Appunti di machine learning

Daniele Besozzi

Anno accademico 2025/2026

Contents

Premesse

Questi sono appunti realizzati per riassumere e schematizzare tutti i concetti presentati durante il corso di machine learning tenuto presso il corso di laurea magistrale in informatica presso l'università degli studi di Milano Bicocca. Lo scopo di questo documento non è quello di sostituire le lezioni del corso o di essere l'unica fonte di studio, bensì integrare le altri fonti con un documento riassuntivo.
Mi scuso in anticipo per eventuali errori e prego i lettori di segnalarli contattandomi via mail all'indirizzo d.besozzi@campus.unimib.it.

Chapter 1

Introduzione

In questo capitolo presenterò gli aspetti matematici fondamentali per andare ad affrontare gli argomenti del corso.

1.1 Vettori e matrici

Denotiamo un vettore riga e colonna rispettivamente con (a, b, c) e $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$

dove $a, b, c \in \mathbb{R}$ sono scalari.
In generale denotiamo con le lettere maiuscole le matrici, e.g. X e i suoi elementi con X_{ij}.
$x \in \mathbb{R}^n$ è un vettore di n elementi e $X \in \mathbb{R}^{m \times n}$ è una matrice di dimensione $m \times n$.

1.2 Norme di vettori e matrici

Dato un vettore $x \in \mathbb{R}^n$ vi sono diversi tipi di norme comunemente utilizzate.

- $||x||_2 = (\sum_{i=1}^{n} x_i^2)^{\frac{1}{2}}$ è il tipo più comune e viene chiamato **norma 2** di un vettore o **norma Euclidea**. Normalmente è denotato semplicemente con $||x||$.
- $||x||_1 = |x_1| + |x_2| + \cdots + |x_n|$, detta la **norma 1** o **distanza di Manhattan**
- $||x||_\infty = \max_{1 \leq i \leq n} |x_i|$, detta la **norma** ∞.

Analogamente, per una matrice $X \in \mathbb{R}^{m \times n}$, si possono definire diverse norme:

- **Norma di Frobenius**: $||X||_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |X_{ij}|^2 \right)^{\frac{1}{2}}$
- **Norma spettrale (o norma 2)**: $||X||_2$
- **Norma 1**: $||X||_1 = \sum_{i,j} |X_{ij}|$

1.3 Notazioni generiche

Siano:

- u: variabile indipendente (input), non necessariamente un vettore o uno scalare
- v: variabile dipendente (output), come sopra

allora abbiamo che:

- $x = \phi(u)$, dove $x \in \mathbb{R}^d$ è il vettore di features e ϕ è la funzione di mapping o embedding.
- $y = \psi(v)$, dove $y \in \mathbb{R}^m$ è il vettore target (o di output) e ψ è la funzione di mapping di feature in output.

Siano $x^1, \dots x^n$ e $y^1, \dots y^n$ due dataset di n esempi, dove x^i e y^i formano la i-esima coppia di dati. Dunque n è il numero di campioni, allora posso associarvi le due matrici dei dati

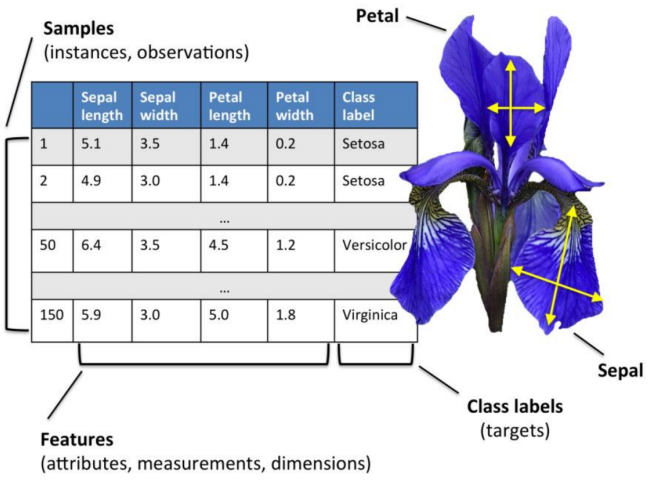
$$X = \begin{bmatrix} (x^1)^T \\ \vdots \\ (x^n)^T \end{bmatrix} \in \mathbb{R}^{n \times d}, \quad Y = \begin{bmatrix} (y^1)^T \\ \vdots \\ (y^n)^T \end{bmatrix} \in \mathbb{R}^{n \times m}$$

Le cui righe sono i vettori feature e i vettori target rispettivamente, trasposti.
Definiamo allora:

- $g_\theta : \mathbb{R}^d \to \mathbb{R}^m$ è un predittore.
- $\hat{y} = g_\theta(x)$ è la predizione di y, dato x.
- $\theta \in \mathbb{R}^p$ è il vettore di parametri del predittore.

