalizza il modello quando stima una bassa probabilità per una classe target

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(\hat{p}_k^{(i)})$$

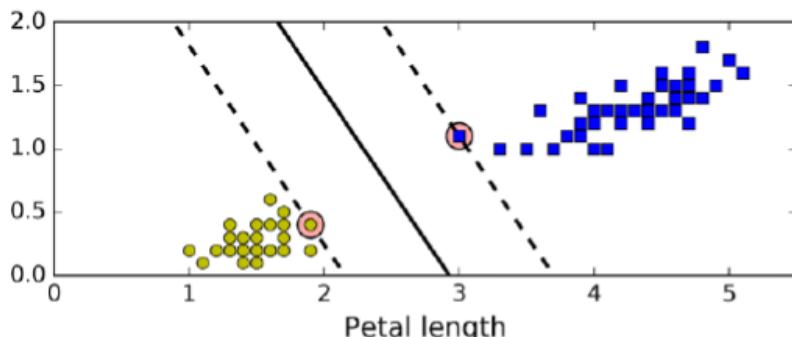


CoScienze
Associazione

SVM (Support Vector Machine)

Una SVM è un modello di ML capace di performare classificazioni, regressioni e detection di valori anomali in maniera lineare e/o non lineare.

In particolare, però, le SVM sono molto utili per questi compiti ma solo per dataset di dati piccoli/medi.



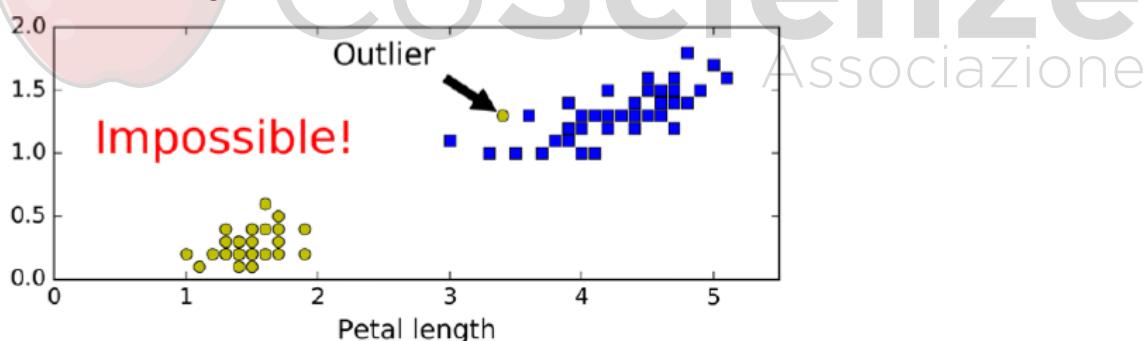
Un classificatore SVM cerca di separare le classi all'interno dei dati in modo da una "strada" larga il più possibile che separi queste classi.

Questa tecnica è chiamata **large margin classification**.

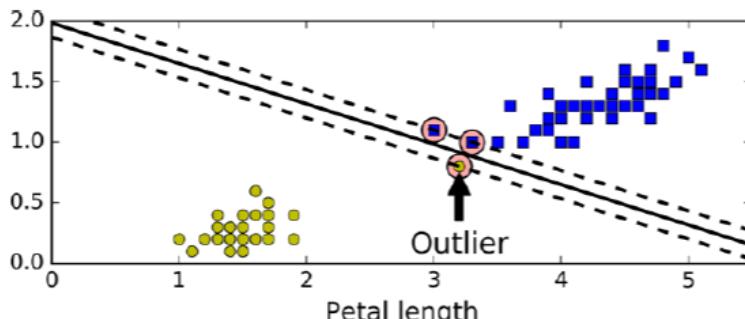
Gli elementi evidenziati nella figura, sono i **support vectors**, e fanno parte del bordo delle due classi.

Esistono due tipi di classificazioni possibili:

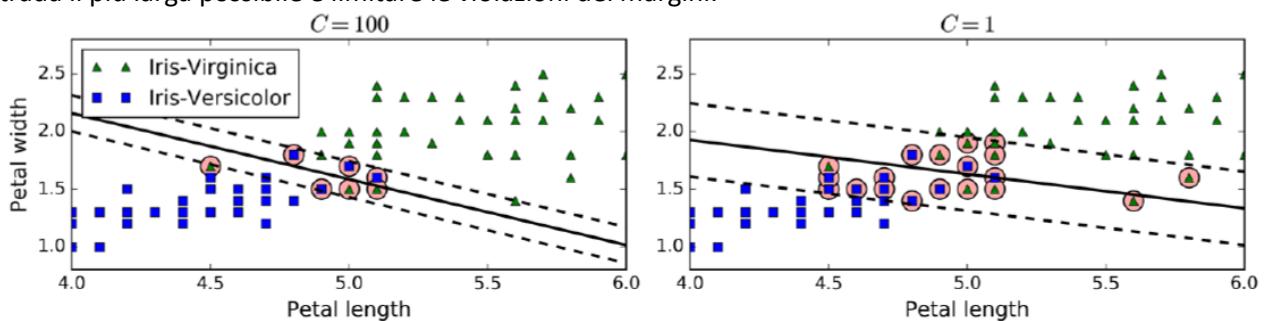
- **Hard margin Classification:** si impone alla SVM che tutte le istanze delle classi debbano essere "fuori dalla strada", quindi all'esterno delle righe tratteggiate. Questo ovviamente presuppone che i dati siano linearmente separabili, cosa non sempre possibile, come nel caso seguente.



Inoltre, anche se questo fosse possibile, potremmo trovarci un caso nel quale la strada creata dalla SVM è molto ristretta, sintomo del fatto che il modello non generalizzerà in maniera corretta i risultati, come nel caso seguente.



- **Soft margin Classification:** consiste nel mantenere un buon bilanciamento tra il mantenere la strada il più larga possibile e limitare le violazioni dei margini.

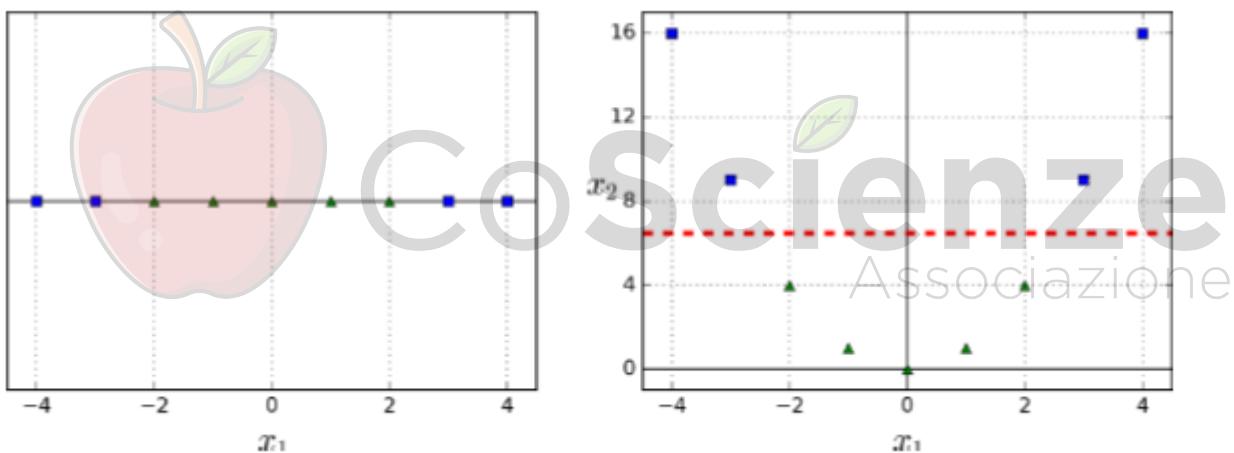


È possibile controllare questo bilanciamento mediante il parametro “ C ” in Scikit-Learn’s. Un valore piccolo renderà la strada più ampia aumentando le violazioni e viceversa.

Non Linear Classification

Molti dataset di dati non sono linearmente separabili. Per gestire questi dati è possibile aggiungere più *features*, come le polinomiali, per renderli linearmente separabili.

Nel seguente caso, ad esempio, se aggiungiamo una seconda feature $x_2 = x_1^2$ il dato diventa linearmente separabile



Per aggiungere features in Scikit-Learn possiamo creare una *Pipeline* con *PolynomialFeatures* seguito da *StandardScaler* e *LinearSVC*

```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

polynomial_svm_clf = Pipeline([
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scaler", StandardScaler()),
    ("svm_clf", LinearSVC(C=10, loss="hinge"))
])

polynomial_svm_clf.fit(X, y)
```

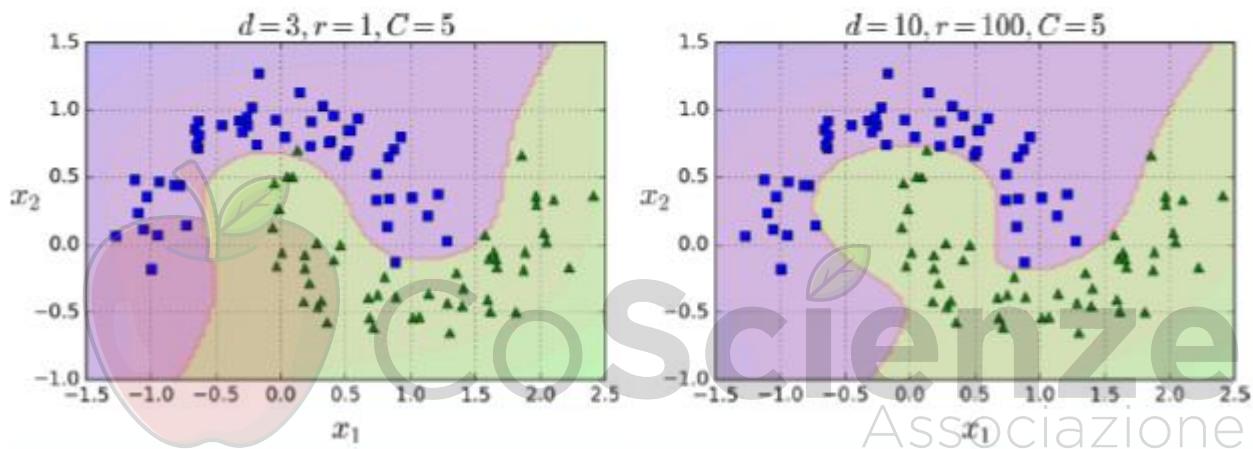
Ovviamente, un polinomio di grado basso non può gestire dataset molto complessi, mentre un polinomio di grado alto può rallentare di molto il modello.

Per ovviare a questo, fortunatamente, esiste un miracolo matematico chiamato **kernel trick**.

Con il *kernel trick* è possibile ottenere lo stesso risultato dell'utilizzare un polinomio anche di grado molto alto senza il bisogno di aggiungerlo al modello. Inoltre, non c'è l'espansione del numero di features poiché in pratica non si stanno aggiungendo altre features.

```
from sklearn.svm import SVC
poly_kernel_svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
])
poly_kernel_svm_clf.fit(X, y)
```

Di seguito due esempi con un kernel di terzo grado e di decimo grado:



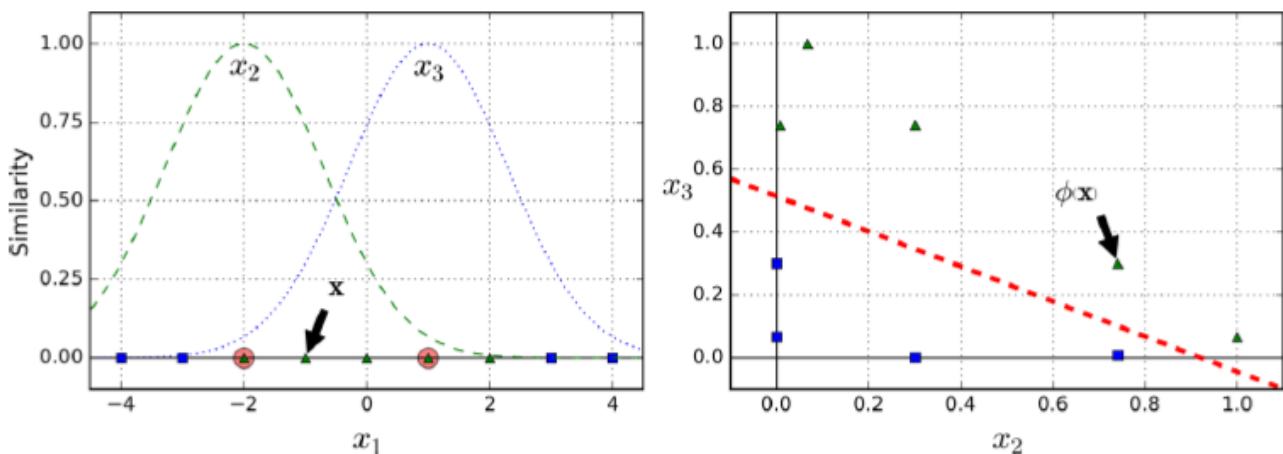
Un'altra tecnica per i problemi non lineari è utilizzare una **funzione di similarità**, che misuri quanto ogni istanza sia simile ad un *landmark*.

L'approccio più semplice consiste nel creare un *landmark* per ogni istanza del dataset.

Questo aumenta le possibilità che i dati diventino linearmente separabili, ma trasforma un training set con “ m ” istanze ed “ n ” features in un training set con “ m ” istanze ed “ m ” features, quindi rendendolo più complesso e lento.

Nel seguente caso, ad esempio, sono stati aggiunti due landmarks in $x_1=-2$ e $x_1=1$ ed è stata utilizzata la funzione di similarità *Gaussian Radial Basis Function (RBF)*:

$$\phi_\gamma(x, \ell) = \exp(-\gamma||x - \ell||^2)$$

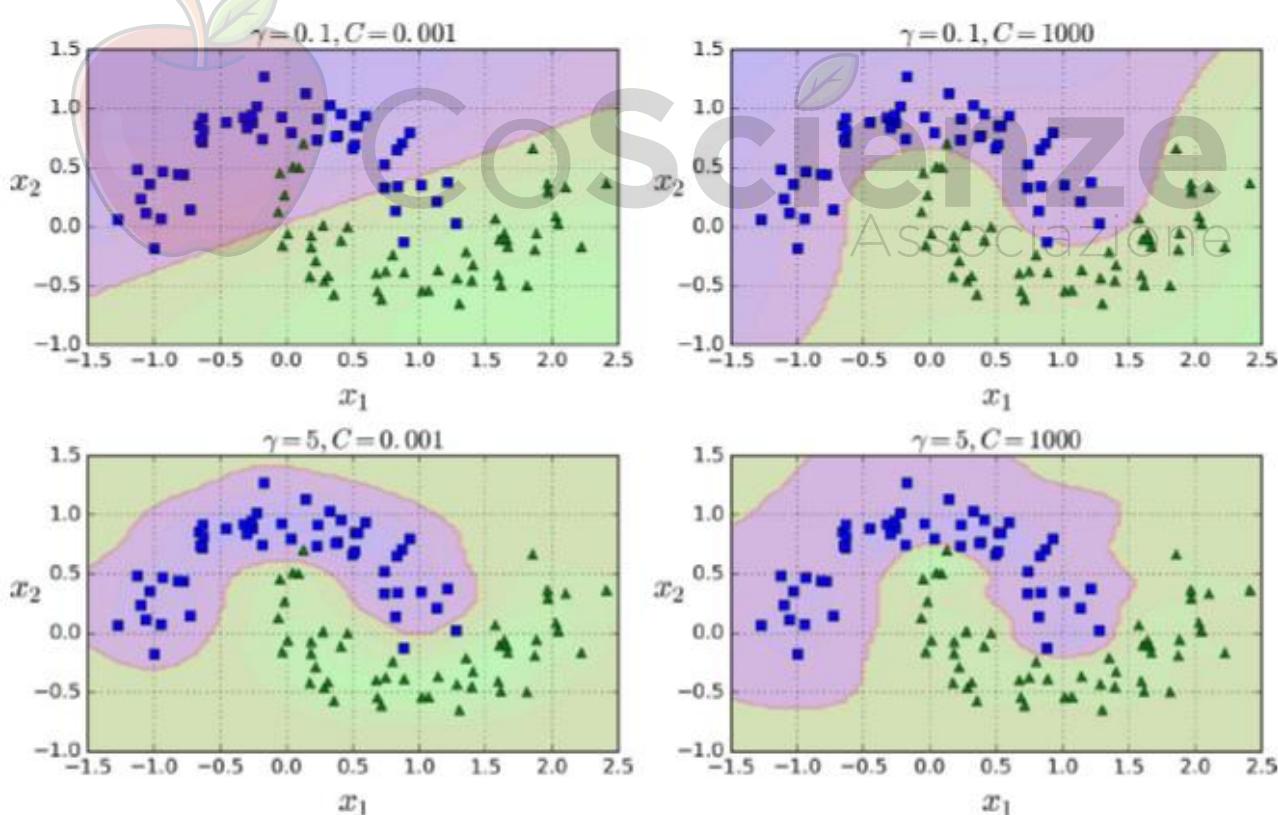


Come si può vedere, diventa linearmente separabile a valle della trasformazione mediante RBF.

In ogni caso, come per il kernel trick, anche in questo caso è possibile ottenere gli stessi risultati senza aggiungere le similarity features, mediante SVM.

```
#Define Pipeline
rbf_kernel_svm_clf = Pipeline([
    ("scaler", StandardScaler()), ("svm_clf", SVC(kernel="rbf", gamma=5, C=1000))])
```

Di seguito un confronto variando i valori di C ed y

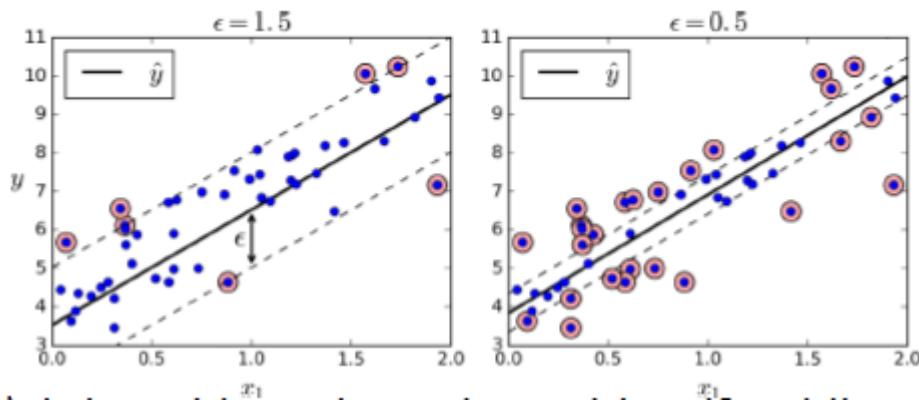


La classe *LinearSVC* basata sulla libreria *liblinear* implementa un algoritmo ottimizzato per SVM che scala linearmente con le istanze di training e le features, con complessità $O(m \cdot n)$, ma che però non supporta il kernel trick.

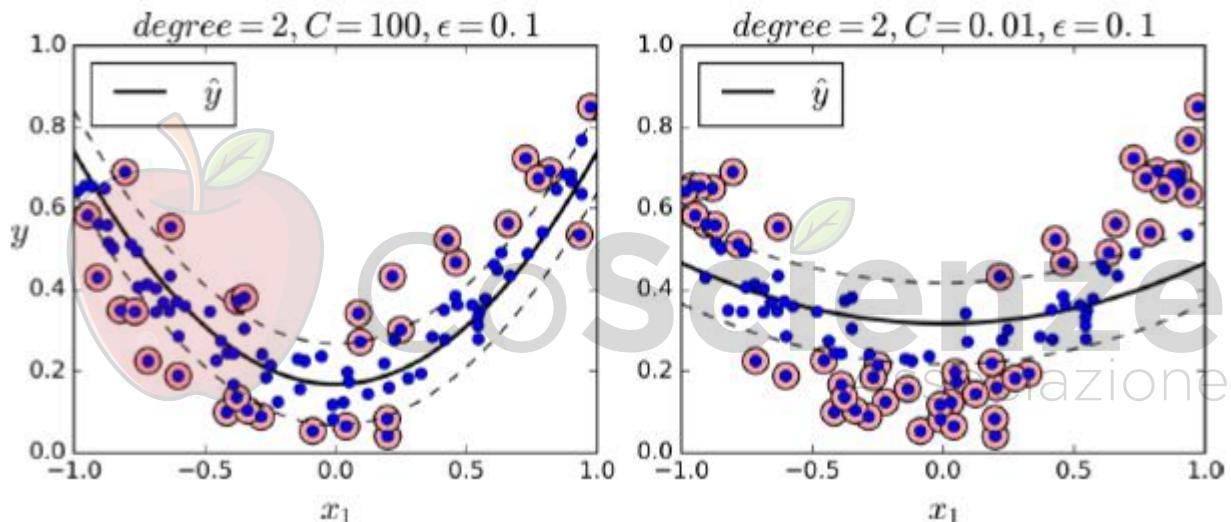
Invece, la classe *SVC* basata sulla libreria *libsvm* implementa un algoritmo di complessità compresa tra $O(m^2 \cdot n)$ e $O(m^3 \cdot n)$ ma che permette il kernel trick.

