Intelligenza Artificiale

Emanuele Fittipaldi

2021

Qualunque tecnologia sufficientemente avanzata è indistinguibile dalla magia.

—Arthur C. Clarke

Contents

List of Figures

1 Introduzione al Machine Learning

L'AI può essere usata per raggiungere diversi obiettivi. Grazie all'AI possiamo **ridurre il tempo di programmazione**. Per esempio, se vogliamo creare un programma che ci permetta di effettuare la correzione dello spelling, potremmo creare un programma specificando tutte le regole grammaticali. Questo non è particolarmente efficiente. Potremmo invece creare un algoritmo di machine learning al quale dare in pasto degli esempi di grammatica scorretta e in poco tempo esso sarà in grado di rilevare errori grammaticali. Se avessimo creato un programma codificando tutte le regole grammaticali, ci sarebbe stato anche il problema della non generalità di esso. Infatti il tutto avrebbe funzionato soltanto per una sola lingua. Grazie al machine learning invece, passare da una lingua all'altra è solo questione di collezionare errori grammaticali di un'altra lingua e darli in input allo stesso modello di machine learning creato in precedenza. Quindi il machine learning ci permette di **Personalizzare e scalare i nostri prodotti**. Infine, il machine learning ci permette di **risolvere dei problemi che sembrano apparentemente irrisolvibili**.

2 Supervised Machine Learning

2.1 Label e Features

Nel machine learning supervisionato, creiamo dei modelli che combinano gli input per produrre delle **predizioni** utili anche su dati non visti in precedenza. Quando **alleniamo** un modello, gli forniamo delle label associate a delle features.

- La **label** corrisponde alla variabile oggetto della predizione e tipicamente viene rappresentata dalla variabile **y**. Nel caso di un sistema di filtraggio email, la label potrebbe essere 'Spam' o 'Non Spam'.
- Le **features** sono le variabili di input che descrivono i nostri dati e tipicamente sono rappresentate dalle variabili $\{x_1, x_2, ..., x_n\}$. Nel caso del sistema di filtraggio email, le features le potremmo derivare dalle email stesse. Un esmepio di feature potrebbe essere il numero delle parole contenute nella email, l'indirizzo del mittente o del ricevente, ecc...

2.2 Labeled e Unlabeled

Se prendiamo una particolare istanza di un dato **x** esso potrebbe essere

- Un **dato labellato** - questo indica che il dato è rappresentato da una coppia $\{features, label\}$ ovvero, usando

le lettere. (**x**,**y**). Riconducendoci al sistema di filtraggio email, un dato così composto potrebbe essere rappresentato da un pezzo di email e dalla label 'Spam' o 'Non Spam'. I Dati labellati vengono usati in fase di allenamento del modello e come vedremo, per testare le predizioni del modello.

- Un **dato non labellato** - questo indica che il dato è rappresentato soltanto dalla $\{features, ?\}$ ovvero, usando le lettere (**x**,**?**). Questi dati sono i soggetti delle predizioni per determinare le label y mancanti. Una volta che abbiamo determinato la label, sia essa 'Spam' o 'Non Spam' abbiamo classificato il dato, e possiamo mettere dunque tale email nella cartella giusta.

Per completare questa introduzione alla terminologia del machine learning, definiamo meglio cosa è il modello. Il **modello** definisce la relazione tra le features e le lables e permette di predire le label **y'**. Esso è definito da parametri interni che sono appresi/calibrati nel processo di allenamento (Training) dove il modello matematico viene appreso (learning). Questo significa mostrare al modello esempi labellati, permettendo al modello di imparare gradualmente la relazione tra le features e le labels. Dopo che il modello matematico è stato appreso avviene la fase di **inferenza**. In questa fase applichiamo il modello allenato ad esempi non labellati per labellarli, ovvero per definire la loro **y'**.