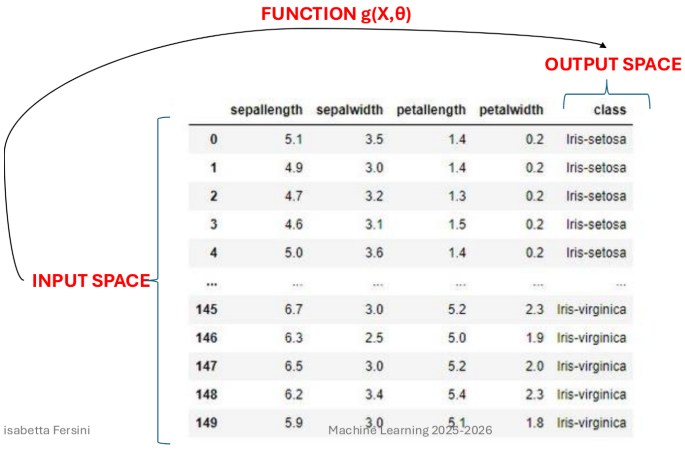
La scelta dei parametri θ a seconda dei dati viene chiamato *training* o *fitting* del predittore.
Separando le definizioni in base al dominio di riferimento, abbiamo le seguenti suddivisioni:

- Spazio di input:
  - Istanza (sample, oggetto, record): un esempio descritto da un certo numero di attributi.
  - Attributo (campo, caratteristica, variabile): misura di un aspetto di una istanza.
- Spazio di output:
  - Classe (label, target): categoria a cui appartiene una istanza.
- Predittore o modello:
  - Una funzione g con parametri θ che dato uno spazio di input X produce uno spazio di output Y.produce una predizione nello spazio di output Y.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Figure 1.1: Esempi pratici.



	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Figure 1.2: Esempi pratici 2.

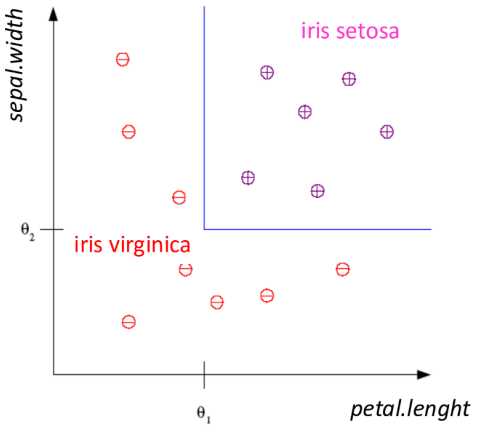


Figure 1.3: Esempi pratici 3.

1.4 Rischio atteso e rischio empirico

Quando un modello di machine learning g viene addestrato, vogliamo che abbia buone prestazioni non solo per i dati utilizzati per il training, ma anche per dati sconosciuti (*generalizzazione*). Dovremo stimare il **rischio atteso**, ovvero la loss media che dovrebbe presentarsi sulla reale distribuzione dei dati $P(x, y)$.
Supponiamo di avere un dataset di n coppie (x^i, y^i), dove x^i è il vettore di feature e y^i il vettore target associato alla i-esima istanza. Definiamo la **loss function** $L(g_\theta(x), y)$, che misura l'errore commesso dal modello g_θ.
Dunque il rischio atteso è definito come:

$$R(g_\theta) = E_{(x,y)\sim P}[L(g_\theta(x), y)]$$

Da notare però che la reale distribuzione P non è nota, dunque non possiamo stimare il rischio atteso. Le cause sono:

1. Distribuzione non nota
   Osserviamo solo un campione finito di campioni estratti dal mondo reale. La distribuzione completa che genera quei dati non è accessibile.