SVM Regression

Anche in questo caso si cerca un compromesso tra larghezza della strada e violazioni di questa, mediante l'iperparametro ϵ .



Per gestire regressioni non lineari, utilizziamo SVM con kernel



Le classi utilizzate sono *SVR*, equivalente a *SVC*, ed *LinearSVR* equivalente a *LinearSVC*.

Decision Function and Predictions

Un classificatore SVM predice le classi elaborando la funzione di decisione

$$\mathbf{w}^T \cdot \mathbf{x} + b = w_1 x_1 + \dots + w_n x_n + b$$

Se il risultato è positivo, allora y sarà 1, viceversa y sarà 0

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T \cdot \mathbf{x} + b < 0, \\ 1 & \text{if } \mathbf{w}^T \cdot \mathbf{x} + b \geq 0 \end{cases}$$

Training Objective

Effettuare il training di una SVM significa trovare " w " e " b " tali che il margine sia il più largo possibile per evitare violazioni o limitarle.

Più piccolo è il vettore dei pesi “ w ”, più largo è il margine.
Quindi l’obiettivo è quello di minimizzare $\|w\|$.

Questo implica che un classificatore *hard margin SVM* può essere espresso come un problema di *constrained optimization*

minimize: $\frac{1}{2} \cdot w^T \cdot w$
 w, b

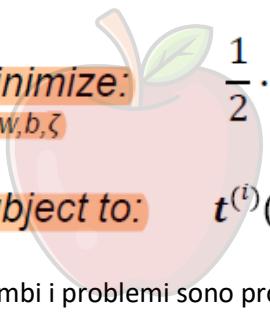
subject to: $t^{(i)}(w^T \cdot x^{(i)} + b) \geq 1 \text{ for } i = 1, 2, \dots, m$

Si noti che nella formula precedente si cerca di ridurre $\frac{1}{2} w^T \cdot w$, che equivale a $\frac{1}{2} \|w\|^2$, anziché ridurre al minimo $\|w\|$.

Questo perché darà lo stesso risultato, ma il termine al quadrato ha una derivata definita e semplice cioè w . Gli algoritmi di ottimizzazione funzionano molto meglio su funzioni differenziabili.

Mentre per un classificatore *soft margin SVM*, è necessario introdurre una *slack variable* $\zeta^{(i)} \geq 0$ che misura quanto all’i-esima istanza è permesso violare il margine.

Quindi il problema può essere espresso nel seguente modo:



minimize: $\frac{1}{2} \cdot w^T \cdot w + C \sum_{i=1}^m \zeta^{(i)}$
 w, b, ζ

subject to: $t^{(i)}(w^T \cdot x^{(i)} + b) \geq 1 - \zeta^{(i)} \text{ and } \zeta^{(i)} \geq 0 \text{ for } i = 1, 2, \dots, m$

Entrambi i problemi sono problemi di ottimizzazione con vincoli lineari, anche conosciuti come **Quadratic Programming (QP) problems**.

La formulazione generale di questi problemi è la seguente:

- ▶ The general problem formulation is

minimize: $\frac{1}{2} \cdot p^T \cdot H \cdot p + f^T \cdot p$
 p

subject to: $A \cdot p \leq b$

- ▶ where

- ▶ p is an n_p – dimensional vector (n_p = number of parameters)
- ▶ H is an $n_p \times n_p$ matrix
- ▶ f is an n_p – dimensional vector
- ▶ A is $n_c \times n_p$ matrix (n_c = number of constraints)
- ▶ b is an n_c – dimensional vector

Dual Problem

In genere, dato un problema di ottimizzazione vincolato, chiamato **primal problem**, è possibile esprimere un altro problema differente ma correlato, chiamato **dual problem**.

La cosa importante è che entrambi i problemi, sotto determinate condizioni, hanno la stessa soluzione, quindi risolverne uno comporta la risoluzione dell'altro.

Nel caso delle SVM, questa condizione è soddisfatta, quindi risultano avere la stessa soluzione.

Ad esempio, in questo caso, una volta che si trova il vettore “ $\alpha(0)$ ” che minimizza questa equazione nel *dual problem*:

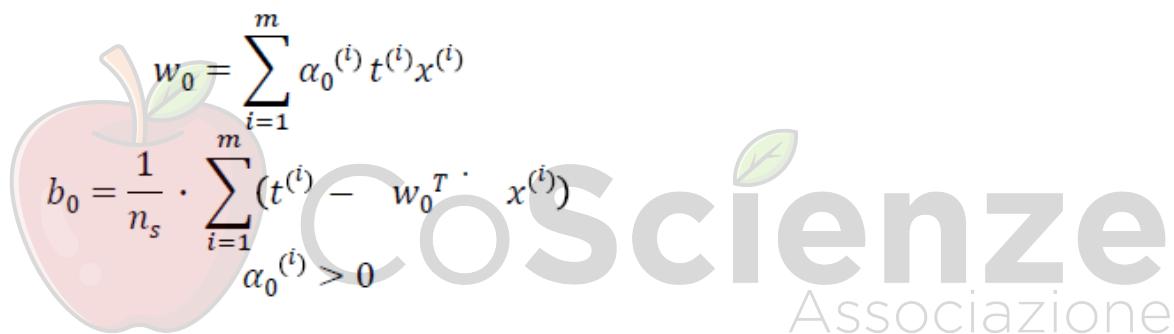
Dual form of the linear SVM objective

$$\underset{\alpha}{\text{minimize}} \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \mathbf{x}^{(i)T} \cdot \mathbf{x}^{(j)} - \sum_{i=1}^m \alpha^{(i)}$$

subject to $\alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \dots, m$

È possibile computare “ w ” e “ b ” che minimizzano il *primal problem*:

From the dual solution to the primal solution



$$w_0 = \sum_{i=1}^m \alpha_0^{(i)} t^{(i)} x^{(i)}$$

$$b_0 = \frac{1}{n_s} \cdot \sum_{i=1}^m (t^{(i)} - w_0^T \cdot x^{(i)})$$

$\alpha_0^{(i)} > 0$

In questo caso, il dual problem è più veloce da risolvere quando le istanze di training sono inferiori al numero di features. Inoltre, nel dual problem il kernel trick è possibile mentre nel primal non è possibile.

Online SVMs

Ricordando che *online learning* significa allenare in maniera incrementale un modello, per i classificatori SVM un metodo possibile è usare il *Gradient Descent (SGDClassifier)*, che minimizza il costo della funzione seguente, che è praticamente derivata dal primal problem

Linear SVM classifier cost function

$$J(\mathbf{w}, b) = \frac{1}{2} \mathbf{w}^T \cdot \mathbf{w} + C \sum_{i=1}^m \max(0, 1 - t^{(i)} (\mathbf{w}^T \cdot \mathbf{x}^{(i)} + b))$$

Sfortunatamente questa converge in maniera molto più lenta rispetto al metodo basato su QP.

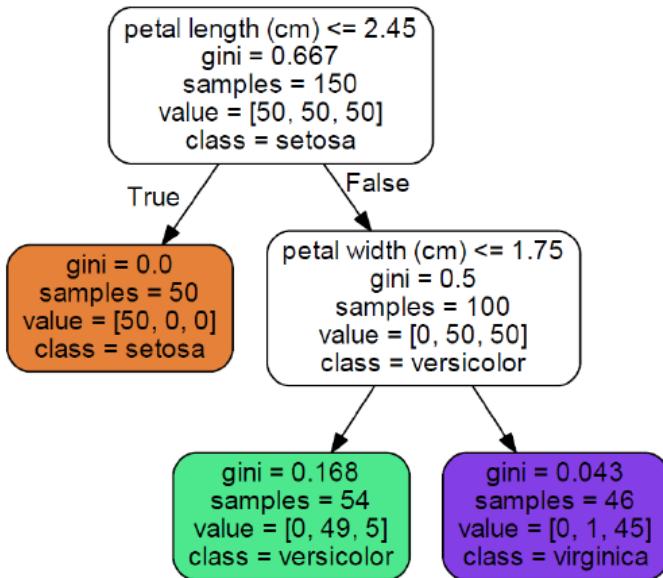
La prima somma $J(\mathbf{w}, b)$ spinge il modello ad avere pesi “ w ” più piccoli per avere un margine più largo.

La seconda somma computa il numero totale di violazioni del margine.

Minimizzare queste assicura che il margine sia il più ampio possibile e che le violazioni siano minime.

Decision Tree

Al pari delle SVM, gli Alberi di decisione rappresentano una soluzione versatile di ML capace di preformare task di classificazione, regressione e multi-output.



Per predire un risultato, si parte dalla radice al livello 0 e ci si muove in basso lungo i rami fino ad arrivare ad una foglia dell'albero in base ai predicati dei singoli nodi.

Nel caso in cui, nella foglia siano presenti più classi, la classificazione fa fede alla classe con probabilità più alta.

Ogni nodo è caratterizzato da diversi attributi:

- *Samples*: il numero di istanze che appartengono a quel nodo;
- *Value*: i valori delle istanze presenti nel nodo;
- *Gini*: una misura di impurità, se 0 indica che tutte le istanze appartengono alla stessa classe.

L'indice Gini si misura con la seguente formula:

$$G_i = 1 - \sum_{k=1}^n p_{i,k}^2$$

Un altro modo di misurare tramite indice Gini uno split è dato dalla seguente formula:

$$gini_{split}(T) = (n_1/n) gini(T_1) + (n_2/n) gini(T_2)$$

Dati “N” booleani, è possibile definire 2^{2^n} differenti alberi di decisione.

Decision Tree Learning

L'idea per la costruzione di un albero di decisione “ottimale” è quella di scegliere ricorsivamente l'attributo più significativo come radice del sottoalbero al passo k .

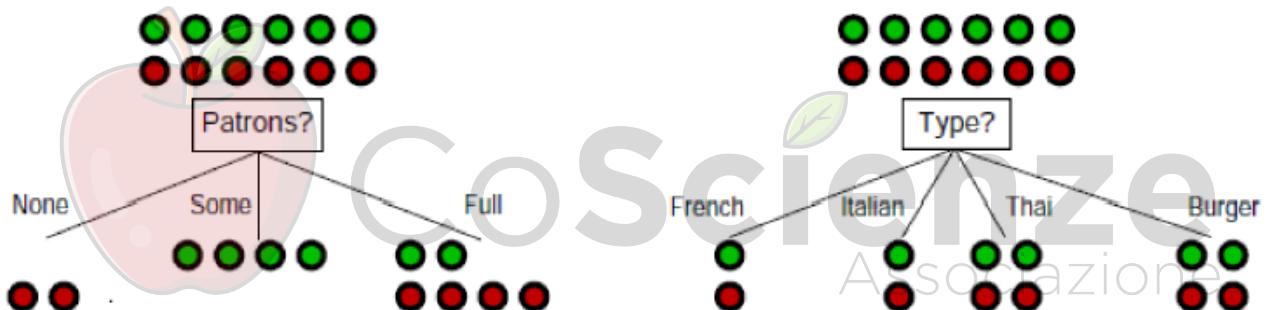
```

function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return Majority-Value(examples)
  else
    best  $\leftarrow$  Choose-Attribute(attributes, examples)
    tree  $\leftarrow$  a new decision tree with root test best
    for each value vi of best do
      examplesi {elements of examples with best = vi}
      subtree  $\leftarrow$  DTL(examplesi, attributes - best, Majority-Value(examples))
      add a branch to tree with label vi and subtree subtree
  return tree

```

Uno split è ottimale, quando si isolano al meglio tutti i casi positivi oppure tutti i casi negativi rispetto al predicato.

Nel caso seguente, lo split migliore è il primo.



Un'altra misura di impurità per i nodi dell'albero è l'**entropia**. Come per l'indice Gini anche in questo caso l'obiettivo è minimizzare questo valore.

$$H(\langle P_1; \dots; P_n \rangle) = - \sum_{i=1}^n P_i \log_2 P_i$$

Scikit-Learn's usa l'algoritmo *Classification And Regression Tree (CART)* per la creazione di un albero di decisione.

Inizialmente splitta il training set in due subset usando una feature "k" ed una soglia "t(k)". Seleziona questa coppia tra tutte le coppie possibili cercando di minimizzare la seguente funzione di costo, che misura l'impurità del sottoalbero sinistro e destro:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

L'algoritmo si ferma quando raggiunge il valore del parametro `max_depth` oppure fin quando non riesce a ridurre l'impurità.

Il problema di trovare l'albero ottimale è **NP-Completo** ed ha una complessità di $O(e^m)$.

Split Predicates

Uno split può essere:

- Binario;
- Multiplo

Per gli attributi numerici:

- Binary split: $A \leq v, A > v$
- Multiple split: $A \leq v_1, v_1 < A \leq v_2, \dots, v_{n-1} < A \leq v_n$

Per gli attributi categorici:

- Binary split: $A \in S', A \in S - S'$
- Multiple split: $A \in S_1, \dots, A \in S_n$

Computational Complexity

Generalmente gli alberi di decisione sono bilanciati, quindi attraversarli costa **$O(\log m)$** .

In ogni caso, però, l'algoritmo di training compara tutte le features su tutti gli esempi in ogni nodo, e quindi costa **$O(n * \log m)$**

Regularization Hyperparameters

Dato che l'albero di decisione è un modello, chiamato *nonparametric model*, vale a dire che la struttura del modello è libera di adattarsi ai dati, per evitare l'overfitting c'è bisogno di restringere e limitare la libertà dell'algoritmo.

Generalmente si limita la profondità massima dell'albero valorizzando il parametro `max_depth`.

```
#Application of the Decision Trees Regressor
tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X, y)
```

Decision Trees Regression

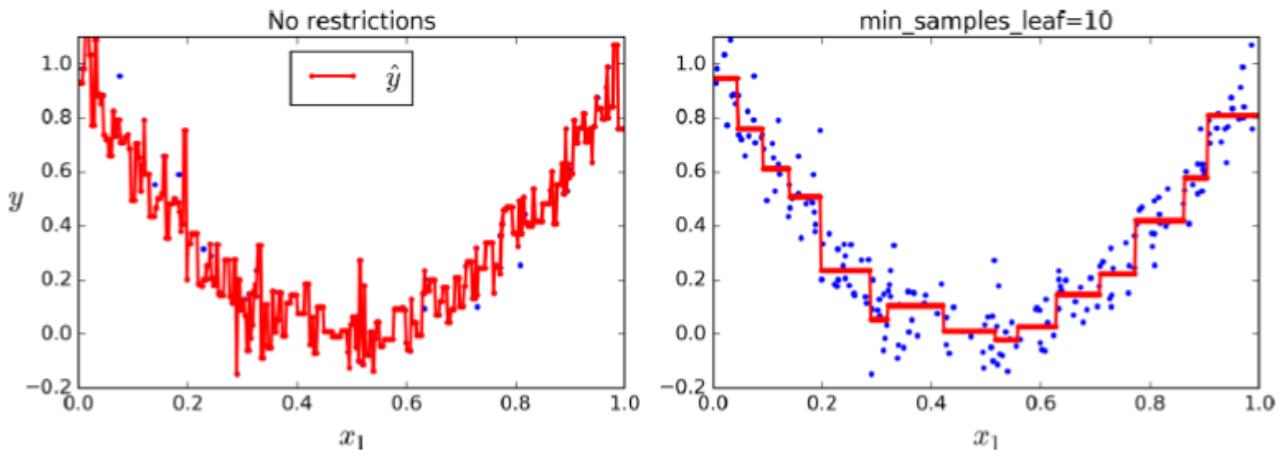
La differenza principale rispetto all'albero di decisione per la classificazione è che piuttosto che predire una classe deve predire un valore.

Questo valore non è altro che la media degli elementi che sono presenti nella foglia.

In questo caso il CART algoritmo piuttosto che minimizzare l'impurità, minimizza l'MSE:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}} \quad \text{where} \quad \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Anche in questo caso si effettua una regolarizzazione dei parametri per limitare la libertà dell'algoritmo per evitare l'overfitting, come l'utilizzo di *min_samples_leaf*.

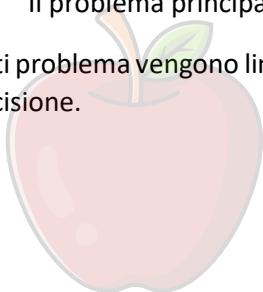


Instability

Gli alberi di decisione hanno delle limitazioni:

- Amano i confini decisionali ortogonali e sono sensibili alle rotazioni nel training set;
- Il problema principale è che sono molto sensibili a piccole variazioni nel training set

Questi problemi vengono limitati con le **Random Forest** che effettuano una media sui risultati di singoli alberi di decisione.



CoScienze
Associazione

Random Forest

Se aggreghiamo le previsioni di un gruppo di predittori (come classificatori o regressori), spesso miglioriamo le previsioni rispetto al miglior preditore individuale.

Il gruppo di predittori è chiamato **ensemble** e la tecnica è chiamata **ensemble learning**.

Si può allenare un gruppo di alberi di decisione ognuno su un differente sotto insieme del training set. Un insieme del genere è chiamato **random forest**.

Utilizzando l'*ensemble learning* è possibile utilizzare due tecniche per la decisione finale:

- **Hard voting**: si classifica basandosi sulla maggioranza tra le classificazioni dei singoli predittori;
- **Soft voting**: si effettua una media pesata tra le classificazioni dei singoli predittori.

In questo modo, anche un classificatore debole può dare un contributo utile per la costruzione di un *ensemble* forte.

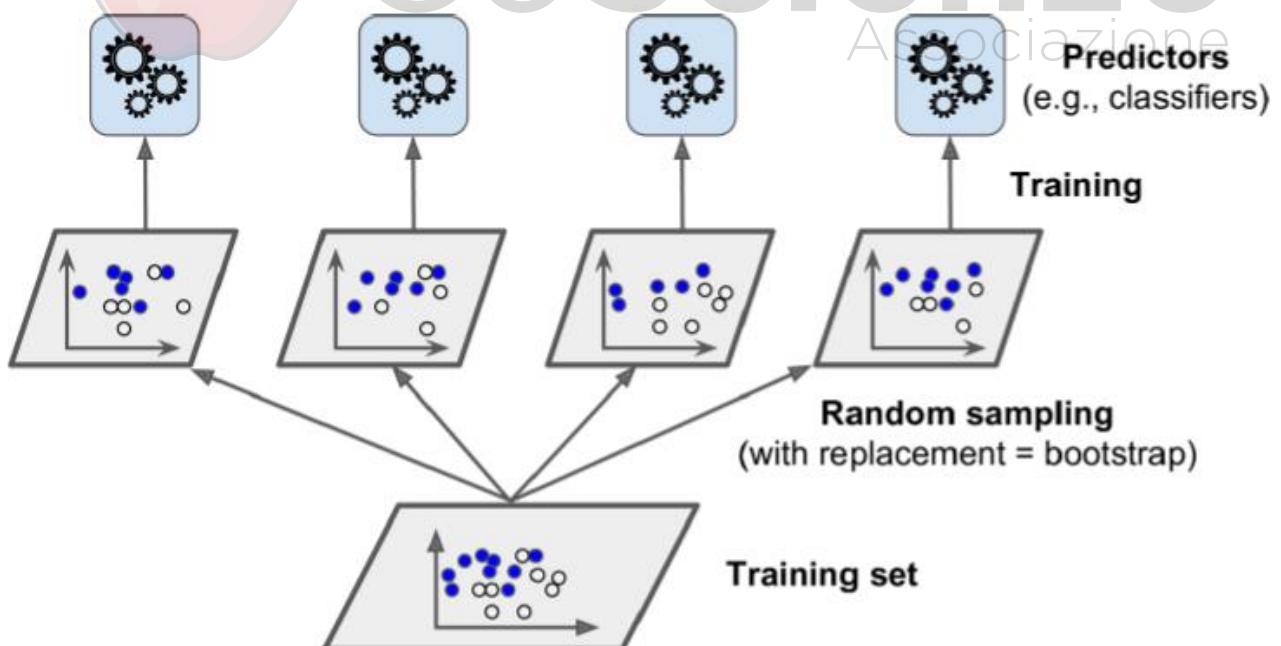
Questo però, è vero solo se i classificatori sono perfettamente indipendenti tra loro, in modo che facciano errori non correlati, cosa che chiaramente non può essere vera dato che vengono allenati sullo stesso training set.

Bagging and Pasting

Consideriamo il caso in cui utilizziamo lo stesso algoritmo di training per ogni preditore, ma ognuno di essi utilizza un sotto insieme del training set scelto in modo casuale.

Quando la scelta del campione è fatta con sostituzione, cioè permettendo la scelta delle stesse istanze più volte, il metodo si chiama **bagging**.

Quando la scelta del campione è fatta senza sostituzione, cioè non permettendo la scelta delle stesse istanze più volte, il metodo si chiama **pasting**.

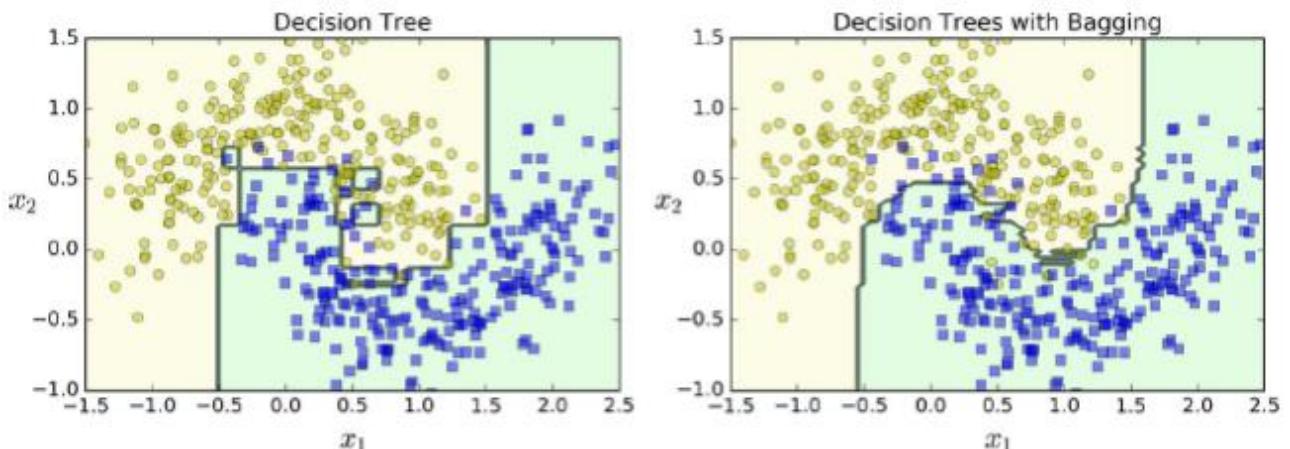


Uno dei vantaggi di queste tecniche è che i predittori possono essere allenati in parallelo, utilizzando diverse CPU.

Inoltre, generalmente il risultato è che l'ensemble ha un bias simile a quello di un singolo preditore ma una varianza minore.

Le successive linee di codice mostrano l'implementazione di 500 alberi di decisione, ognuno allenato su 100 istanze di training con bagging:

```
#Define BaggingClassifier
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
```



Per eseguire il training con *pasting* basta sostituire `bootstrap=true` con `bootstrap=false`.

Il bootstrapping (campionamento con sostituzione) produce un po' più di diversità, quindi il bagging produce un bias leggermente più alto del pasting.

Anche se, in generale, il bagging ha risultati migliori del pasting.

Tuttavia, quando si effettua il campionamento delle istanze di training, solitamente una parte di questi dati non è minimamente considerata.

È stato dimostrato che il 37% dei dati rientra in questa categoria, ed è chiamato **out-of-bag**.

Dato che questi dati non vengono considerati dai predittori, si possono utilizzare nel processo di validazione di un modello senza utilizzare tecniche come la cross validation.

Per fare questo basta settare a `true` il parametro `oob_score` in Scikit-Learn's nella classe `BaggingClassifier`.

```
#Define BaggingClassifier with oob_score=True
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, oob_score=True)
```

Random Patches

Effettuare il campionamento sia delle istanze di training che delle features è chiamato metodo **Random Patches**.

Mantenere le istanze di training interamente e campionare l'insieme delle features è chiamato metodo **Random Subspaces**.

Random Forest

Una *Random Forest* è un insieme di alberi di decisione.

È possibile utilizzare la classe `RandomForestClassifier` piuttosto che utilizzare `BaggingClassifier` e passargli `DecisionTreeClassifier`, dato che è più ottimizzata.

```
#Define Random Forest Classifier
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16,
n_jobs=-1, random_state=42)
```

È possibile, inoltre, misurare l'importanza di ogni feature mediante il parametro *feature_importances*. Questo parametro misura l'importanza di una caratteristica nell'abbassare l'impurità all'interno degli alberi della foresta.

```
#Feature evaluations
for name, score in zip(iris["feature_names"],
rnd_clf.feature_importances_):
    print(name, score)
sepal length (cm) 0.11249225099876374
sepal width (cm) 0.023119288282510326
petal length (cm) 0.44103046436395765
petal width (cm) 0.4233579963547681
```

Extra Trees

L'algoritmo introduce casualità nella costruzione degli alberi di decisione, dato che, piuttosto che cercare il miglior attributo per lo splitting, lo cerca tra un insieme casuale di attributi.

È possibile, inoltre, introdurre ulteriore casualità usando soglie casuali per ogni attributo.

Una foresta del genere è chiamata **Extremely Randomized Trees Ensemble** oppure **Extra Trees**.

Il vantaggio è che allenare un'insieme del genere è molto più veloce rispetto alle *Random Forest* ed inoltre si ottien un bias più alto rispetto però ad una varianza più bassa.

Boosting

Il *boosting* si riferisce ai metodi ensemble che combinano diversi predittori deboli in uno forte.

I metodi più importanti sono:

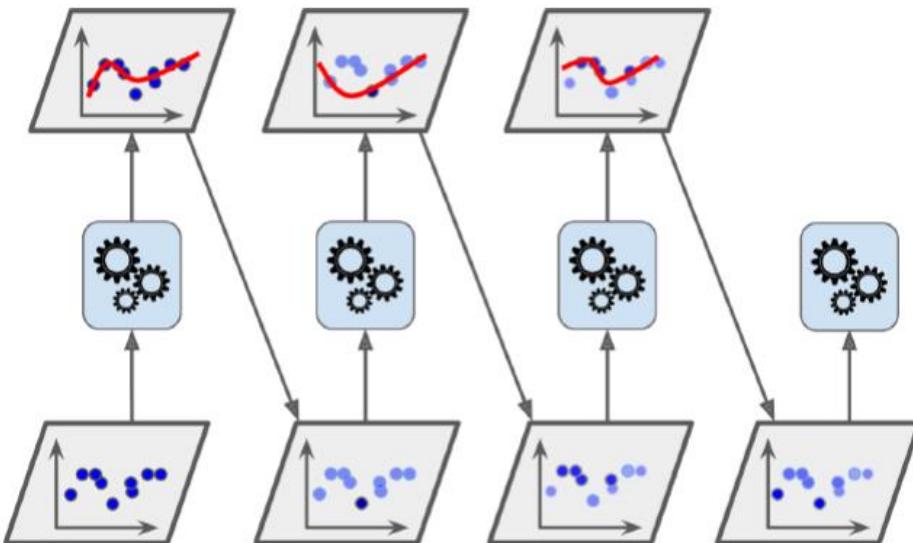
- AdaBoost
- Gradient Boosting

Ada Boost

Un modo in cui un nuovo predittore corregge il suo predecessore è farlo prestando un po' più di attenzione alle istanze di training per le quali il predecessore ha avuto come risultato un *underfit*.

Ad esempio, per creare un classificatore AdaBoost abbiamo le seguenti fasi:

1. Viene addestrato e utilizzato un classificatore di base (come un albero decisionale) per fare previsioni sul set di training.
2. Il peso relativo delle istanze di addestramento classificate erroneamente viene quindi aumentato.
3. Un secondo classificatore viene addestrato utilizzando i pesi aggiornati e di nuovo fa previsioni sul set di training, i pesi vengono aggiornati e così via.



Questo metodo è simile al *Gradient Descent*, con la differenza che in questo caso si aggiungono predittori all'insieme migliorati gradualmente, mentre in quel caso si modificano i parametri di un predittore per migliorarne i risultati.

Scikit-Learn usa la classe *SAMME* per AdaBoost ed una variante chiamata *SAMME.R*.

Gradient Boosting

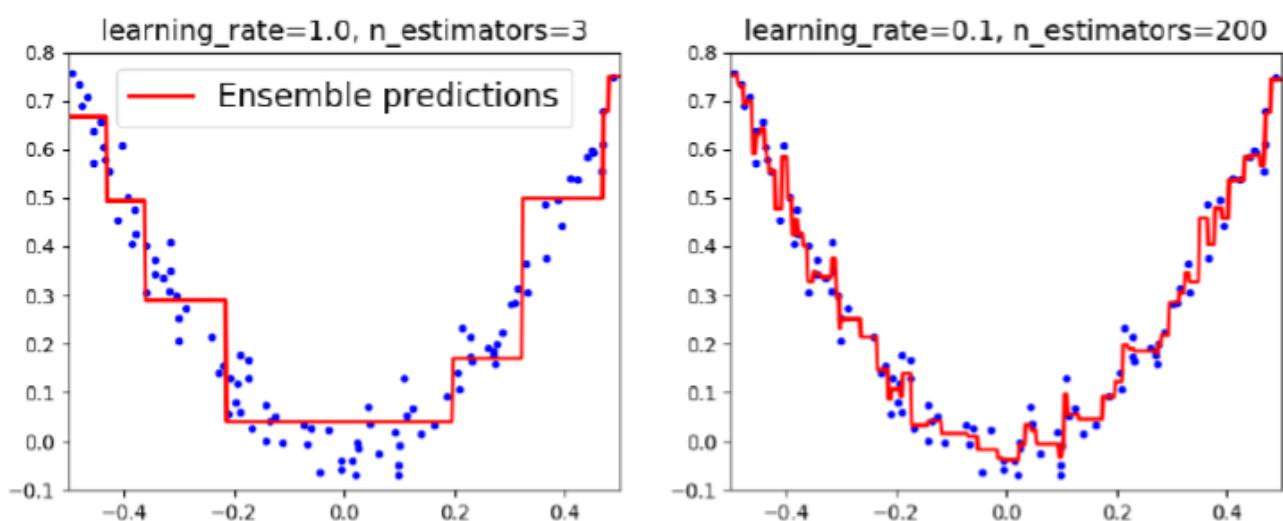
Invece di modificare i pesi dell'istanza in ogni iterazione come fa AdaBoost, questo metodo cerca di adattare un nuovo predittore agli errori residui commessi dal precedente.

In Scikit-Learn è presente la classe *GradientBoostingRegressor*.

Di seguito un confronto tra due regressori del genere:

```
#Define GradientBoostingRegressor classifier
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0, random_state=42)

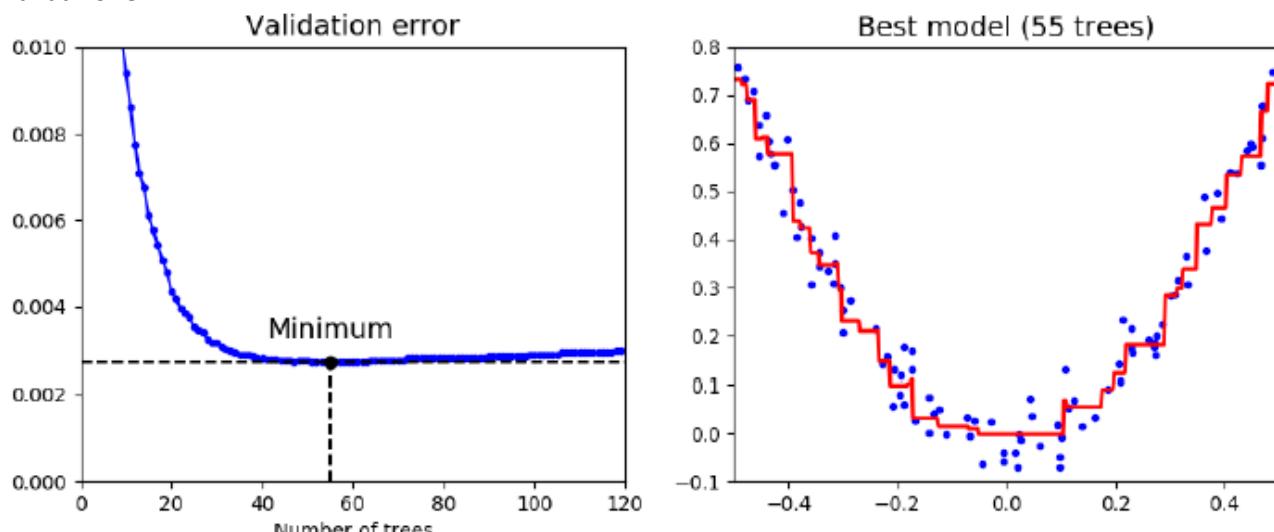
#Define GradientBoostingRegressor classifier
gbrt_slow = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.1, random_state=42)
```



Il parametro *learning_rate* se settato ad un valore basso, come nell'esempio destro, ha bisogno di molti più alberi per adattarsi alle istanze di training, ma il risultato è generalmente migliore.

Questa regolarizzazione è chiamata **shrinkage**.

Per trovare il miglior numero di alberi è possibile utilizzare l'*early stopping* ed analizzare l'errore di validazione.



La classe `GradientBoostingRegressor` supporta anche un parametro di campionamento.

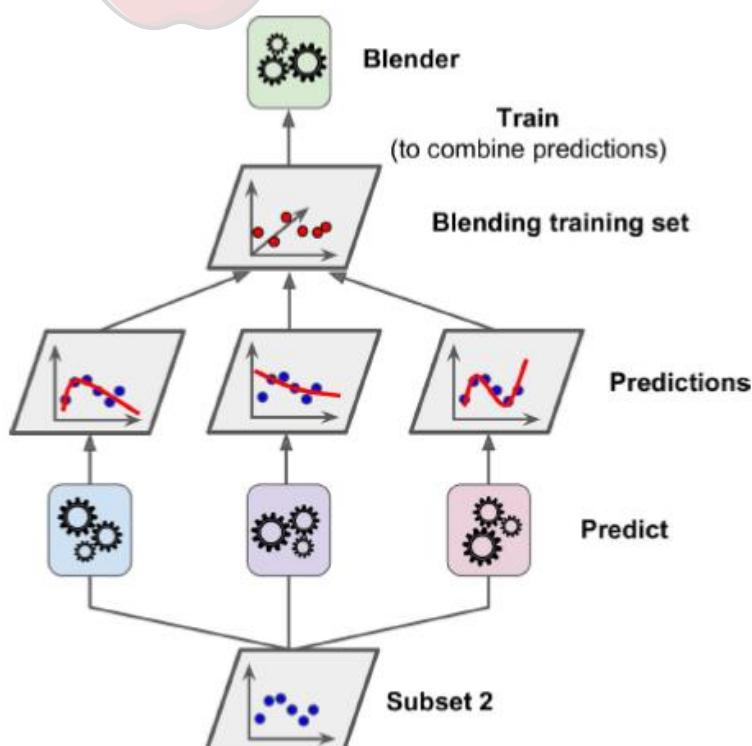
Ad esempio, se il valore è pari a 0.25, ogni albero sarà allenato sul 25% delle istanze di training scelte in modo casuale.

Questa tecnica è chiamata **Stochastic Gradient Boosting**.

Stacking

In questo metodo, piuttosto che effettuare un calcolo sugli output dei singoli predittori, si utilizza un predittore finale, chiamato **blender**, che prende in input le predizioni ed effettua una decisione finale.

È possibile definire più livelli in questo grafo, e generalmente si dividono le istanze di training in diversi sottoinsiemi per allenare sia i predittori che il blender.



Dimensionality Reduction

Molti problemi di Machine Learning coinvolgono migliaia o addirittura milioni di features per ogni istanza di training.

Questo problema è spesso indicato come **maledizione della dimensionalità (curse of dimensionality)**.

La riduzione della dimensionalità fa perdere alcune informazioni.

Inoltre, la riduzione può filtrare un po' di rumore e quindi ottenere prestazioni più elevate.

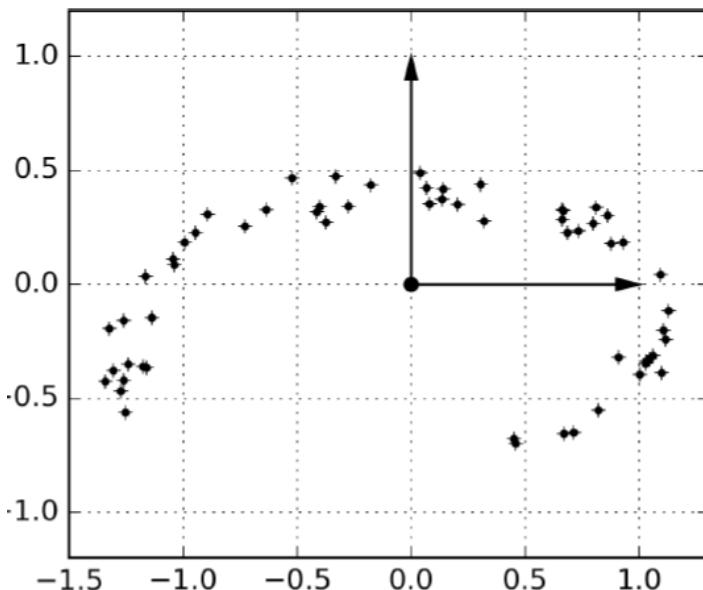
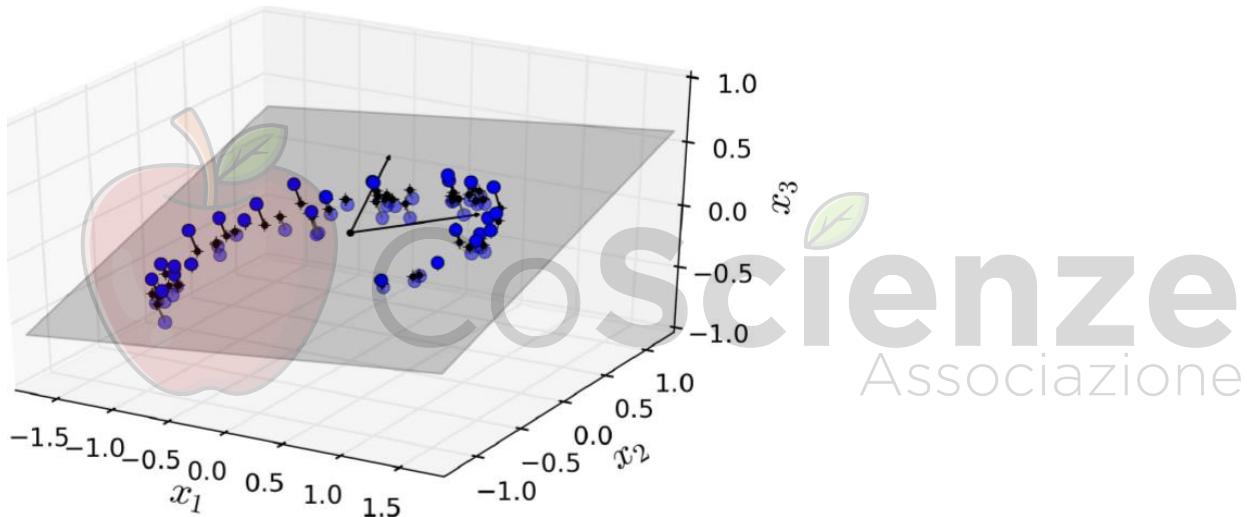
Esistono due approcci principali per la riduzione della dimensionalità:

- *Proiezione;*
- *Manifold Learning*

Projection

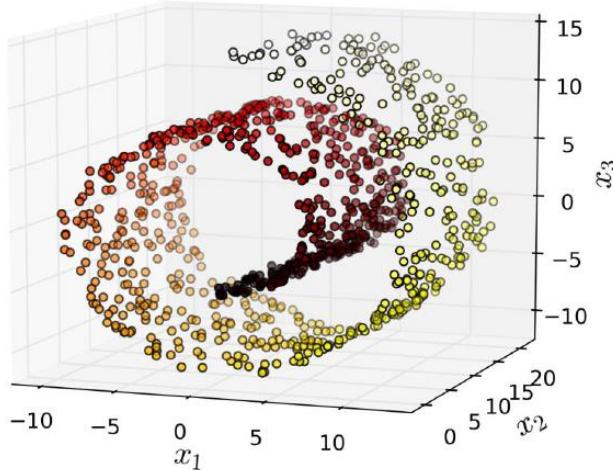
Nella maggior parte dei problemi del mondo reale le istanze di training non sono distribuite uniformemente su tutte le dimensioni.