2. Impossibilità pratica
   Anche se conoscessimo il processo di generazione in teoria, calcolare esattamente la predizione del rischio è impossibile in generale perché richiederebbe di sommare/integrare tutte i possibili esempi.

3. Dati finiti e rumorosi
   Il dataset a disposizione è finito, spesso presenta rumore e errori di misura, o bias di raccolta. Dunque possiamo solo approssimare il rischio utilizzando una **stima empirica**.

Dunque, vogliamo stimare e minimizzare il rischio empirico. Supponiamo di avere un dataset di n coppie (x^i, y^i). Definiamo la loss function $L(g_\theta(x), y)$, che misura l'errore commesso dal modello g_θ. Allora il rischio empirico $\hat{R}(g_\theta)$ è definito come:

$$\hat{R}(g_\theta) = \frac{1}{n} \sum_{i=1}^{n} L(g_\theta(x^i), y^i)$$

La maggior parte degli algoritmi di machine learning minimizzano il rischio empirico.

$$g^* = arg \min_{g_\theta \in G} \hat{R}(g_\theta)$$

Dunque dato un dataset di training, x_i, y_i per $i = 1, \dots n$, per cui la loss sull' i-esima coppia di dati è $l(g_\theta(x_i), y_i)$, il rischio empirico è la loss media sul dataset di training. L'empirical risk minimization (ERM) si occupa di scegliere i parametri θ che minimizzano il rischio empirico.

1.5 Estrazione delle features

Utilizziamo u per denotare i dati di input grezzi, ad esempio un vettore, testo, immagine, audio, video, ecc. $x = \phi(u)$ è il vettore di features, ottenuto tramite la funzione ϕ, chiamata embedding, funzione feature o mappamento di feature. La funzione ϕ può variare molto nel livello di complessità in base al caso.

1.6 Modelli

Noi cerchiamo un predittore (o modello) $g_\theta : \mathbb{R}^d \to \mathbb{R}^m$ che dato un vettore di features $x \in \mathbb{R}^d$ produce una predizione $\hat{y} = g_\theta(x) \in \mathbb{R}^m$. La scelta del modello g_θ viene effettuata sia in basa ai dati a disposizione che in base alle conoscenze pregresse. In termini di dati grezzi, il predittore è $\hat{v} = \psi^{-1}(g(\phi(u)))$, con qualche variante se la funzione ψ è invertibile. Se $\hat{y} \approx y$ allora il predittore ha fatto una buona predizione sulla i-esima coppia di dati. Però, l'obiettivo è avere $\hat{y} \approx y$ non solo sui dati di training, ma anche su dati mai visti prima. Due forme equivalenti per rappresentare un modello sono $\hat{y} = g_\theta(x)$ e $\hat{y} = g(x, \theta)$. Si dice che la funzione g da la struttura (o forma) del predittore, θ è un parametro per il modello predittivo La scelta di un particolare $\theta \in \mathbb{R}^p$ si chiama *tuning* o *fitting* o *addestramento* del modello. Un algoritmo di apprendimento è una ricetta per scegliere i parametri θ a partire dai dati di training.

1.7 Assunzione $i.i.d.$

Prima di presentare il concept learning è necessario presentare l'assunzione $i.i.d.$ (independent and identically distributed). Dunque si assume che i dati siano stati campionati in modo indipendente e che provengano dalla stessa distribuzione.

- In teoria : L'unione delle funzioni di distribuzione di probabilità possono essere scritte come il prodotto delle funzioni di distribuzione di probabilità marginali.

$$f(X, \theta) = \prod_{i=1}^{n} f(X_i, \theta)$$

- In pratica: Ogni modello considera un campione alla volta, ignorando le feature degli altri campioni.