Di conseguenza, tutte le istanze di training si trovano effettivamente all'interno di (o vicino a) un sottospazio dimensionale molto più basso dello spazio cui appartengono.

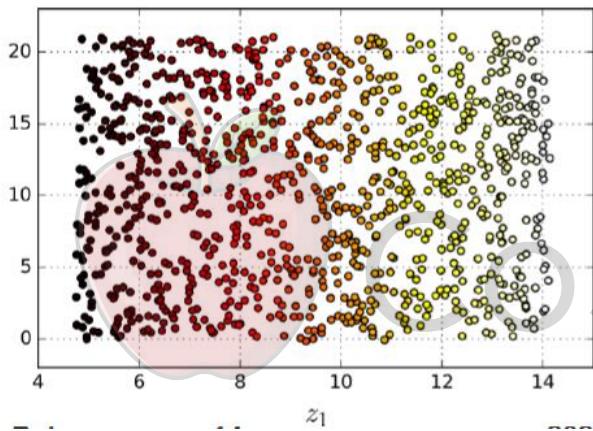


Manifold Learning

In molti casi, però, il sottospazio può ruotare e girare, come ad esempio nel famoso dataset **Swiss Roll**



In questi casi, quello che si vuole realmente ottenere, è srotolare questo *roll* per rappresentare i dati in due dimensioni



Scienze
Associazione

Questo è un esempio di **manifold**.

Più in generale, un manifold “ d ” dimensionale fa parte di uno spazio “ n ” dimensionale ($d < n$) che localmente assomiglia a un iperpiano “ d ” dimensionale.

Nel caso dello *Swiss Roll* $n=3$ e $d=2$.

Molti algoritmi di riduzione della dimensionalità funzionano modellando il *manifold* su cui le istanze giacciono.

Questo processo è chiamato **Manifold Learning**.

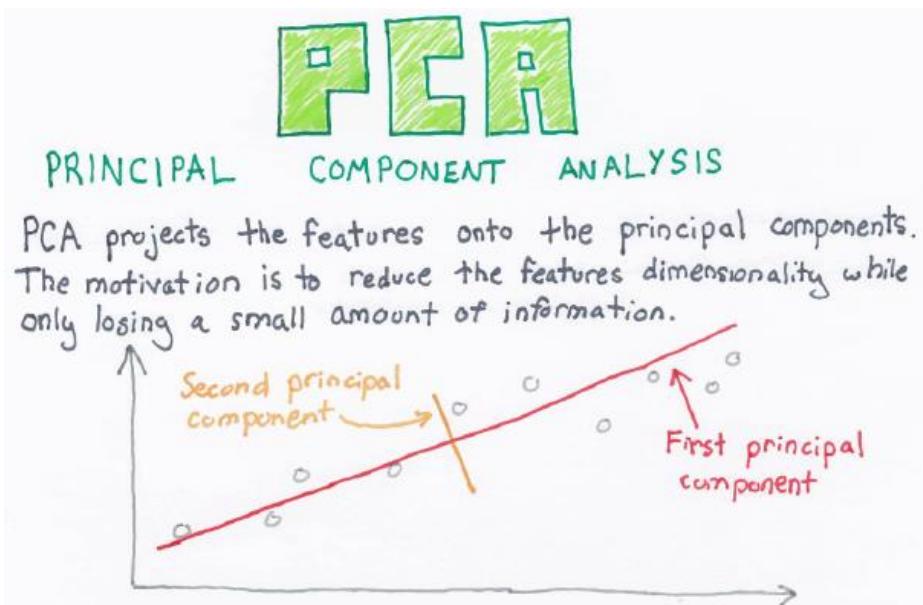
Pensa al set di dati MNIST. Tutte le immagini delle cifre scritte a mano hanno alcune somiglianze.

Sono fatte di linee collegate, i confini sono bianchi, sono più o meno centrati e così via.

Questi vincoli tendono a comprimere il set di dati in un valore dimensionale inferiore.

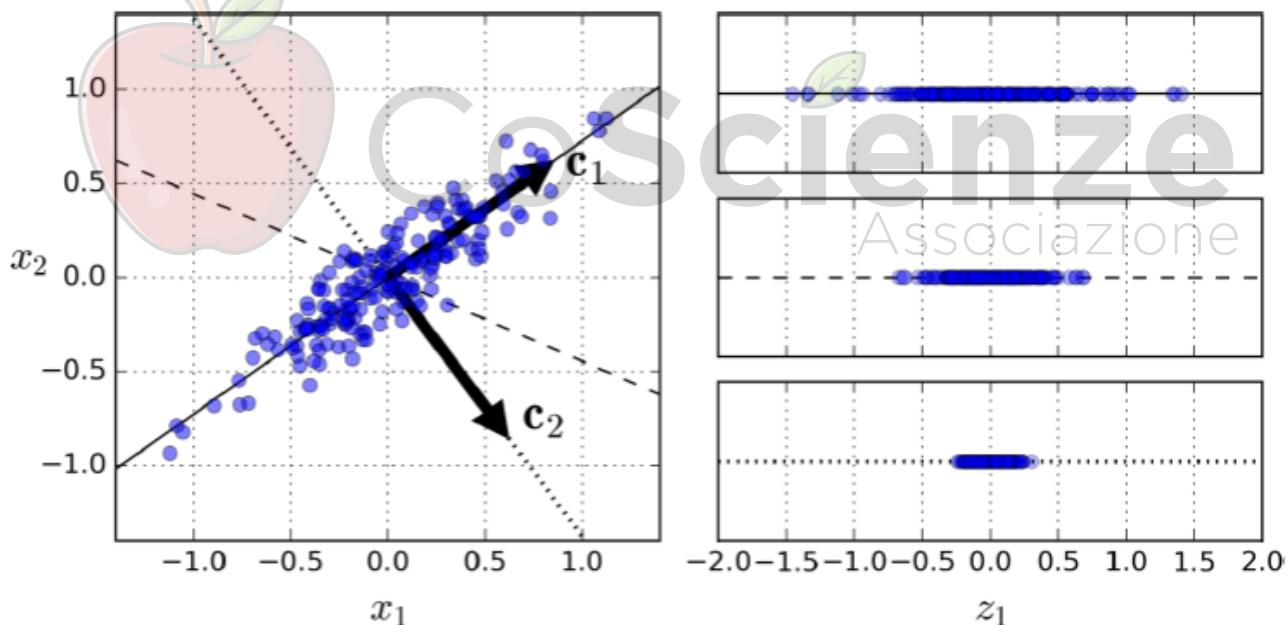
PCA (Principal Component Analysis)

Innanzitutto, identifica l'iperpiano più vicino ai dati. Quindi proietta i dati su di esso.



Come si può vedere:

- la proiezione sulla linea continua conserva la massima varianza
- la proiezione sulla linea punteggiata conserva pochissima varianza
- la proiezione sulla linea tratteggiata conserva un intermedia quantità di varianza



In pratica, PCA identifica l'asse che rappresenta la quantità maggiore di varianza nel training set (la linea continua). Trova anche un secondo asse, ortogonale al primo, quello rappresenta l'importo maggiore della varianza rimanente (linea tratteggiata) e così via fino a trovare N assi.

Il vettore unitario che definisce l'i-esimo asse è chiamato *i-esima componente principale*.

Esiste una tecnica standard di fattorizzazione della matrice chiamata **Singular Value Decomposition (SVD)**. Può scomporre la matrice del training set X nel prodotto di tre matrici $U \cdot \Sigma \cdot V^T$, dove V contiene tutti i componenti principali che stiamo cercando.

A questo punto, basta semplicemente moltiplicare la matrice X per il vettore delle componenti principali.

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X} \cdot \mathbf{W}_d$$

Un esempio di implementazione in Scikit-Learn:

```
#PCA using Scikit-Learn
pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

Le componenti principali possono essere viste attraverso l'attributo `components_`.

Inoltre, l'attributo `explained_variance_ratio_` mostra l'informazione che porta ciascuna componente.

```
print("explained_variance:\n{}".format(pca.explained_variance_ratio_))
      explained_variance:
      [0.84248607 0.14631839]
```

Invece di specificare il numero di entità, è possibile impostare `n_components` come un float compreso tra 0.0 e 1.0, indicando il rapporto di varianza che si desidera preservare

```
#PCA using Scikit-Learn
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
print("pca.n_components : {}".format(pca.n_components_))
```

Ovviamente dopo la riduzione della dimensionalità, il training set occupa molto meno spazio.

Infatti, ogni istanza avrà poco più di 150 caratteristiche, invece delle caratteristiche originali 784 se utilizziamo il dataset MNIST.

È possibile anche effettuare l'operazione inversa, per cercare di ritornare ai dati originali.

Ovviamente questo non restituirà esattamente i dati originali, poiché la proiezione ha perso un po' di informazioni.

La distanza media al quadrato tra i dati originali e i dati ricostruiti viene chiamata **reconstruction error**.

```
#PCA using Scikit-Learn
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

Equation 8-3. PCA inverse transformation, back to the original number of dimensions

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}} \cdot \mathbf{W}_d^T$$

Incremental PCA

È possibile suddividere il training set in mini batch e alimentare un algoritmo IPCA con un mini batch alla volta.

Questo è utile per set di training di grandi dimensioni e anche per applicare PCA online.

```
#IncrementalPCA using Scikit-Learn
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
    print(".", end="")
    inc_pca.partial_fit(X_batch)
X_reduced = inc_pca.transform(X_train)
```

Randomized PCA

Questo è un algoritmo stocastico che trova rapidamente un'approssimazione delle prime “ d ” componenti principali.

La sua complessità computazionale è $O(m * d^2) + O(d^3)$, invece di $O(m * n^2) + O(n^3)$, quindi è drammaticamente più veloce rispetto ai precedenti algoritmi quando “ d ” è molto più piccolo di “ n ”.

Di seguito presentiamo un codice di esempio

```
#Randomized PCA using Scikit-Learn
rnd_pca = PCA(n_components=154, svd_solver="randomized",
               random_state=42)
X_reduced = rnd_pca.fit_transform(X_train)
```

Kernel PCA

Nelle lezioni precedenti, abbiamo discusso del *kernel trick*, una tecnica matematica che mappa implicitamente le istanze in uno spazio dimensionale molto elevato (chiamato *feature space*), consentendo la classificazione e la regressione non lineare con SVM.

Lo stesso trucco può essere applicato alla PCA, rendendo possibile l'esecuzione di proiezioni non lineari per la riduzione della dimensionalità.

```
#KernelPCA using Scikit-Learn
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

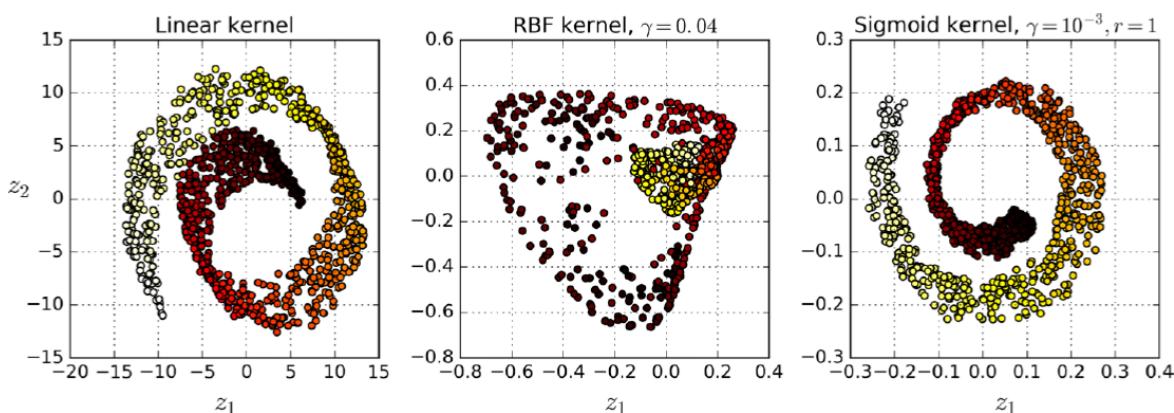


Figure 10. Swiss roll reduced to 2D using kPCA with various kernels

Tuttavia, la riduzione della dimensionalità è spesso una fase di preparazione per un compito di apprendimento supervisionato (ad es. Classificazione).

Ad esempio, il codice seguente crea innanzitutto una pipeline per ridurre la dimensionalità a due dimensioni usando kPCA, e quindi applicare la regressione logistica per la classificazione

```
clf = Pipeline([
    ("kPCA", KernelPCA(n_components=2)),
    ("log_reg", LogisticRegression(solver="liblinear"))
])
param_grid = [{ "kPCA__gamma": np.linspace(0.03, 0.05, 10),
    "kPCA__kernel": ["rbf", "sigmoid"] }]
grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

Un altro approccio, questa volta del tutto non supervisionato, è selezionare il kernel e iperparametri che producono l'errore di ricostruzione più basso.

La seguente figura mostra il set di dati 3D originale di Swiss roll (in alto a sinistra) e il set di dati 2D risultante dopo l'applicazione di kPCA utilizzando un kernel RBF (in alto a destra).

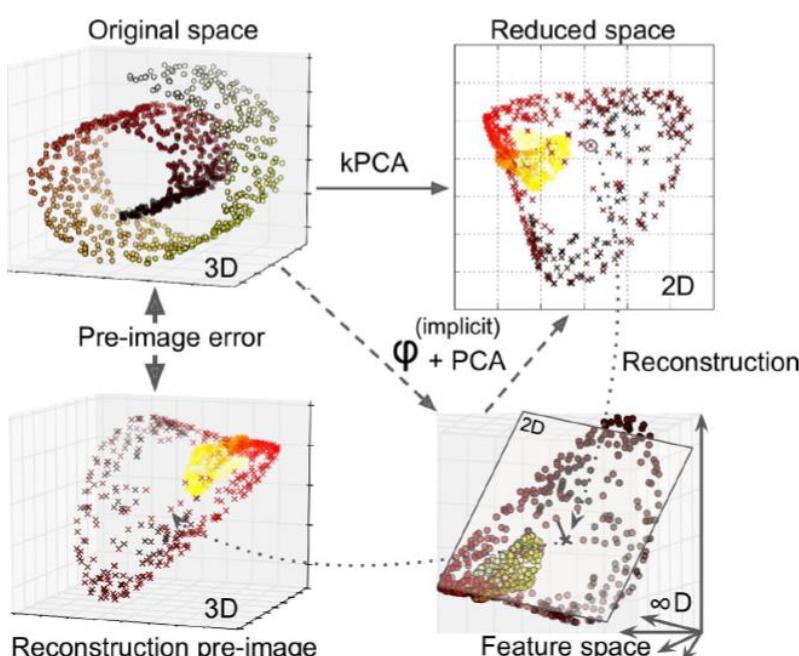
Grazie al trucco del kernel, questo è matematicamente equivalente a mappare il set di addestramento su uno spazio di funzionalità dimensionale infinito (in basso a destra) utilizzando la mappa delle caratteristiche ϕ , quindi proiettare il training set trasformato in 2D utilizzando PCA linear.

Si noti che se potessimo invertire il passaggio PCA linear per un dato esempio nello spazio ridotto, il punto ricostruito si troverebbe nello spazio delle caratteristiche, non nello spazio originale.

Poiché lo spazio delle caratteristiche è infinito di dimensione, non possiamo calcolare il punto ricostruito, e quindi non si è in grado di calcolare il vero errore di ricostruzione.

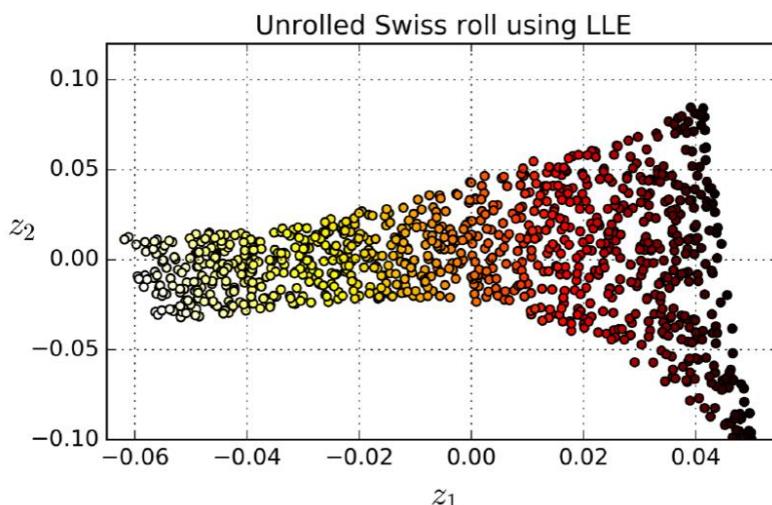
Fortunatamente, è possibile trovare un punto nello spazio originale che si trova vicino al punto ricostruito. Questa è chiamata ricostruzione pre-immagine. A questo punto è possibile misurare la distanza al quadrato tra il punto ricostruito e quello originale.

È quindi necessario selezionare il kernel e gli iperparametri che riducano al minimo questo errore di ricostruzione.



LLE (Locally Linear Embedding)

In poche parole, LLE funziona prima misurando come ogni istanza di training si riferisce linearmente ai suoi vicini e quindi ricerca una rappresentazione a bassa dimensione del training set dove queste relazioni locali sono meglio conservate. Questo lo prende particolarmente buono per lo srotolamento di twisted manifolds, specialmente quando non c'è troppo rumore. Tuttavia, le distanze non sono conservative su ampia scala: la parte sinistra dello Swiss roll srotolato viene schiacciata, mentre la parte destra è allungata.



```
#LLE using Scikit-Learn
```

```
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10,
random_state=42)
X_reduced = lle.fit_transform(X)
```

Il funzionamento di LLE: in primo luogo, per ogni istanza di training $x(i)$, l'algoritmo identifica i suoi "k" vicini più prossimi (in codice precedente $k = 10$), quindi tenta di ricostruire " x " come funzione lineare di questi vicini.

la somma dei pesi

Più precisamente, trova i pesi " $w(i,j)$ " tali che la distanza al quadrato tra $x(i)$ e $\sum_1^m w_{i,j} * x^j$ sia la più piccola possibile.

Equation 8-4. LLE step 1: linearly modeling local relationships

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^m \left(\mathbf{x}^{(i)} - \sum_{j=1}^m w_{i,j} \mathbf{x}^{(j)} \right)^2$$

subject to
$$\begin{cases} w_{i,j} = 0 & \text{if } \mathbf{x}^{(j)} \text{ is not one of the } k \text{ c.n. of } \mathbf{x}^{(i)} \\ \sum_{j=1}^m w_{i,j} = 1 & \text{for } i = 1, 2, \dots, m \end{cases}$$

La matrice "W" adesso conterrà i pesi.

Ora il secondo passaggio consiste nel mappare le istanze di training in uno spazio dimensionale " d " (dove $d < n$) preservando per quanto possibile queste relazioni locali.