Possiamo notare che nella figura 1.5 sono presenti dei campi nulli, questo è dovuto al fatto che non tutti i dati sono stati raccolti, ciò presenta un problema. Anche una rappresentazione relazionale dei dati può essere effettuata, come ad esempio in figura 1.6. Infine è anche possibile combinare le due rappresentazioni, come in figura 1.7.
Il processo di **flattening** su un file è detto **proposizionalizzazione**, ovvero l'unione di più relazioni

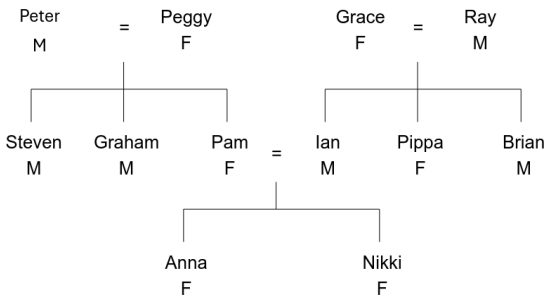


Figure 1.4: Esempio di struttura familiare

Name	Gender	Parent1	parent2
Peter	Male	?	?
Peggy	Female	?	?
Steven	Male	Peter	Peggy
Graham	Male	Peter	Peggy
Pam	Female	Peter	Peggy
Ian	Male	Grace	Ray
Pippa	Female	Grace	Ray
Brian	Male	Grace	Ray
Anna	Female	Pam	Ian
Nikki	Female	Pam	Ian

Figure 1.5: formato tabellare dei dati della figura 1.4

in una sola. Questa operazione è possibile per ogni insieme finito di relazioni finite, Un problema che emerge è la presenza di relazioni senza un numero pre specificato di oggetti. La proposizionalizzazione può introdurre delle regolarità fittizie che riflettono la struttura del database, inoltre possono introdurre dei **bias** causati da dati ripetuti.

First person	Second person	Sister of?
Peter	Peggy	No
Peter	Steven	No
...	...	...
Steven	Peter	No
Steven	Graham	No
Steven	Pam	Yes
...	...	...
Ian	Pippa	Yes
...	...	...
Anna	Nikki	Yes
...	...	...
Nikki	Anna	yes

First person	Second person	Sister of?
Steven	Pam	Yes
Graham	Pam	Yes
Ian	Pippa	Yes
Brian	Pippa	Yes
Anna	Nikki	Yes
Nikki	Anna	Yes
All the rest		No

Closed-world assumption

Figure 1.6: formato relazionale dei dati della figura 1.4

First person				Second person				Sister of?
Name	Gender	Parent1	Parent2	Name	Gender	Parent1	Parent2	
Steven	Male	Peter	Peggy	Pam	Female	Peter	Peggy	Yes
Graham	Male	Peter	Peggy	Pam	Female	Peter	Peggy	Yes
Ian	Male	Grace	Ray	Pippa	Female	Grace	Ray	Yes
Brian	Male	Grace	Ray	Pippa	Female	Grace	Ray	Yes
Anna	Female	Pam	Ian	Nikki	Female	Pam	Ian	Yes
Nikki	Female	Pam	Ian	Anna	Female	Pam	Ian	Yes
All the rest								No

Figure 1.7: formato misto dei dati della figura 1.4

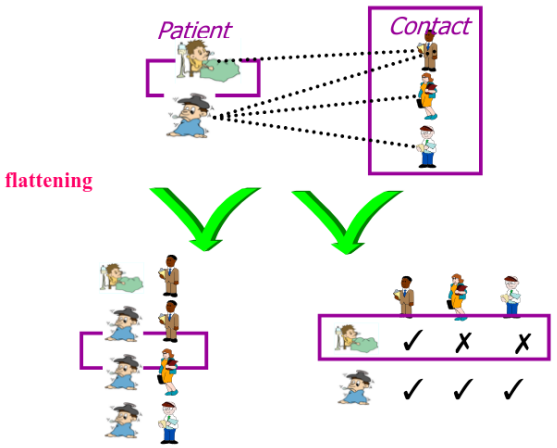


Figure 1.8: esempio di flattening

Nella colonna a sinistra della figura 1.8 possiamo notare che viene introdotto un bias dato dalla ripetizione di un dato, che in realtà è causato dalla natura relazionale dello stesso (dunque perdo la i.i.d.). Mentre colonna di destra viene introdotto un bias dato che viene considerato come non avvenuto un evento che in realtà potrebbe essere ancora non stato osservato, inoltre assumo a prescindere il numero di attributi e questo causa problemi nel momento in cui ne vengono aggiunti di nuovi.

1.8 tipi di attributi

Ogni istanza è descritta da un numero fisso predefinito di feature, i suoi attributi. Ci sono diversi tipi di attributi:

- **Quantità nominali**:Valori a simboli distinti, i valori in sé servono solo come etichette o nomi. Non vi è nessuna relazione implicita tra i valori nominali, come ordine o distanza. L'unico test effettuabile è quello di eguaglianza.
- **Quantità ordinali**: Valori che possono essere ordinati, ma non vi è alcuna informazione sulla distanza tra i valori (addizione e sottrazione non hanno significato).
- **Quantità di ratio**: Il metodo di misurazione definisce lo zero, ad esempio una distanza. Assumono valori in $\mathbb{R}$ e sono dotati di un ordine naturale. Sono possibili tutte le operazioni aritmetiche.

Perché è importante conoscere il tipo di attributo? Operazioni e confronti tra attributi di tipi diversi potrebbero non avere senso al livello semantico. Inoltre ci permettono di effettuare un check sulla validità dei valori, gestire la mancanza di essi ed effettuare confronti di eguaglianza.

Chapter 2

Inductive learning: Concept learning

Prima di iniziare con la spiegazione precisiamo la differenze tra apprendimento induttivo e deduttivo.

- **Apprendimento deduttivo**: Approccio di ragionamento dove un sistema applica regole generali per effettuare predizioni riguardanti casi specifici. È il contrario dell'apprendimento induttivo. L'idea principale è quella di partire da principi generali, teorie o regole note a priori. Applicare queste a situazioni specifiche per determinare risultati.
- **Apprendimento induttivo**:Un modello apprende regole generali o pattern a partire da esempi specifici o osservazioni. L'idea principale è quella di partire da data points specifici (esempi, dati di training) e cercare di generalizzare da questi esempi per formare un'ipotesi o regola che può prevedere casi sconosciuti.

	Inductive Learning	**Deductive Learning**
Definition	Learning patterns and models from data examples.	Applying known rules or logic to derive predictions.
Approach	Bottom-up: generalizes from specific training data.	Top-down: uses existing knowledge/rules to reason about new data.
Input	Labeled or unlabeled data (e.g., features and labels).	Formal rules, logic statements, or knowledge base.
Output	Predictive model or hypothesis (e.g., neural net).	Conclusions derived logically from existing rules.
Goal	Learn a function that maps inputs to outputs based on data.	Derive consequences or inferences from rules.
Example in ML	Training a classifier on labeled images (e.g., cat vs. dog).	Using a knowledge base and logical inference to classify an image.
Common Algorithms	Decision Trees, SVM, Neural Networks, k-NN, etc.	Logic programming, Expert Systems, Prolog-based systems.

Table 2.1: Confronto tra Inductive e Deductive Learning

Nell'apprendimento induttivo buona parte del processo di apprendimento involve la raccolta di concetti generali da esempi specifici di training. Ogni concetto può essere visto come descrivere un certo sotto insieme di oggetti o eventi definiti su un insieme più ampio. Una definizione alternativa considera ogni concetto come una funzione a valori booleani, definita su un insieme più ampio di oggetti (concept learning).

2.1 Concept learning

Consideriamo l'azione di apprendere il concetto target *"days on which my friend Aldo enjoys his favorite water sport"*. L'obiettivo è prevedere il valore *"enjoy sport"* per un giorno arbitrario, basandosi sui valori

degli altri attributi.

Sky	**Temp**	**Humid**	**Wind**	**Water**	**Forecast**	**EnjoySport**
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Mid	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Table 2.2: EnjoySport dataset

Cominciamo considerando una semplice rappresentazione nella quale ogni ipotesi consiste nella congiunzione dei requisiti sugli attributi delle istanze. Ogni ipotesi è un vettore di 6 requisiti, che specifica il valore dei 6 attributi. Per ogni attributo l'ipotesi può essere:

- ?: qualsiasi valore è accettabile
- specifico valore: l'attributo deve avere quel valore
- $\emptyset$: nessun valore è accettabile