Se $z(i)$ è l'immagine di $x(i)$ in questo d spazio dimensionale, allora vogliamo la distanza al quadrato tra $z(i)$

somma dei $\sum_1^m W_{i,j} * z^j$ sia la più piccola possibile.

pesi * immagine

Equation 8-5. LLE step 2: reducing dimensionality while preserving relationships

$$\hat{Z} = \underset{Z}{\operatorname{argmin}} \sum_{i=1}^m \left(z^{(i)} - \sum_{j=1}^m \hat{w}_{i,j} z^{(j)} \right)^2$$

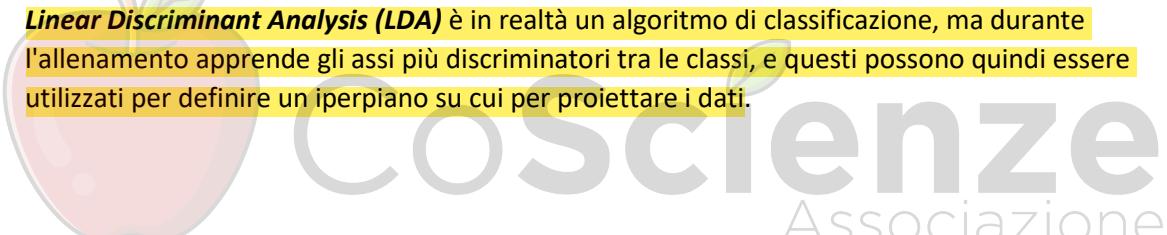
L'implementazione LLE di Scikit Learn ha quanto segue come complessità computazionale: $O(m \log m * n \log k)$ per la ricerca dei "k" vicini più prossimi, $O(mnk^3)$ per ottimizzare i pesi, e $O(dm^2)$ per costruire la rappresentazione nella dimensione ridotta

Sfortunatamente, m^2 nell'ultimo termine fa in modo che questo algoritmo è scarso a scalare quando un dataset è molto grande

Other

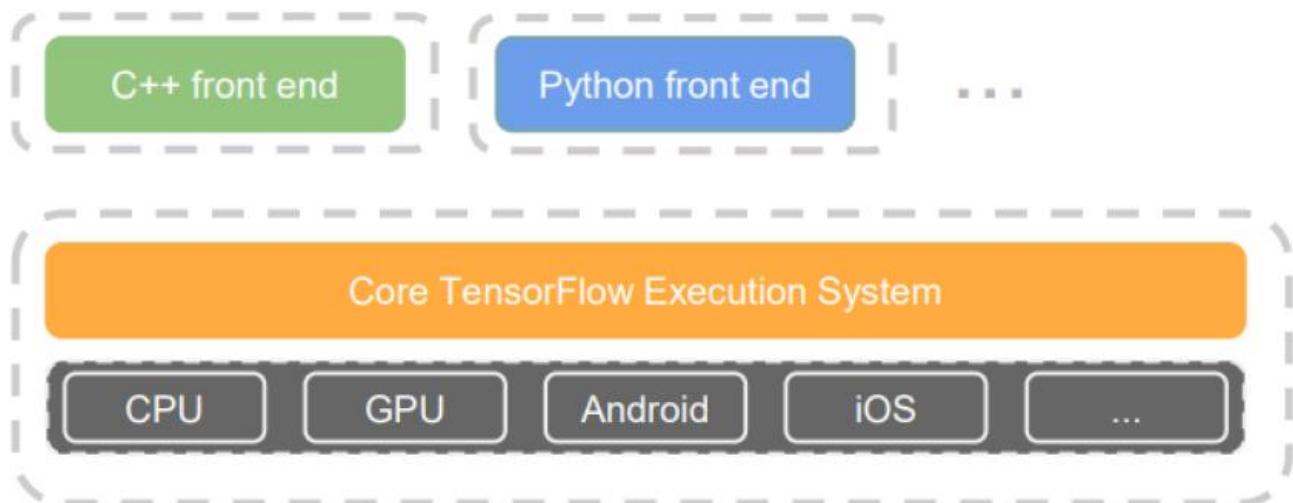
Altri algoritmi per la riduzione della dimensionalità sono:

- **Multidimensional Scaling (MDS)** riduce la dimensionalità cercando di preservare le distanze tra le istanze
- **Isomap** crea un grafo collegando ogni istanza ai vicini più vicini, quindi riduce la dimensionalità mentre cerca di preservare le distanze geodetiche tra le istanze cammino più breve tra due punti
- **t Distributed Stochastic Neighbor Embedding (t SNE)** riduce la dimensionalità mentre cerca di mantenere simili istanze vicine e istanze dissimili lontane
- **Linear Discriminant Analysis (LDA)** è in realtà un algoritmo di classificazione, ma durante l'allenamento apprende gli assi più discriminatori tra le classi, e questi possono quindi essere utilizzati per definire un iperpiano su cui per proiettare i dati.

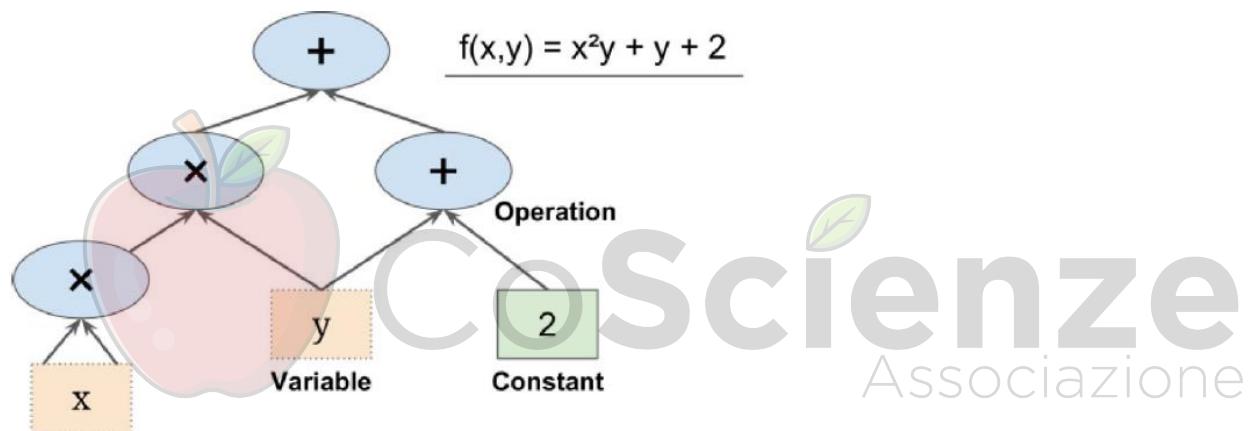


Tensorflow

Una libreria open source per il calcolo numerico utilizzando il flusso di dati grafici sviluppata da Google.



Il suo principio di base è definire prima un grafo di calcoli ed eseguirlo utilizzando codice C++ ottimizzato.



Vi sono API ad alto livello costruite su TensorFlow che permettono di modellare, allenare e valutare i modelli. Tra queste troviamo *Keras* o *Pretty Tensor*.

Tensorflow, inoltre, fornisce **TensorBoard**, uno strumento di visualizzazione che consente di sfogliare il grafico di calcolo, visualizza le curve di apprendimento, ecc.

L'elemento di base è il **tensores**.

Un tensore è un oggetto geometrico che mappa in modo multi lineare vettori, scalari e altri tensori a un tensore risultante.

A **vector** is a tensor ($f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f(e_i) = v_i$)

A **matrix** is a tensor ($f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$, $f(e_i, e_j) = A_{ij}$)

A **scalar** is a tensor ($f : \mathbb{R} \rightarrow \mathbb{R}$, $f(e_1) = c$)

Le operazioni sono chiamate **ops**. Costanti e variabili rappresentano le **source ops**. Gli input e gli output sono i **tensors**.

Dataflow Graph

Il flusso di dati è un modello di programmazione comune per il calcolo parallelo.

TensorFlow utilizza un grafico del flusso di dati per rappresentare il calcolo in termini di dipendenze tra operazioni individuali.

I vantaggi del dataflow sono:

- Parallelismo;
- Esecuzione distribuita;
- Velocità di Compilazione;
- Portabilità.

Il codice seguente crea solo un grafico di calcolo.

Per valutarlo è necessario aprire una sessione di TensorFlow e usarla per inizializzare le variabili e valutare f. La sessione esegue operazioni su dispositivi come CPU e GPU e mantiene i valori di queste operazioni.

```
import tensorflow as tf
x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
f = x*x*y + y + 2
```

Il seguente codice crea una sessione, inizializza variabili, valuta f e chiude il sessione liberando le risorse:

```
>>> sess = tf.Session()
>>> sess.run(x.initializer)
>>> sess.run(y.initializer)
>>> result = sess.run(f)
>>> print(result)
42
>>> sess.close()
```

O equivalentemente, in cui la sessione viene gestita e chiusa automaticamente:

```
with tf.Session() as sess:
    x.initializer.run()
    y.initializer.run()
    result = f.eval()
```

Per inizializzare le variabili in maniera collettiva, è possibile utilizzare

```
Init = tf.global_variables_initializer()
```

E poi

```
with tf.Session() as sess:
    init.run() # actually initialize all the variables
    result=f.eval()
```

Quando si valuta un nodo, TensorFlow determina i nodi da cui dipende e li valuta per primi. y dipende da x , che dipende da w , quindi prima TensorFlow valuta w , quindi x , quindi y e restituisce y . Poi, il codice esegue il grafico per valutare z .

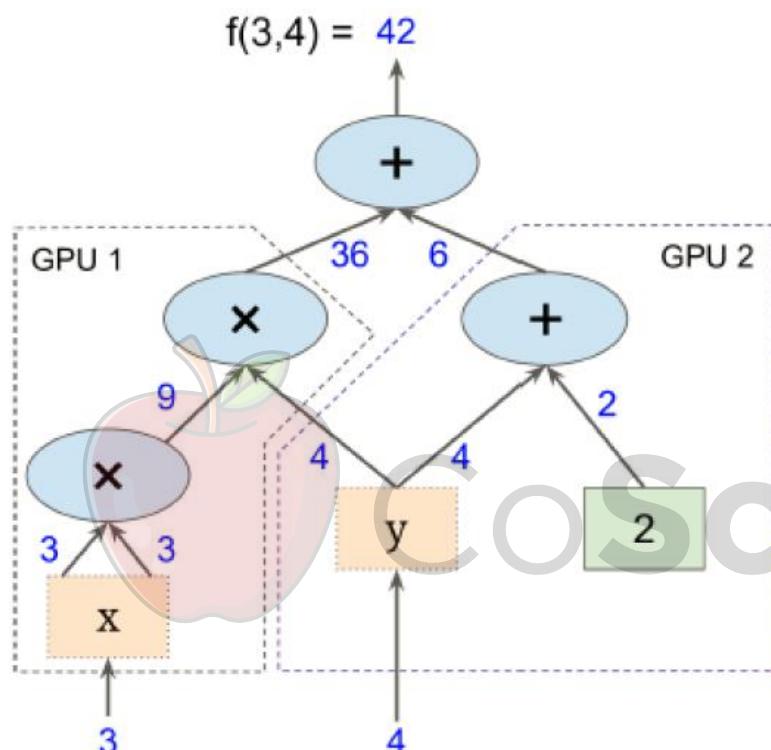
I valori vengono eliminati tra le esecuzioni del grafico, ad eccezione dei valori delle variabili, che vengono mantenute in tutte le esecuzioni del grafico.

Le costanti (`tf.constant`) sono tensori a valore fisso non addestrabili.

Le variabili (`tf.Variable`) sono tensori inizializzati in una sessione addestrabile

I segnaposto (`tf.Placeholder`) sono tensori di valori sconosciuti durante la costruzione del grafico, ma passato come input durante una sessione.

È possibile spezzare il grafico in più blocchi ed eseguirli in parallelo su più CPU o GPU.



```

# Creates a graph.
with tf.device('/gpu:0'):
    a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])
    b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])
    c = tf.matmul(a, b)

    # Creates a session with log_device_placement set to True.
    with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as s:
        # Runs the op.
        print(s.run(c))
  
```

Il valore di un `Placeholder` deve essere inserito utilizzando l'argomento facoltativo `feed_dict` in `Session.run`, `Tensor.eval()` o `Operation.run`.

```

import tensorflow as tf
A = tf.placeholder(tf.float32, shape=(None, 3))
B = A + 5
with tf.Session() as sess:
    B_val_1 = B.eval(feed_dict={A: [[1, 2, 3]]})
    B_val_2 = B.eval(feed_dict={A: [[4, 5, 6], [7, 8, 9]]})
    print(B_val_1, B_val_2)

```

TensorBoard

Per utilizzare TensorBoard e visualizzare il grafico di un'applicazione, è necessario salvare il grafico dal codice sorgente:

```

import tensorflow as tf
x = tf.Variable(2, name="x")
y = tf.Variable(3, name="y")

add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)

with tf.Session() as sess:
    # add this line to use TensorBoard.
    sess.run(x.initializer)
    sess.run(y.initializer)
    writer = tf.summary.FileWriter('./graphs', sess.graph)
    z = sess.run(pow_op)
writer.close() # close the writer when you're done using it

```

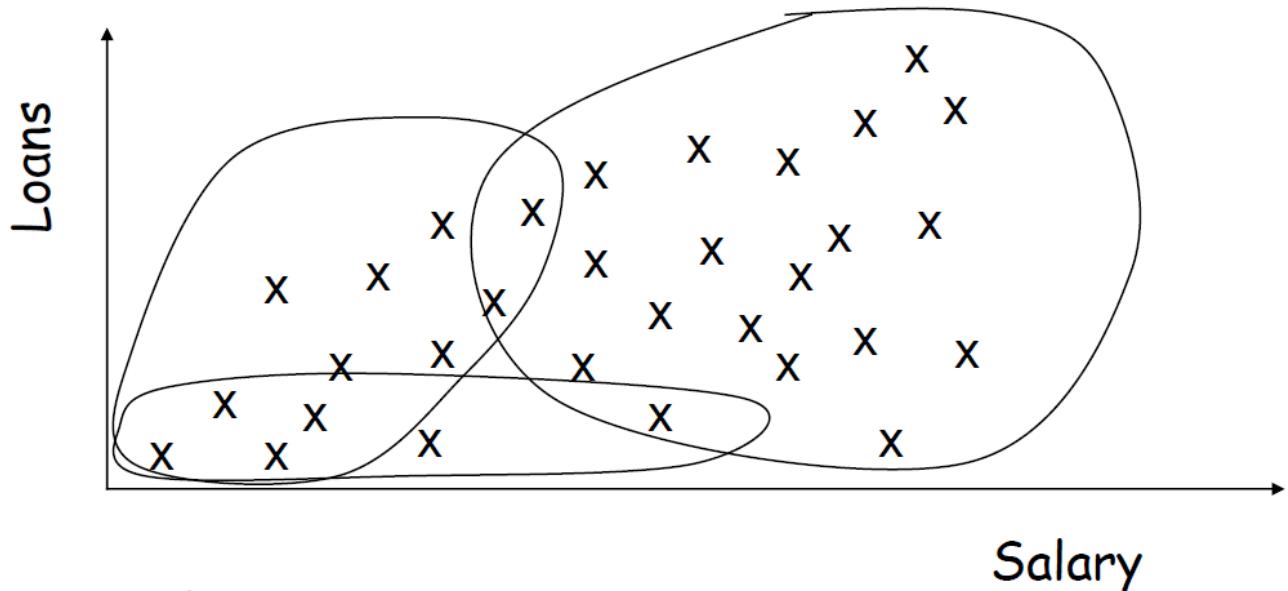
E poi richiamare dalla console

tensorboard --logdir=".//graphs" --port 6006

Quindi aprire il browser ed andare su: *http://localhost:6006/*

Clustering

Dato un insieme di punti ed una nozione di distanza, raggruppa i punti in un numero di cluster, in modo che i membri dello stesso cluster siano vicini/simili tra loro ed i membri tra diversi cluster siano dissimili.



Gerarchico

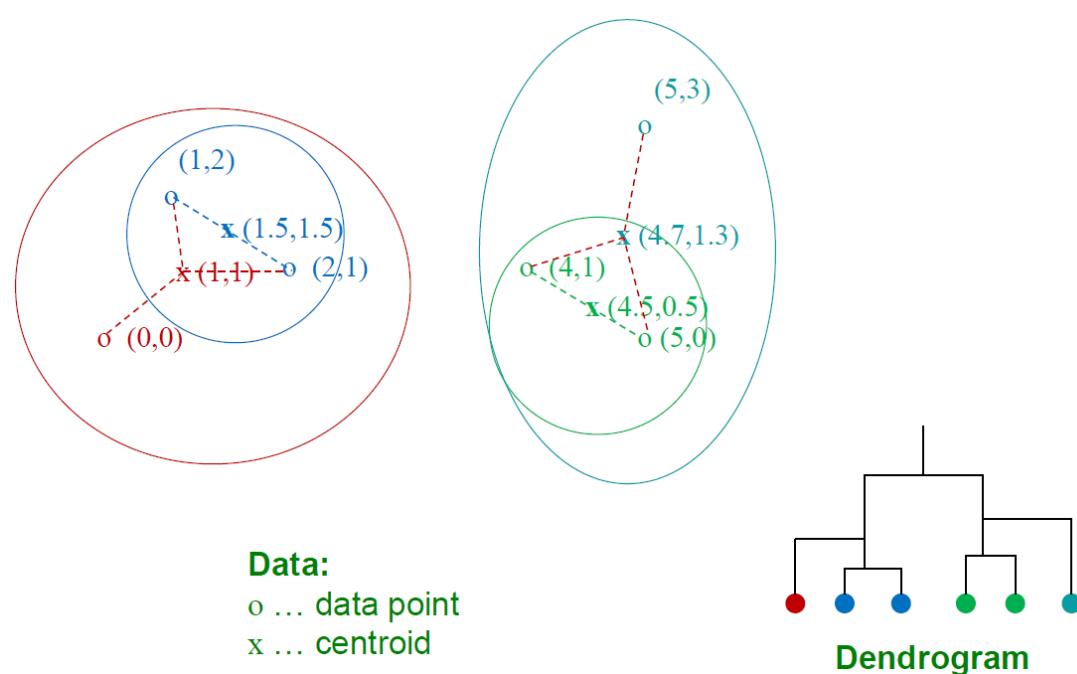
Esistono due tipi di cluster gerarchico:

- Agglomerativo: bottom up;
- Divisivo: Top down

Operazione chiave: Combina ripetutamente due cluster più vicini.

Ogni cluster ha un **centroide** che è un punto che rappresenta la media dei suoi punti (dati).

Si misurano le distanze dei cluster in base alle distanze dei centroidi.



Nel caso di uno spazio non euclideo, si calcola il **clustroide** che è il punto esistente tra i dati più vicino agli altri punti, e si tratta come se fosse un centroide per misurare la distanza dagli altri.

Possibili significati di punto "più vicino" ad altri punti:

- La più piccola distanza massima da altri punti
- Distanza media più piccola da altri punti
- Somma minima dei quadrati delle distanze da altri punti

$$\min_c \sum_{x \in C} d(x, c)^2$$

Un altro approccio nella misurazione del cluster più vicino prevede la scelta di una nozione di "coesione" di cluster. In pratica si uniscono i cluster la cui unione è più coesa:

- Usa il diametro del cluster unito = distanza massima tra i punti del cluster
- Usa la distanza media tra punti nel cluster
- Utilizzare un approccio basato sulla densità

Implementazione *naive* del clustering gerarchico:

Ad ogni passo, calcola le distanze a coppie tra tutte le coppie di cluster, quindi unisci $\rightarrow O(N^3)$.

Un'attenta implementazione utilizzando la coda di priorità può ridurre il tempo a $O(N^2 \log n)$.

K-Means

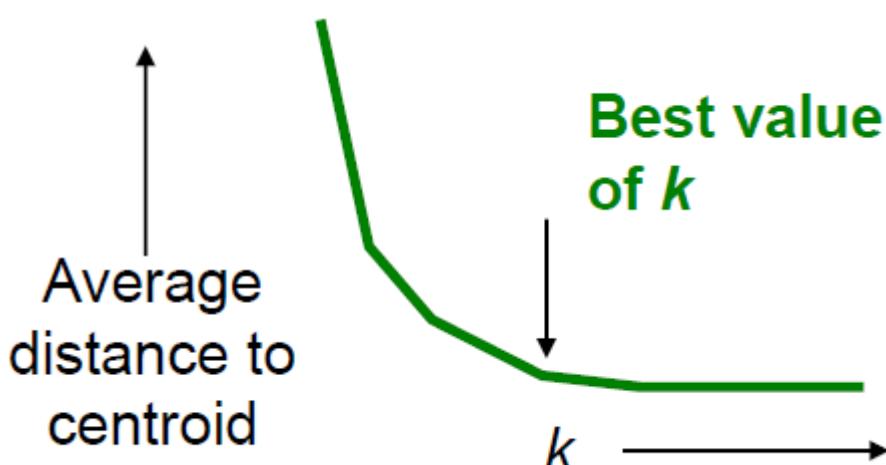
ancora troppo costoso per grandi dataset che non entrano in memoria

Inizia selezionando "k", il numero di cluster. Inizializza i cluster selezionando un punto per cluster.

1. Per ogni punto, posizionalo nel cluster il cui centroide è più vicino
2. Dopo aver assegnato tutti i punti, aggiornare le posizioni dei centroidi dei "k" cluster
3. Riassegna tutti i punti al loro centroide più vicino
4. Sposta i punti tra i cluster
5. Ripetere 2 e 3 fino alla convergenza

Convergenza: i punti non si spostano tra i cluster e i centroidi si stabilizzano

Per decidere il valore giusto di "k", si osserva la variazione della distanza media dal baricentro all'aumentare di k. La media scende rapidamente fino a che "k" diventa un valore ottimale, successivamente cambia di poco.



BFR

È una variante di K-Means progettata per gestire set di dati molto grandi (residenti su disco). L'obiettivo è che la memoria richiesta sia di complessità simile a **O(clusters)** piuttosto che **O(dati)**.

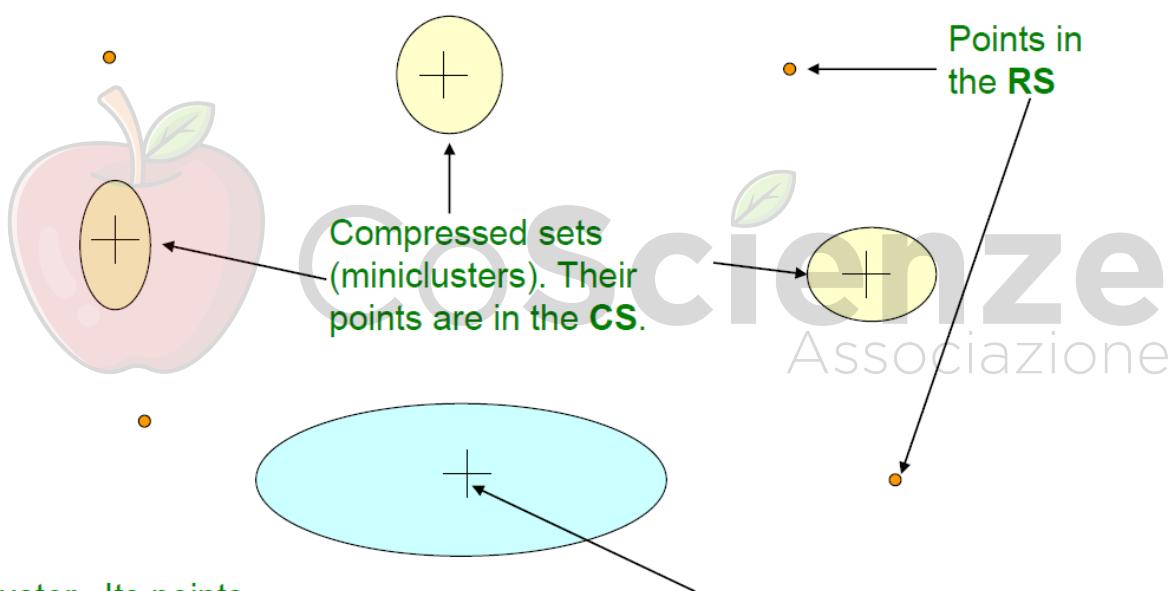
La maggior parte dei punti sono riassunti da semplici statistiche.

Per iniziare, si selezionano gli iniziali “ k ” centroidi con uno dei seguenti approcci:

- Si prendono “ k ” punti casuali
- Si prende un piccolo campione casuale e si raggruppa in modo ottimale
- Si Prende un campione, si sceglie un punto a caso, e poi “ $k-1$ ” punti in più, ciascuno il più lontano possibile da quello precedentemente selezionato

A questo punto, abbiamo tre serie di punti di cui teniamo traccia:

- **Discard set (DS):** Punti abbastanza vicini a un centroide per essere riassunti. Ogni DS è riassunto con statistiche come numero di elementi, la somma degli elementi, e la somma dei quadrati.
- **Compression set (CS):** Gruppi di punti vicini tra loro ma non vicini ad un centroide esistente. Questi punti formano minicluster. Sono riassunti in minicluster, ma non assegnato a un cluster.
- **Retained set (RS):** Punti isolati in attesa di essere assegnati a un minicluster



A cluster. Its points are in the **DS**.

Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

I passi dell'algoritmo:

1. Si effettua un caricamento nella memoria principale di un campione di dati
2. Si trovano quei punti che sono “sufficientemente vicini” ad un centroide del cluster e si aggiungono a quel cluster ed al DS
3. Per i restanti punti si utilizza qualsiasi algoritmo di clustering per raggrupparli insieme a punti della vecchia RS.
Punti raggruppati andranno a far parte del nuovo CS.
4. Si regolano le statistiche dei cluster DS, tenendo conto dei nuovi punti

5. Il CS ora ha minicluster vecchi e nuovi. Si considera di unirli, riassumendo le loro statistiche
6. Punti assegnati a un cluster di DS o minicluster del CS vengono scartati e scritti nella memoria secondaria insieme al cluster o minicluster di appartenenza
7. Se questo è l'ultimo round, unisci tutti i minicluster nel CS e tutti i punti RS nel cluster DS più vicino o trattali come valori anomali e non raggruppali mai

Per decidere se un punto è vicino o meno ad un cluster, il BFR suggerisce l'uso della **distanza di Mahalanobis**.

Distanza euclidea dal centroide di un punto, al centroide del cluster normalizzato con la deviazione standard del cluster per ogni dimensione.

Per il punto (x_1, \dots, x_n) e il baricentro (c_1, \dots, c_n)

$$d(x, c) = \sqrt{\sum_{i=1}^d \left(\frac{x_i - c_i}{\sigma_i} \right)^2}$$

Se i cluster sono normalmente distribuiti in "d" dimensioni, dopo la trasformazione, $1 \text{ std} = \sqrt{d}$

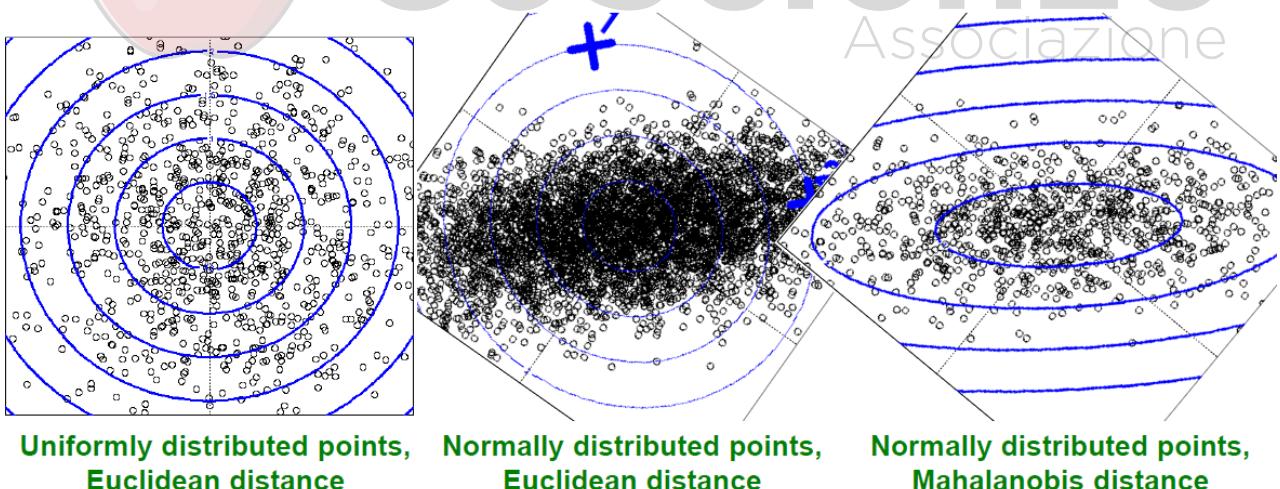
Ovvero, il 68,26% (es. $2 \times 34,13\%$) dei punti del cluster avrà una distanza di Mahalanobis dal suo baricentro $< 1\sigma = \sqrt{d}$

Il 95,44% di loro avrà un M.D. $< 2\sigma$

Il 99,99% di loro avrà un M.D. $< 4\sigma$

► Euclidean vs. Mahalanobis distance

Contours of equidistant points from the origin



Due CS possono essere combinati solo se unendoli il cluster creato avrà varianza minore di una certa soglia.

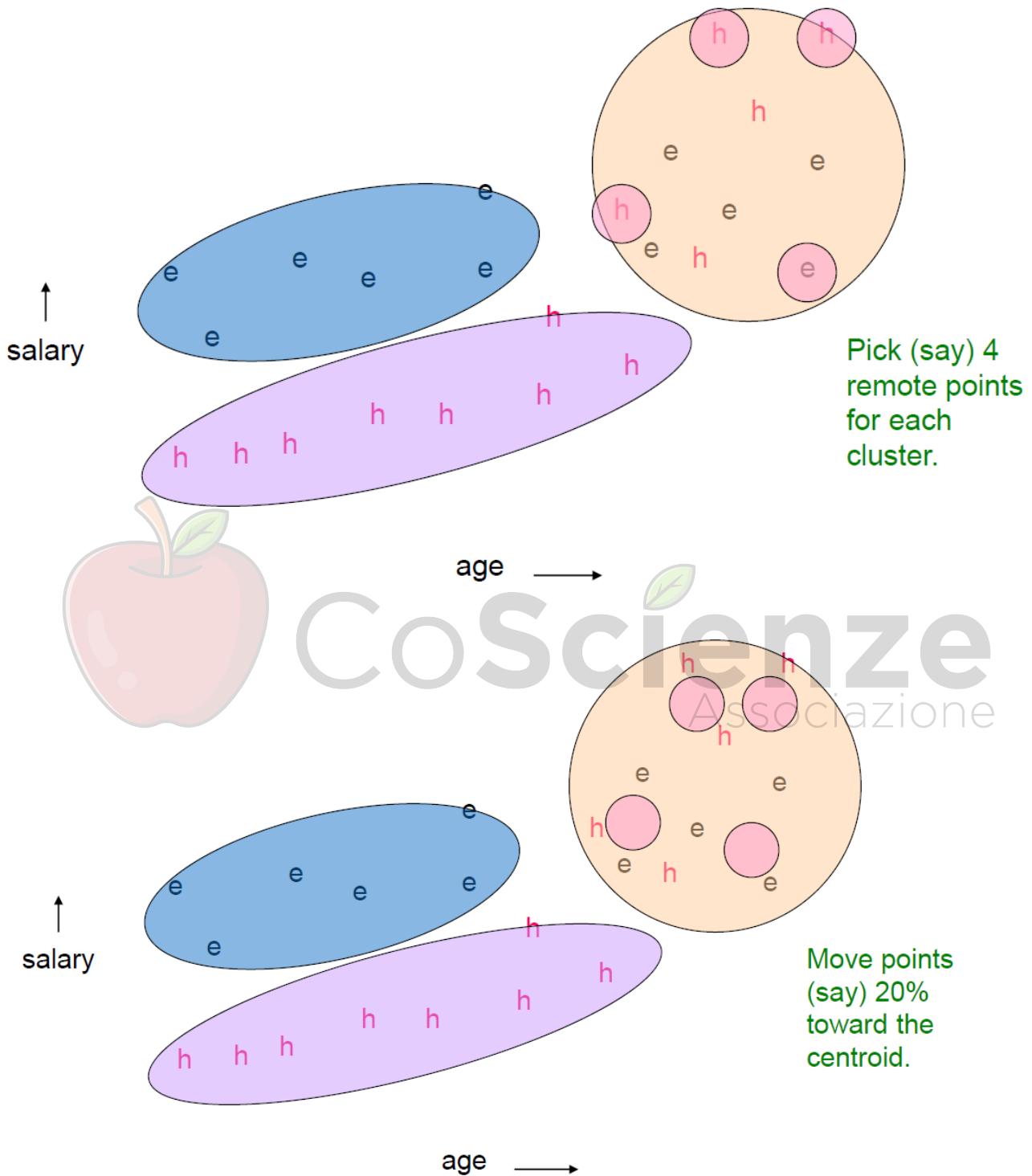
CURE Algoritm

Utilizza una distanza euclidea, consente ai cluster di assumere qualsiasi forma, utilizza una raccolta di punti per rappresentare i cluster.

Il primo passo dell'algoritmo:

1. Si sceglie un campione casuale di punti che rientrano nella memoria principale
2. Cluster iniziali: Si raggruppano questi punti in modo gerarchico

3. Si scelgono i punti più rappresentativi: Per ogni cluster, si sceglie un campione di punti, più dispersi possibile
4. Dal campione, si scelgono i punti isolati del punto 3 spostandoli del (diciamo) 20% verso il baricentro del cluster



Il secondo passo dell'algoritmo:

1. Ora, si scansiona nuovamente l'intero set di dati e si controlla ogni punto " p " nel set di dati
2. Si posiziona nel cluster più vicino

ANN

I primi successi delle ANN fino agli anni '60 hanno portato alla convinzione diffusa che presto avremmo dialogato con macchine intelligenti.

All'inizio degli anni '80 vi fu una rinascita di interesse per le ANN, con lo sviluppo di nuove architetture di rete e di migliori tecniche di allenamento.

Negli anni '90 l'utilizzo delle SVM era preferito dalla maggior parte dei ricercatori.

Ad oggi ci sono ragioni più importanti che hanno portato allo sviluppo di questo interesse:

- Ora è disponibile un'enorme quantità di dati per addestrare reti neurali
- Il tremendo aumento della potenza di calcolo dagli anni '90
- Gli algoritmi di addestramento sono stati migliorati
- Nella pratica, la convergenza all'ottimo locale è rara
- Quando si verificano, gli ottimi locali sono spesso vicini alla soluzione ottimale
- Le ANN sono entrate in un circolo virtuoso di finanziamenti e progresso

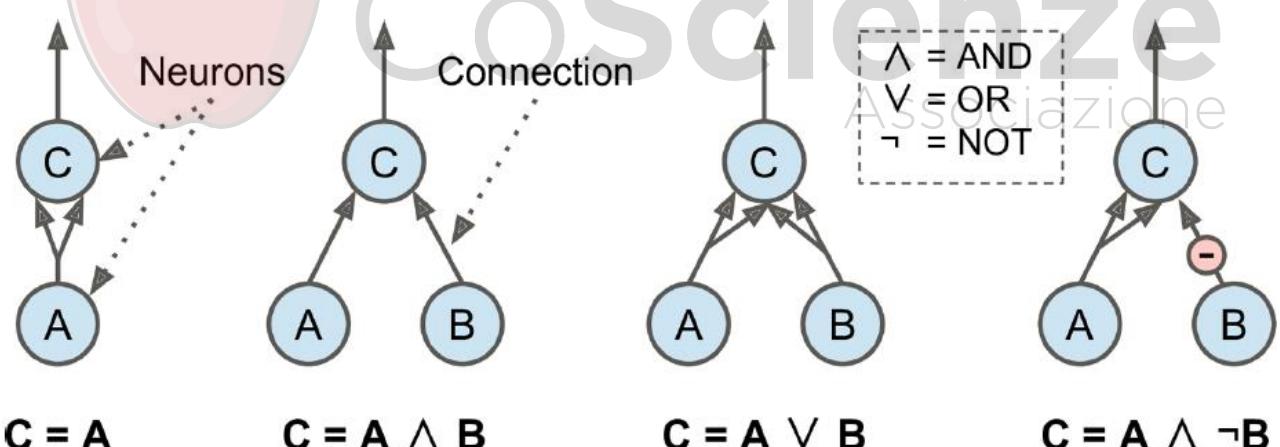
La cellula delle cortecce cerebrali animali è composta da un corpo contenente il *nucleo* e la maggior parte dei componenti complessi della cellula ed inoltre molte estensioni ramificate chiamate *dendriti*.

I neuroni ricevono impulsi elettrici chiamati *segnali* provenienti da altri neuroni attraverso le *sinapsi*.

La ricerca è ancora incentrata sull'architettura delle reti neurali biologiche **BNN**, alcune parti del cervello sono state mappate, e sembra che i neuroni siano spesso organizzati in strati consecutivi.

McCulloch e Pitts hanno proposto un modello semplice, *Neurone artificiale*.

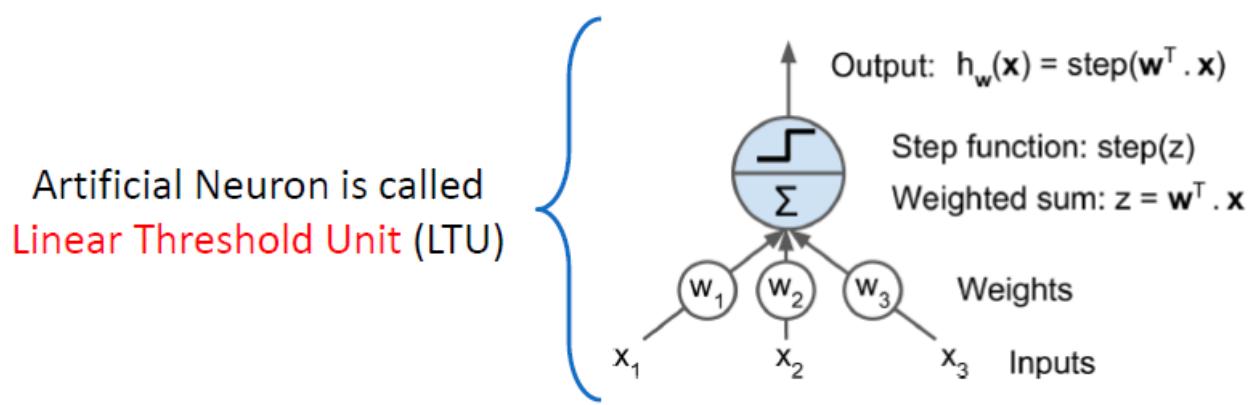
Dispone di uno o più ingressi binari (on/off) e una uscita binaria. Attiva la sua uscita quando più di un certo numero dei suoi ingressi sono attivi (2 nell'esempio).



Perceptron

Il **perceptron** è una delle architetture più semplici di una ANN.

Gli input sono numero piuttosto che binari on/off.



Viene effettuata una somma pesata in cui ogni input è moltiplicato con il peso della corrispondente connessione, ed a questa somma viene aggiunto il termine chiamato **bias**.

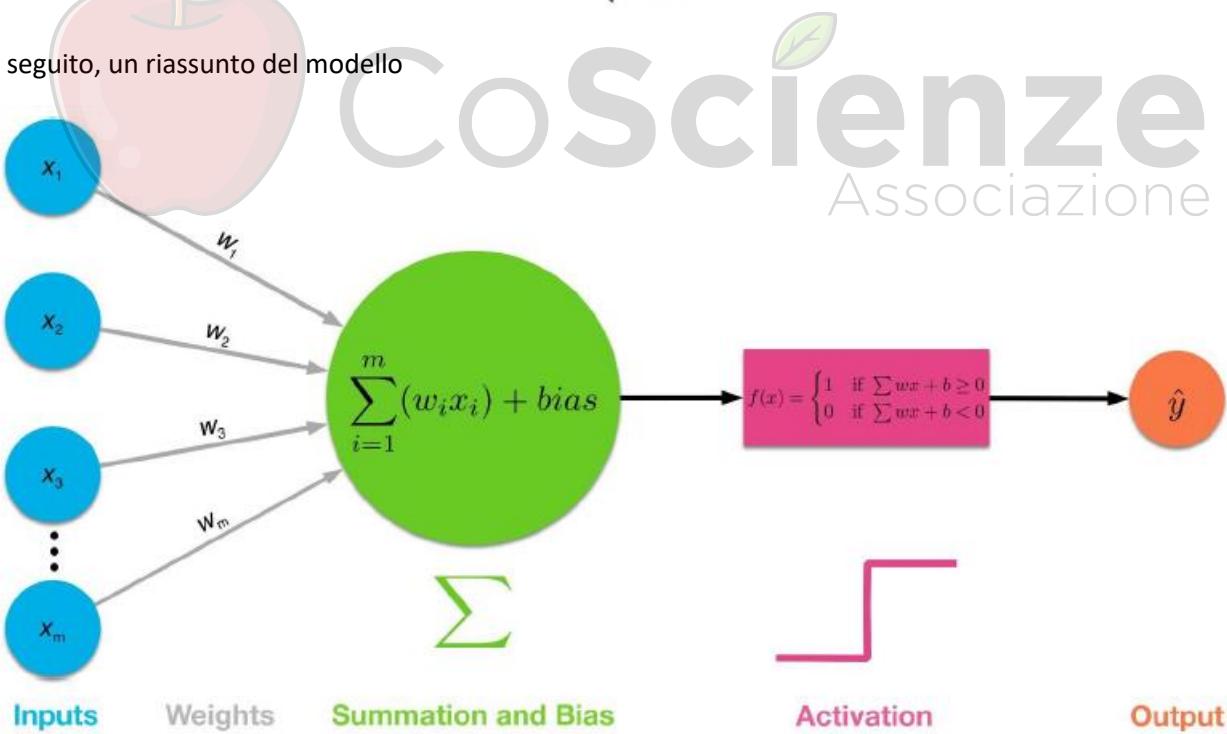
Il risultato viene passato ad una funzione di attivazione che lascia passare o meno il segnali agli strati successivi.

Tra queste funzioni, troviamo le due funzioni **step**:

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

$$\text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Di seguito, un riassunto del modello



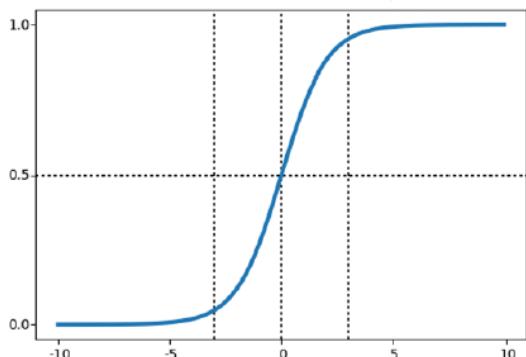
Un'altra funzione di attivazione è la **sigmoid**.

Il motivo principale per cui utilizziamo la funzione sigmoidale è perché esiste tra 0 e 1.