Dunque il nostro obiettivo è trovare le ipotesi g in G tali che $g(x) = 1$ per ogni istanza x.
L'ipotesi di apprendimento induttivo ci dice che ogni ipotesi trovata che approssima la funzione di target bene su un insieme di dati di training sufficientemente grande approssimerà anche la funzione target su dati non osservati. Il concept learning può essere visto come il processo di cercare in un grande spazio delle ipotesi implicitamente definite dalla rappresentazione delle ipotesi. L'obiettivo di questa ricerca è di trovare l'ipotesi che combacia meglio con gli esempi di training. Sia G lo spazio di ricerca, allora vogliamo trovare g che meglio "fitta" D. Riprendendo l'esempio di prima definiamo i requisiti $<?, \text{Cold}, \text{High}, ?, ?, ? >$. I possibili valori sono:

- **Sky:** Sunny, Cloudy, Rainy
- **AirTemp:** Warm, Cold, Mid
- **Humidity:** Normal, High
- **Wind:** Strong, Weak
- **Water:** Warm, Cold
- **Forecast:** Same, Change

Le possibili istanze sono $3 \times 3 \times 2 \times 2 \times 2 \times 2 = 288$
Le possibili ipotesi distinte sono $|G| = 5 \times 5 \times 4 \times 4 \times 4 \times 4 = 6400$ (bisogna considerare anche $\emptyset$ e ?), queste sono le ipotesi sintatticamente distinte.
Però dobbiamo rimuovere le possibilità che contengono $\emptyset$ in quanto non accettano nessun valore.
$|G| = 6400 - 5104 = 1296 = 4 \times 4 \times 3 \times 3 \times 3 \times 3$, che sono le ipotesi semanticamente distinte. Ciò viene descritto dalla formula $|G| = \prod_{i=1}^{n}(|D_i| + 1)$, dove D_i è l'insieme dei valori possibili per l'attributo i e n è il numero di attributi.
Data la visione del concept learning come problema di ricerca sorge naturalmente la questione dello studio di algoritmi efficienti per ridurre lo spazio delle ipotesi G.

Chapter 3

Feature Engineering

def. Trasformare i dati grezzi in feature significative, ovvero variabili che permettono al modello di capire meglio e modellare le relazioni nei dati. Idee di base:

- iniziare con delle feature
- processarle o trasformarle per creare delle nuove feature "ingegnerizate"
- usare queste nuove feature nel predittore

Il risultato ottenuto era soddisfacente? Ho migliorato le performance del modello?

- training del modello con le feature originali e validazione delle performance
- fare lo stesso con le feature nuove
- confrontare i risultati

3.1 Tipi di feature

3.1.1 Feature categoriche(nominali)

Prendono solo un numero finito di valori, ovvero $U = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, dove ogni α_i è una etichetta di categoria. Per riferirci a queste possiamo usare anche semplicemente gli indici i. Per trattare queste feature $U = \{1, 2, \ldots, k\}$ possiamo effettuare un **one-hot embedding**, $\phi(i) = e_i \in \mathbb{R}^k$.

3.1.2 Feature ordinali

I dati sono categorici, con un ordine. Ad esempio: {strongly disagree, disagree, neutral, agree, strongly agree}
Ci sono due strategie:

- Embeddare in $\mathbb{R}$, con valori $\{-2, -1, 0, 1, 2\}$. **N.B.** sto assumendo distanza unitaria e uniforme.
- Usare un embedding one-hot, come per le feature categoriche.

Quando usare l'una o l'altra? Nella prima soluzione utilizzo una sola colonna, nella seconda ne uso k. Uso la prima strategia quando necessito della concezione di ordine, dato che è esplicitato con questo metodo. Non usiamo la prima strategia quando la distanza tra i valori non è uniforme. La seconda strategia viene utilizzata quando non è necessaria la nozione di ordine (che viene persa).
Cosa succede se codifico con one-hot e calcolo la distanza euclidea (radice della somma dei quadrati delle differenze)? **Attenzione!** La distanza euclidea è costante.

3.2 Tipi di trasformazioni di feature

1. **Modfica di feature individuali**: rimpiazzo la feature x_i con una feature trasformata o modificata x_i^{new} (utile quando voglio portare sulla stessa scala)
2. **Combinazione di feature**: creo nuove feature combinando più feature

3. **Creazione di nuove features da più vecchie features**. Che problemi ha? Viene modificata la relazione che sto apprendendo.