Viene utilizzata in modelli in cui l'obiettivo è predire una probabilità come output.

La funzione è differenziabile ed è monotonica, ma non nella derivata.

Può causare un blocco durante la fase di training del modello.



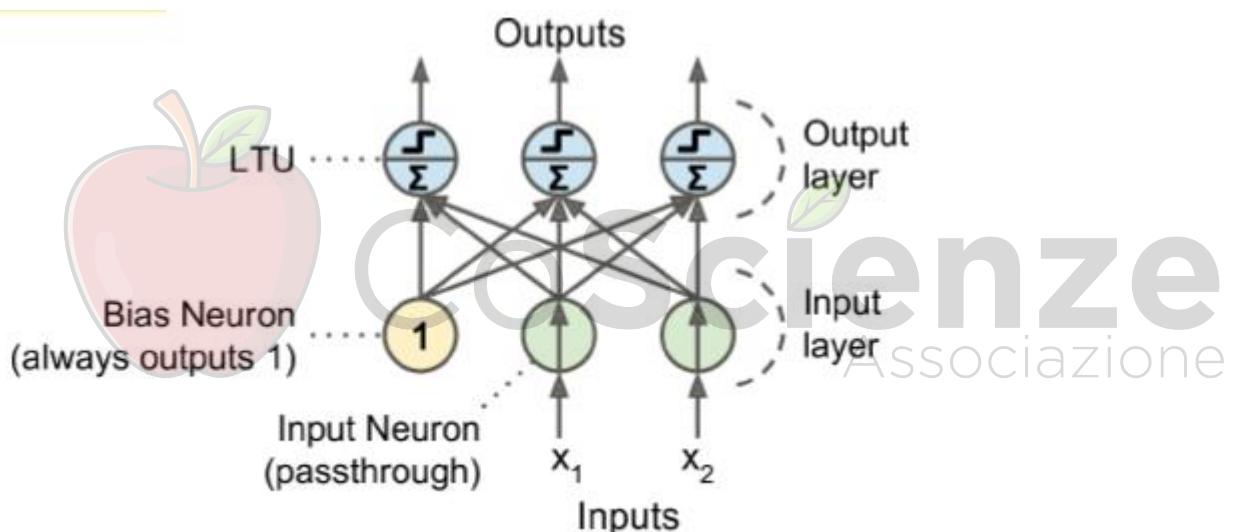
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$z = \sum_{i=1}^m w_i x_i + bias$$

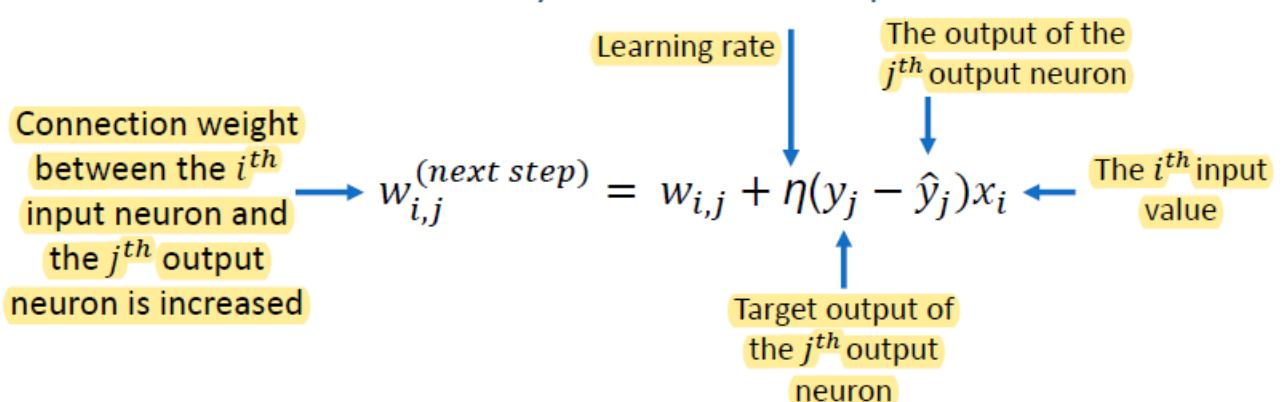
Multioutput Classifier

Classifica le istanze contemporaneamente in tre diverse classi binarie.

La caratteristica di bias è rappresentata da un neurone speciale chiamato **bias neuron** che manda in output 1 tutte le volte.



La regola di Hebb afferma che il peso di connessione tra due neuroni aumenta ogni volta che hanno la stessa uscita.



Per ogni neurone di output che ha prodotto un errore nella previsione, vengono rafforzati i pesi di connessione dagli input che hanno, invece, contribuito alla previsione corretta.

Se le istanze di addestramento sono separabili linearmente, Rosenblatt(che ha proposto l'algoritmo di training basato sulla regola di Hebb) ha dimostrato che questo algoritmo converge ad una soluzione.

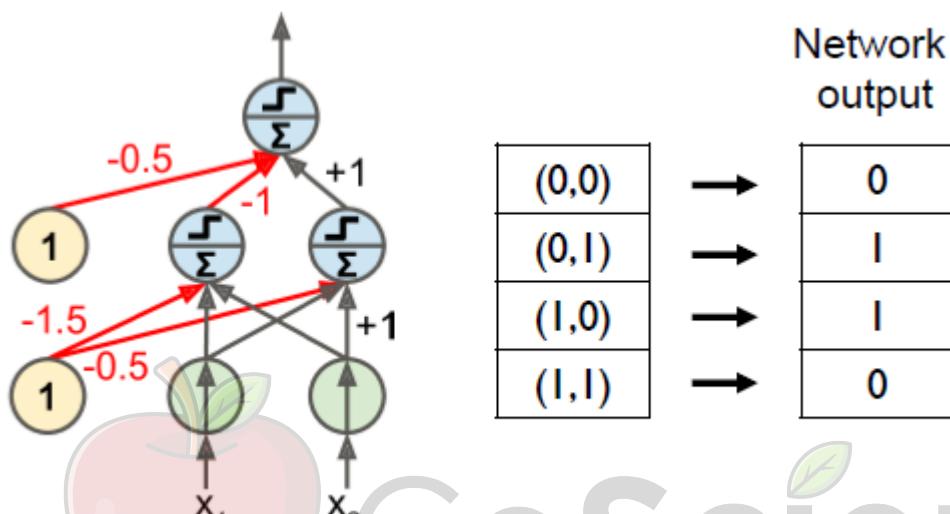
Questo è chiamato il **teorema di convergenza del percepitrone**.

Multi-Layer Perceptron

I percepitori sono incapaci di risolvere alcuni problemi banali (es. il problema di classificazione Exclusive OR (XOR)).

I limiti possono essere spesso eliminati concatenando più percepitori, che formano un **Multi-Layer Perceptron (MLP)**.

MLP può risolvere il problema XOR:

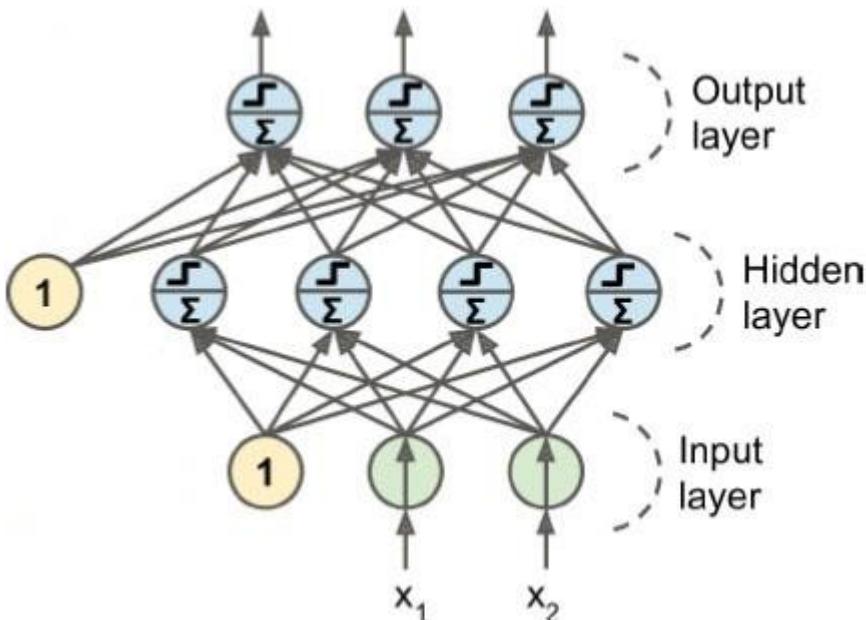


Un MLP è composto da:

- uno strato di input;
- uno o più livelli di LTU chiamati **hidden layers**;
- uno strato finale di LTU chiamato **output layer**.

Una ANN con due o più hidden layer è chiamata **deep neural network DNN**.

CoScienze
Associazione



Backpropagation

Per ogni istanza di addestramento l'algoritmo:

1. Alimenta le istanze nella rete
2. Calcola l'output di ogni neurone in ogni livello consecutivo (**forward pass**);
3. Misura l'errore di output della rete come la differenza tra l'output desiderato e output effettivo
4. Calcola quanto ha contribuito ogni neurone nell'ultimo hidden layer all'errore di ciascun neurone di output, e prosegue all'indietro nella rete fino a raggiungere il livello di input
5. Misura il gradiente di errore su tutti i pesi delle connessioni nella rete propagandolo all'indietro
6. L'ultimo passaggio dell'algoritmo di *backpropagation* consiste nell'utilizzo del *Gradient Descent* su tutti i pesi delle connessioni in rete, utilizzando gli errori dei gradienti misurati in precedenza

In altre parole, per ogni istanza di training la *backpropagation* prima fa una previsione (**forward pass**), misura l'errore, quindi passa attraverso ogni livello all'indietro per misurare il contributo di errore di ciascuna connessione (**reverse pass**) e infine modifica leggermente i pesi della connessioni per ridurre l'errore (**Gradient Descent step**).

Per farlo funzionare correttamente, gli autori hanno sostituito la funzione step con la funzione logistica:

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

Perché la funzione step contiene solo segmenti piatti, quindi lì non c'è gradiente con cui lavorare, mentre la funzione logistica ha definita una derivata diversa da zero ovunque, consentendo al *Gradient Descent* di fare qualche progresso ad ogni passo.

Alternative MLP alle Step Function

- **The hyperbolic tangent function:** Il suo output varia da -1 a 1 invece che da 0 a 1

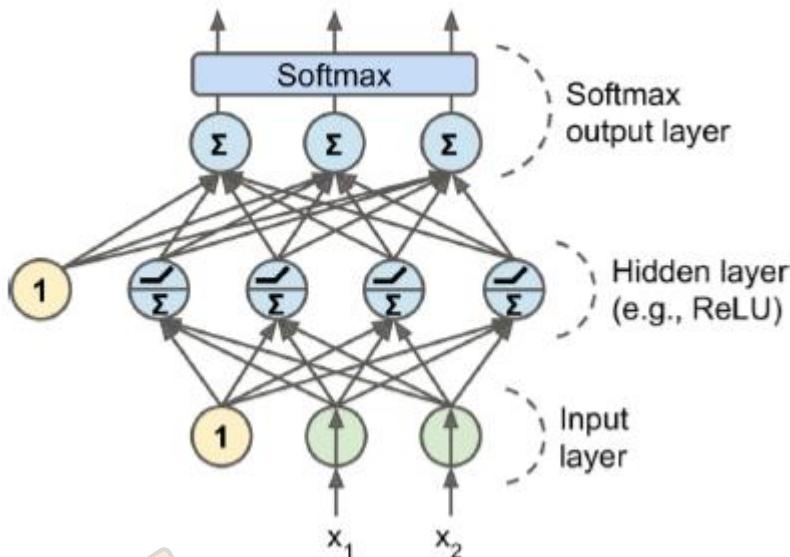
$$\tanh(z) = 2\sigma(2z)-1$$
- **The ReLU function:**

$$\text{ReLU}(z) = \max(0, z)$$

Classification

Ci sono due tipi di classificazione:

- **Classificazione binaria:** ogni output corrisponde ad una classe binaria differente;
- **Classificazione generale:** Quando le classi sono esclusive lo strato di output è tipicamente modificato sostituendo le funzioni di attivazione con una funzione di attivazione **softmax**



La funzione **Softmax** è una funzione che prende come input un vettore di “ K ” numeri reali e lo normalizza in una distribuzione di probabilità costituita da “ K ” probabilità.

Scegliamo la classe con la probabilità più alta.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K$$

Il modo più semplice per allenare una MLP con Tensorflow è utilizzare la classe *DNNClassifier* che costituisce una rete neurale profonda con una funzione di attivazione softmax nell’output layer

```
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[300,100], n_classes=10,
                                         feature_columns=feature_cols)
dnn_clf = tf.contrib.learn.SKCompat(dnn_clf) # wrap DNNClassifier in a Scikit-Learn compatibility helper
dnn_clf.fit(X_train, y_train, batch_size=50, steps=40000) # 40,000 training iterations using 50 instances
```

CNN

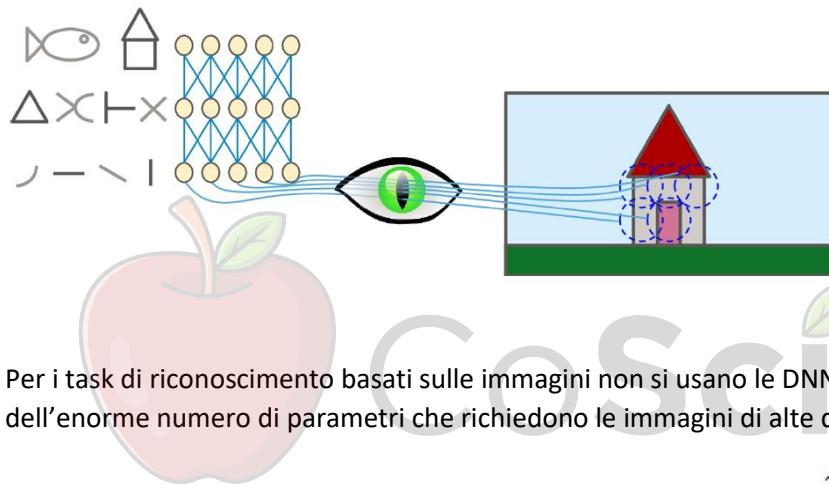
Le reti neurali convoluzionali, o CNN, sono progettate per mappare i dati di un'immagine in una variabile di output.

Il termine convoluzione assume diversi significati:

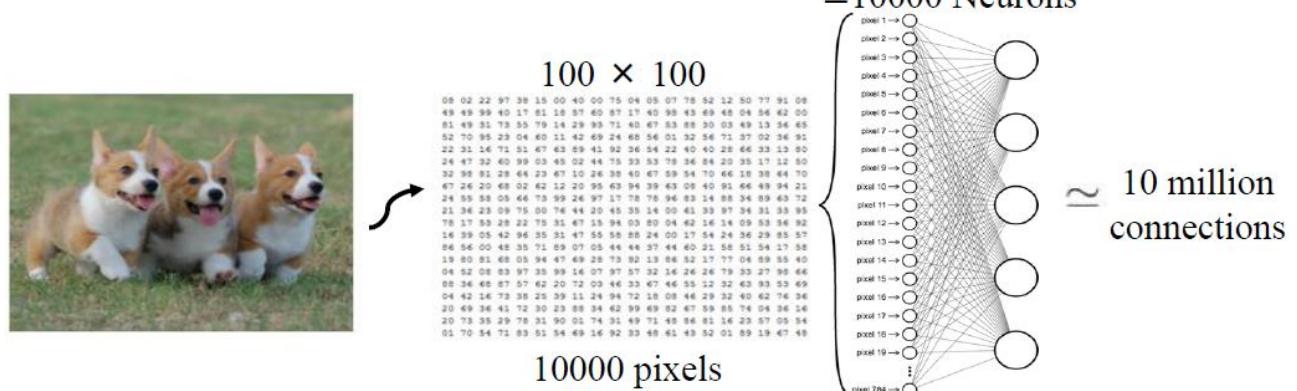
- In matematica è un'operazione che fa scorrere una funzione lungo un'altra e misura l'integrale della loro moltiplicazione.
- Nelle reti neurali è una funzione derivata da due funzioni date che esprime come il valore e la forma di una sono modificati dall'altra.

David H. Hubel e Torsten Wiesel hanno eseguito una serie di esperimenti sui gatti e poi sulle scimmie. Hanno scoperto che i campi ricettivi di diversi neuroni possono sovrapporsi e insieme formano l'intero campo visivo.

Queste osservazioni hanno portato all'idea che i neuroni di livello alto si basano sugli output di neuroni vicini di livello inferiore.

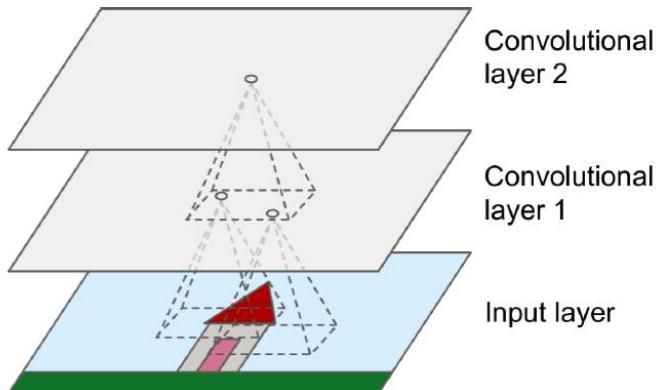


Per i task di riconoscimento basati sulle immagini non si usano le DNN, dato che falliscono a causa dell'enorme numero di parametri che richiedono le immagini di alte dimensioni.



Convolutional Layer

I neuroni nel primo strato convoluzionale sono collegati solo ad una porzione di pixel che fa parte del loro campo recettivo.



Un neurone alla riga “ i ” colonna “ j ” di uno strato è collegato alle uscite dei neuroni nello strato precedente alle righe da “ i ” a “ $i + f(h) - 1$ ”, colonne da “ j ” a “ $j + f(w) - 1$ ” dove “ $f(h)$ ” e “ $f(w)$ ” sono l'altezza e la larghezza del campo ricettivo.

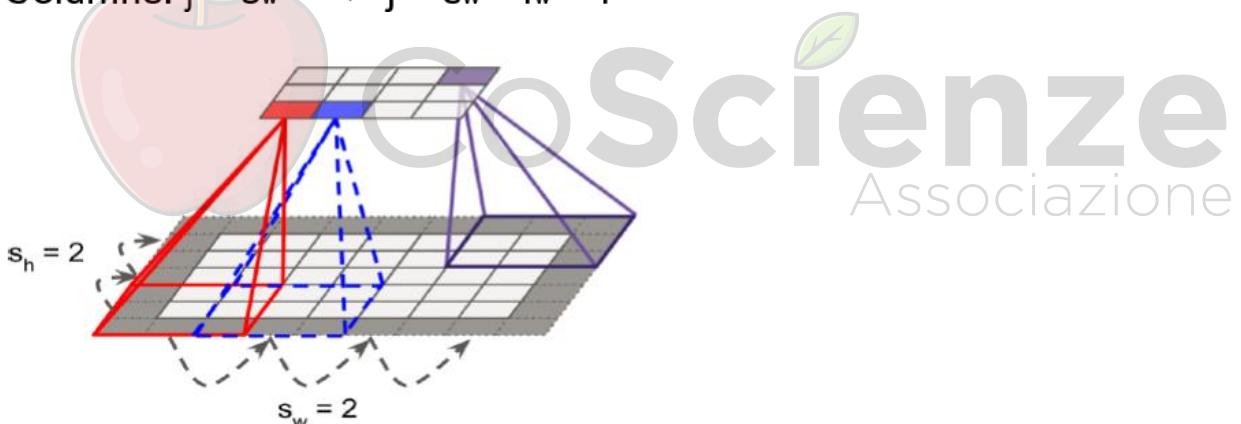
Affinché un livello abbia la stessa altezza e larghezza del precedente, vengono aggiunti degli zeri, questa tecnica è chiamata **Zero Padding**.

La distanza tra due campi ricettivi consecutivi è chiamata **stride**.

Un neurone alla riga “ i ”, colonna “ j ” nello strato superiore è collegato alle uscite di i seguenti neuroni nello strato precedente:

Rows: $i \times s_h \rightarrow i \times s_h + f_h - 1$

Columns: $j \times s_w \rightarrow j \times s_w + f_w - 1$

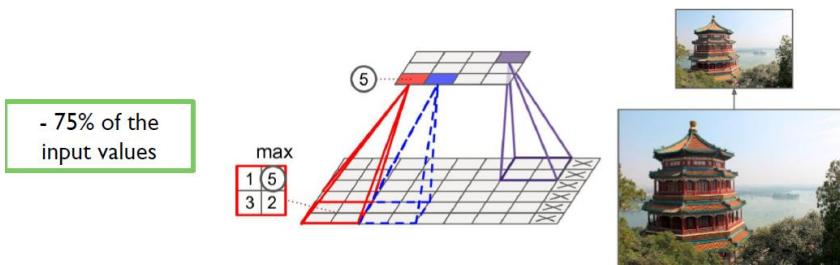


Pooling Layer

L'obiettivo del livello di pooling è sottocampionare (ridurre) l'immagine di input per ridurre il carico di calcolo, l'utilizzo della memoria e il numero di parametri.

È un livello che non prevede pesi.

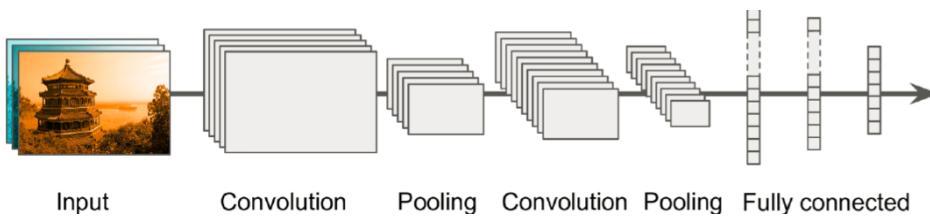
Solo il valore di input massimo in ogni kernel arriva a livello successivo, mentre gli altri input vengono eliminati.



CNN Architectures

Le tipiche architetture della CNN, quindi, concatenano alcuni livelli convoluzionali con uno strato di pooling, poi altri strati convoluzionali, quindi un altro livello di pooling e così via.

In cima allo stack c'è un normale feedforward rete neurale (FNN), in questo modo il livello finale emette la previsione.

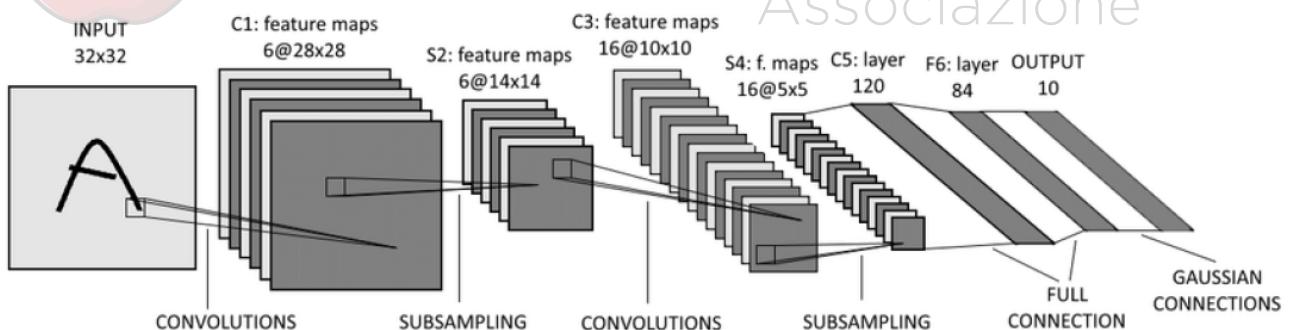


Una buona misura per valutare queste architetture è il tasso di errore in competizioni come la sfida ILSVRC ImageNet.

Il tasso di errore dei primi cinque è il numero di immagini di prova per cui le prime 5 previsioni del sistema non includevano la risposta corretta.

Alcune delle principali architetture:

- **LeNet-5:** È ampiamente utilizzata per il riconoscimento delle cifre scritte a mano (MNIST).



Characteristics	The Gap
<ul style="list-style-type: none"> • Repeat of Convolution – Pooling – Non Linearity • Average pooling • Sigmoid activation for the intermediate layer • tanh activation at F6 • 5x5 Convolution filter • 7 layers and less than 1M parameters 	<ul style="list-style-type: none"> • Slow to train • Hard to train (Neurons dies quickly) • Lack of data

- **AlexNet:** molto simile alla precedente, ma più complessa e profonda. È stato il primo modello a concatenare strati convoluzionali direttamente sopra l'un l'altro invece di concatenare uno strato di pooling sopra ciascun strato convoluzionale.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

Per ridurre l'overfitting, gli autori hanno introdotto due tecniche:

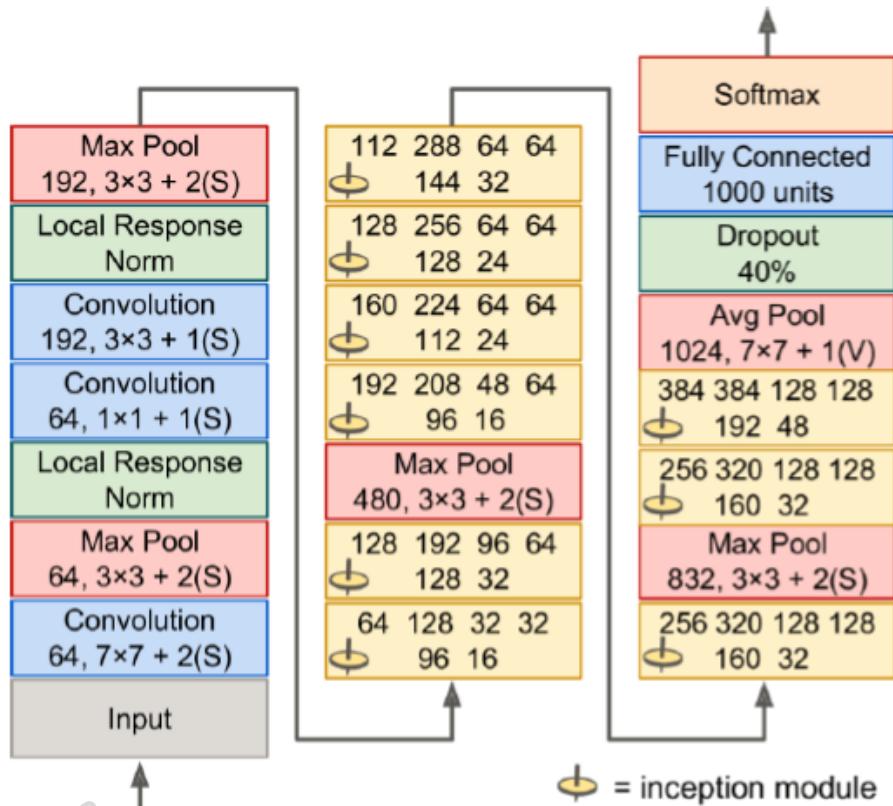
- *Dropout:* elimina una percentuale di neuroni tra uno strato e l'altro;
- *Data Augmentation:* crea varianti di immagini andando a ruotare, spostare, ingrandire le immagini sorgente.

Inoltre, AlexNet usava un ulteriore step di normalizzazione dopo gli step ReLU del livello C1 e C3, chiamato **local response normalization**

$$b_i = a_i \left(k + \alpha \sum_{j=j_{\text{low}}}^{j_{\text{high}}} a_j^2 \right)^{-\beta} \quad \text{with} \quad \begin{cases} j_{\text{high}} = \min \left(i + \frac{r}{2}, f_n - 1 \right) \\ j_{\text{low}} = \max \left(0, i - \frac{r}{2} \right) \end{cases}$$

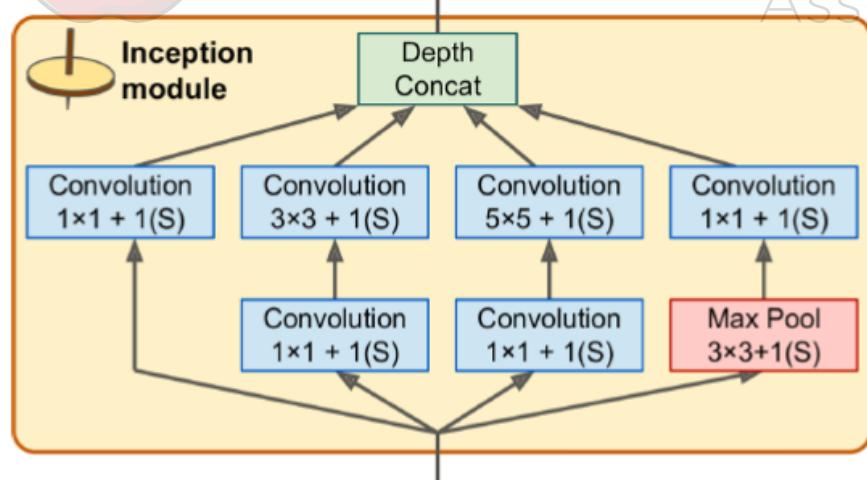
- ▶ b_i is the normalized output of the neuron located in feature map i , at some row u and column v
- ▶ a_i is the activation of that neuron after the ReLU step
- ▶ k , α , β , and r are hyperparameters: k is called the *bias*, and r is called the *depth radius*
- ▶ f_n is the number of feature maps

- **GoogleNet:** GoogLeNet utilizza sottoreti chiamate *inception modules* che aumentano le prestazioni e riducono il numero di parametri.
Infatti, GoogleNet ha un numero di parametri 10 volte inferiore rispetto a AlexNet.
Tutti gli strati convoluzionali utilizzano “ReLU” come funzione di attivazione.



In ognuno di questi moduli il segnale di ingresso viene prima copiato e inviato a 4 livelli diversi. Ogni singolo strato utilizza uno stride di 1 e il SAME padding, quindi i loro output hanno tutti la stessa altezza e larghezza dei loro input.

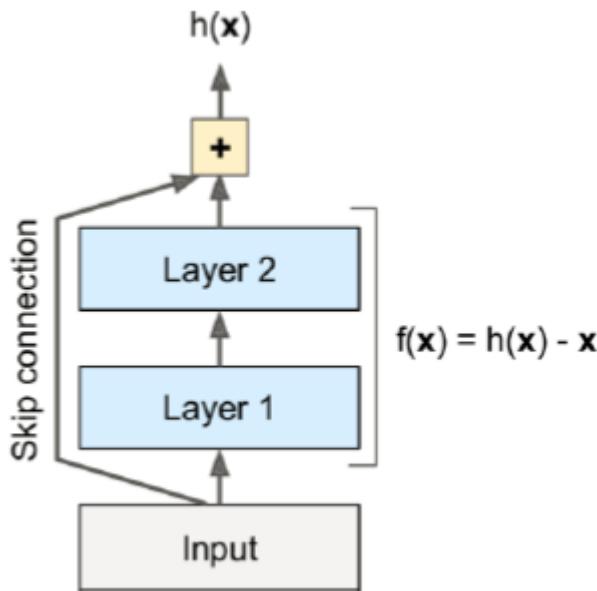
In questo modo è possibile concatenare tutte le uscite nello strato "concat" finale.



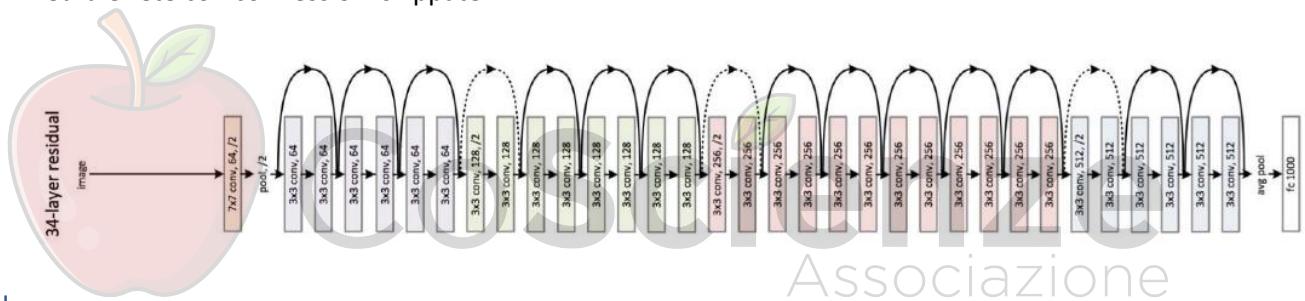
Per evitare che la parte centrale della rete "si estinguesse", gli autori di GoogLeNet avevano introdotto due classificatori ausiliari. Hanno essenzialmente applicato "softmax" alle uscite di due *inception modules* e calcolato una perdita ausiliaria sulle etichette.

- **ResNet:** Una rete profonda è buona per la memorizzazione, ma non così buona per generalizzazione. La maggior parte dei neuroni morirà durante la "back propagation" dei pesi, questo problema è chiamato *Vanishing Gradient*.

L'invenzione chiave per evitare questo problema è "skippare" 2 livelli. Grazie alle connessioni saltate, il segnale può facilmente farsi strada su tutta la rete.



La ResNet può essere vista come una pila di unità residue, dove ogni unità residua è una piccola neurale rete con connessioni skipgate.



RNN

Le reti neurali ricorrenti (RNN) rappresentano una classe di reti che riesce a predire il futuro.

Possono analizzare dati di serie temporali come i prezzi delle azioni. Nei sistemi di guida autonoma, possono anticipare traiettorie dell'auto e aiutano a evitare incidenti.

Le RNN possono generare frasi, didascalie di immagini e molto di più.

Il loro problema principale è la **scomparsa/l'esplosione del gradiente**.

Un RNN sembra un *NN feedforward*, tranne per il fatto che ha anche collegamenti che puntano all'indietro.

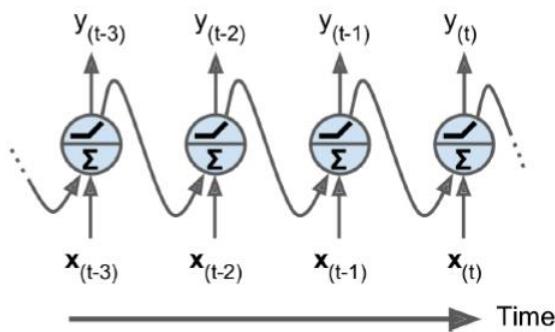
L'RNN più semplice possibile è composto da un solo neurone che riceve input, produce un output e rimanda questo a sé stesso.



- At each time step t (frame), this recurrent neuron receives the inputs $x_{(t)}$ and the output from the previous time step, $y_{(t-1)}$.

Possiamo rappresentare l'RNN precedente rispetto all'asse del tempo.

Si chiama *scorrimento della rete attraverso il tempo*



Ogni neurone ricorrente ha due serie di pesi: uno per gli ingressi $x(t)$ e l'altro per le uscite del precedente passo temporale, $y(t-1)$.

Siano $w(x)$ e $w(y)$ i vettori di peso in ingresso e in uscita.

L'output di un livello ricorrente per una singola istanza è calcolato nel seguente modo:

$$\mathbf{y}_{(t)} = \phi(\mathbf{W}_x^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_y^T \cdot \mathbf{y}_{(t-1)} + \mathbf{b})$$

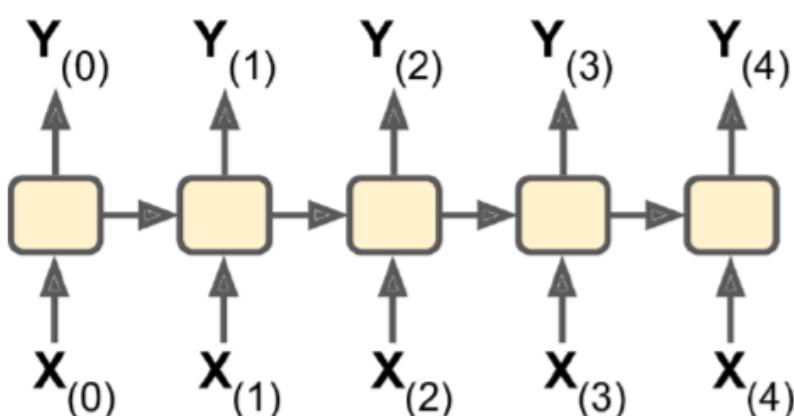
Da notare che $y(t)$ è una funzione di $x(t)$ e $y(t-1)$, che è una funzione di $x(t-1)$ e $y(t-2)$... quindi $y(t)$ è una funzione di tutti i precedenti output, tranne nella prima iterazione nel quale i pesi vengono considerati 0.

La parte di una rete neurale che conserva lo stato, attraverso passaggi temporali, è chiamata **cella di memoria (o cella)**.

Lo stato di una cella al passo temporale t , indicato con $h(t)$ è una funzione degli input in quel passo e il suo stato nel passo temporale precedente.

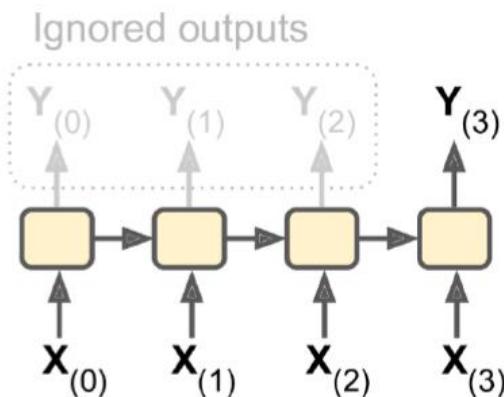
$$h_{(t)} = f(h_{(t-1)}, x_{(t)})$$

Un RNN può prendere simultaneamente una sequenza di input e produrre una sequenza di output. Questo tipo di rete è utile per prevedere serie temporali, ad esempio poiché i prezzi delle azioni usando gli ultimi N giorni e l'RNN emette i prezzi da N-1 giorni fa a domani.



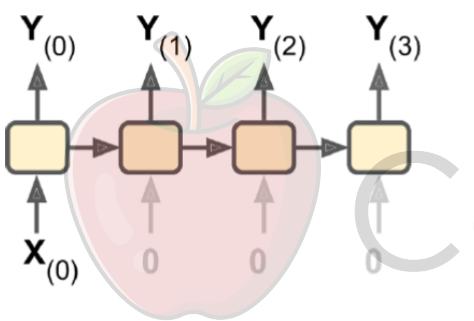
L'RNN può anche accettare una sequenza di input e ignorare tutte le uscite tranne l'ultima.

Ad esempio, l'input potrebbe essere una sequenza di parole corrispondente a una recensione di un film e l'RNN restituirebbe in uscita un punteggio di sentimento



L'RNN può anche accettare un singolo input al primo passaggio (zeri per altri passi di tempo) ed emettere una sequenza.

Ad esempio, l'input potrebbe essere un'immagine e l'output la sua didascalia

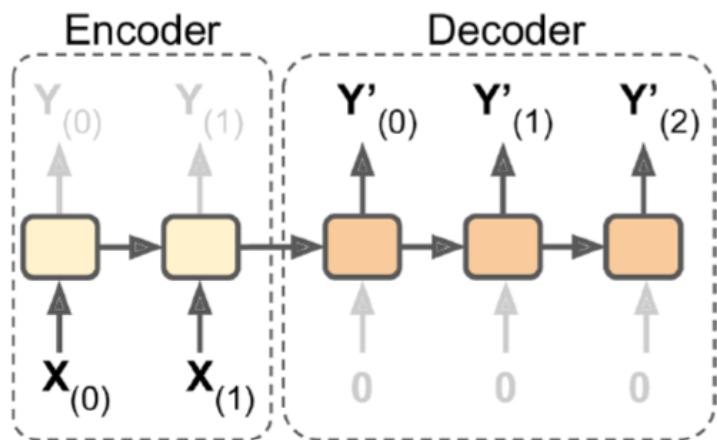


Encoder-Decoder

Potremmo anche avere una RNN sequenza-vettore, chiamato **encoder**, seguito da un RNN vettore-sequenza, chiamato **decoder**.

Ad esempio, l'input potrebbe essere una frase in una lingua, l'encoder lo convertirebbe in un vettore, mentre il decoder la tradurrebbe in un'altra lingua.

Questo modello in due fasi, funziona meglio rispetto ad una singola RNN da sequenza a sequenza, poiché le ultime parole di una frase possono influenzare il significato della prima parola, quindi meglio aspettare l'intera frase prima di tradurla.



CoScienze
Associazione

IMPORTANTE!

DISCLAIMER

Il materiale contenuto nel drive è stato raccolto e richiesto tramite autorizzazione ai ragazzi frequentanti il corso di studi di Informatica dell'Università degli Studi di Salerno.

Gli appunti e gli esercizi nascono da un uso e consumo degli autori che li hanno creati e risistemati, per tanto non ci assumiamo la responsabilità di eventuali mancanze o difetti all'interno del materiale pubblicato.

Il materiale sarà modificato aggiungendo il logo dell'associazione, in tal caso questo possa recare problemi ad alcuni autori di materiale pubblicato, tale persona può contattarci in privato ed eliminaremo o modificheremo il materiale in base alle sue preferenze. Ringraziamo eventuali segnalazioni di errori così da poter modificare e fornire il miglior materiale possibile a supporto degli studenti.

Associazione CoScienze