Con il metodo 1 e 3 ho il problema di **curse of dimensionality**, ovvero il numero di campioni (necessari per avere la stessa accuratezza) cresce esponenzialmente con il numero di feature. Ma nella pratica il numero di esempi di training è fisso, dunque le performance in generale calano per alti numeri di features. In molti casi l'informazione che viene persa dalla rimozioni di variabili viene compensata da un miglior training in uno spazio di dimensione minore.

3.3 Feature reduction

In maniera ingenua possiamo pensare che più feature abbiamo, meglio è. In pratica, in molti casi, avere troppe feature può essere dannoso, vogliamo feature con un forte potere discriminante. Due strategie tipiche:

- **Feature selection**: selezionare un sottoinsieme delle feature esistenti, in modo da ridurre il numero di feature.

- **Feature extraction**: creare nuove feature a partire da quelle esistenti, in modo da ridurre il numero di feature. A partire da un insieme di feature $a = \{a_i | i = 1, \dots, n\}$, trovo un mapping $\hat{a} = f(a) : \mathbb{R}^n \to \mathbb{R}^m$, con $m < n$. Questo mapping deve essere fatto in modo da preservare la maggior parte delle informazioni e struttura presente in $\mathbb{R}^n$. Un **mapping ottimale** $\hat{a} = f(a)$ è quello che non risulta in un aumento della minima probabilità di errore. Un esempio di trasformazioni lineari è la principal component analysis.

3.4 Principal Component Analysis (PCA)

L'idea principale è quella di approssimare ogni dato x con una combinatone lineare di un numero ridotto di "vettori", chiamati **componenti principali**. Questi vettori formano il miglior sottospazio che fitta i dati. Il numero di componenti principali M è la nuova dimensione ridotta. Più formalmente:

- Siano $\phi_1, \dots, \phi_r \in \mathbb{R}^d$, inoltre $\Phi = [\phi_1, \dots, \phi_r]$ è una matrice $d \times r$.

- Le loro combinazioni lineari formano un sottospazio $S = \subseteq \mathbb{R}^d$, dove $S = \{\Phi a | a \in \mathbb{R}^r\}$

- La distanza da un punto x a S è definita come:

$$dist(x, S) = \min_a ||x - \Phi a||_2$$

- Il punto più vicino a x del sottospazio S è chiamato **proiezione** di x su S, ed è indicato con $\hat{x}$.

$$\hat{x} = \Phi(\Phi^T \Phi)^{-1} \Phi^T x$$

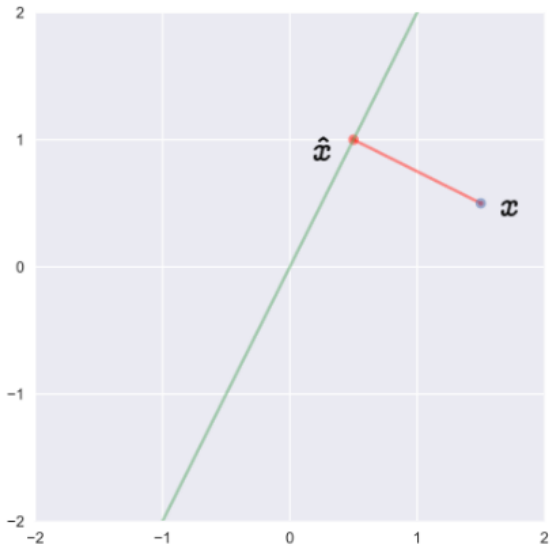


Figure 3.1: Proiezione di un punto su un sottospazio

- Ogni osservazione x viene assunta come "vicina" a una combinazione lineare di $\{\phi_1, \ldots, \phi_r\}$.
- Le colonne di Φ sono chiamate **componenti principali**.
- Il parametro $r < d$ è detto rango di riduzione, o dimensione del sottospazio.
- La funzione di loss associata è la distanza quadratica media tra i punti e le loro proiezioni:

$$l(\Phi) = \frac{1}{n}||x - \hat{x}||_2^2$$

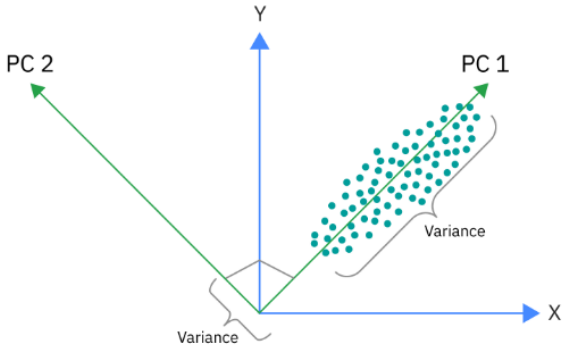


Figure 3.2: Esempio di PCA in 2D

Un nuovo possibile sistema di coordinate (quello verde in figura 3.2) è quell dove gli assi sono uguali agli autovettori della matrice di covarianza del dataset. Adesso immaginiamo di poter ruotare gli assi, in modo che nel nuovo sistema di coordinate la matrice di covarianza sia:

$$Cov(x, y) = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

In questo modo le due variabili non sono correlate tra loro.

Term	Expression	Description
Covariance matrix	$\Sigma = \frac{1}{n} X^T X$	Describes the variability of the data
Eigenvector	ϕ_i	Principal direction (principal component)
Eigenvalue	λ_i	Variance explained along ϕ_i
Component matrix	$\Phi = [\phi_1, \ldots, \phi_r]$	Basis of the reduced subspace
Projection	$\hat{x} = \Phi\Phi^T x$	Point projected onto the subspace

Table 3.1: Summary of key concepts in Principal Component Analysis (PCA).

Quindi la PCA è una trasformazione lineare che proietta il dataset in un nuovo sistema di coordinate, dove la prima coordinata ha più varianza, ogni coordinata successiva ha varianza decrescente e tutte le coordinate sono ortogonali tra loro.
In pratica trasformiamo un insieme di x variabili correlate in un insieme di p componenti principali non correlate.

1. Creare la matrice X di dimensione $K \times N$ dei dati, dove ogni riga è un'osservazione e ogni colonna una feature.
2. Calcolare la matrice di covarianza S, basata su X. Questo cattura la ridondanza del dataset.
3. Risolvere $S\underline{e} = \lambda\underline{e}$ per trovare gli autovettori $\underline{e}$ e autovalori λ di S. L'autovettore è una direzione, l'autovalore ci dice quanta varianze c'è nei dati in quella direzione.
4. Risolvere $P = X\underline{e}$ per calcolare le componenti principali (N componenti)

Tecnicamente, la quantità di varianza catturata dalla i-esima componente principale è misurata dall'auto valore λ_i. La PCA è particolarmente utile quando le variabili nel dataset sono altamente correlate. La correlazione indica che c'è ridondanza nei dati. A causa della ridondanza, la PCA può essere usata per ridurre le variabili originali in un numero minore di variabili (componenti principali), spiegando la maggior parte della varianza delle variabili originali.
Gli autovalori possono essere usati per determinare il numero di PC da mantenere dopo la PCA.

- Un autovalore > 1 indica che le PC contano di più per la varianza che una delle variabili originali.
- In maniera alternativa, possiamo limitare il numero di componenti al numero che spiega una certa percentuale della varianza totale.

Se voglio ad esempio spiegare almeno il 70% dell'esempio in figura 3.3, scelgo le prime 3 componenti.

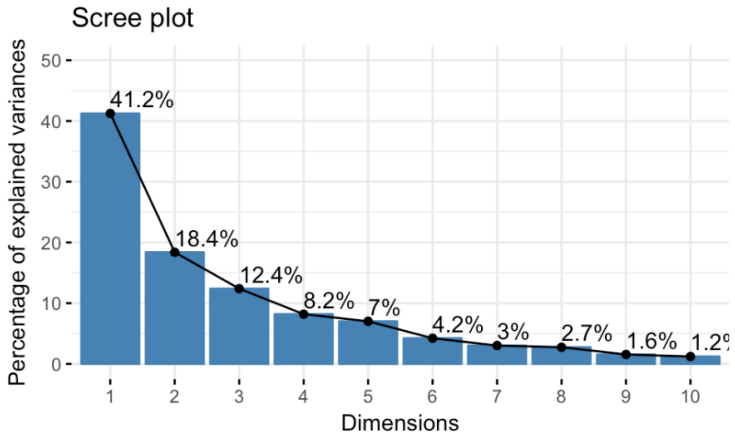


Figure 3.3: Varianza spiegata dalle componenti principali