2.3 Regressione vs. Classificazione

Una modello di **regressione** predice valori continui. Per esempio, un modello di regressione fa delle predizioni che rispondono a domande del tipo: Quale è il valore di una casa in California? Quale probabilità c'è che un utente clicchi su questa

pubblicità?. Un modello di **classificazione** invece predice valori discreti. Per esempio, esso fa delle predizioni che rispondono a domande del tipo: Questa email è Spam o Not Spam? Questa immagine è di un cane, gatto o criceto?

3 Regressione Lineare

La **regressione lineare** è un metodo per trovare una linea retta o un iperpiano che meglio descrive l'andamento dei punti. I punti rappresentano i dati contenuti nel dataset.

NB: Parliamo di retta se ci troviamo in uno spazio 2D, mentre parliamo di iperpiano se ci troviamo in uno spazio N-Dimensionale.

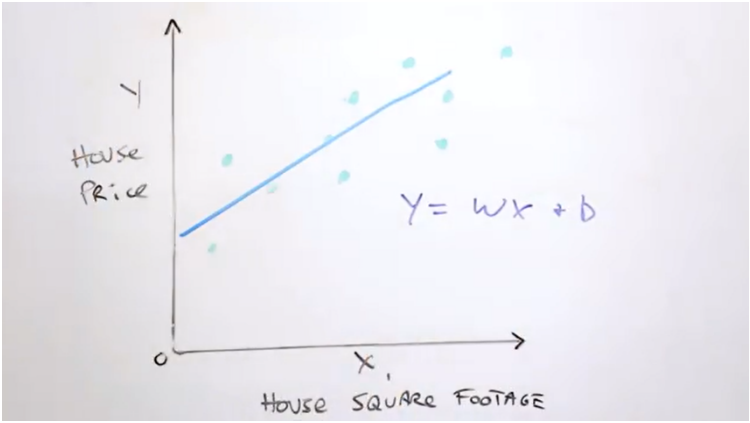

Figure 3.1: Regressione Lineare

Prediamo in considerazione questo esempio. Questa è la rappresentazione del dataset relativo ai prezzi delle case in California. Sull'asse delle **y** Troviamo la label 'House Price' e sull'asse delle **x** troviamo la feature 'House Square Footage'. Tutti i punti

blu, sono i dati presenti nel dataset rappresentati da coppie *(House Price, House Square Footage)*. Ad un certo valore di *House Square Footage* corrisponde un certo valore di *House Price*. A partire da questi dati, vogliamo disegnare una retta che descriva l'andamento di questi dati. Questa linea è il nostro modello e tramite essa possiamo predire l'Housing Price avendo in input soltanto House Square Footage. Se abbiamo una retta infatti, basta proiettare l'*House square footage* sulla retta e vedere il valore di *House price* corrispondente. Come sappiamo dall'algebra, una retta è definita come $y = Wx, + b$. Abbiamo scritto W invece di M per definire il coefficiente angolare perchè W sta per **weight**, ma hanno esattamente lo stesso significato. C'è inoltre una virgola per indicare che potremmo essere in più di una dimensione, ovvero avere più features (x) con associati pesi diversi (w). B invece è l'intercetta sull'asse delle y. Di conseguenza, spesso nell'ambito del machine learning, l'equazione della retta viene riscritta così $y' = b + w_1 x_1$. Dove:

- y' è la label (l'output, la predizione).
- b è il bias (y-intercetta), spesso scritta come w_0.
- w_1 è il peso della feature 1. Peso è lo stesso concetto di "Pendenza" m nella equazione della retta tradizionale.
- x_1 è la feature (l'input conosciuto).

Per **inferire** (predire) y' noi andiamo semplicemente a sostituire x_1 in questo modello, avendo scelto dei valori per b e w arbitrariamente. Così facendo, otteniamo un numero reale, ovvero la nostra predizione. Anche se stiamo facendo un esempio su un dataset dove ogni dato dispone soltanto di una feature, possono essere creati dei modelli più sofisticati con molte features, ognuno provvista di un proprio peso ($w_1, w_2, etc.$). Un modello che si basa su tre features per esempio è fatto così:

$$y' = b + w_1 x_1 + w_2 x_2 + w_3 x_3$$

3.1 Loss

Come sappiamo però se la retta che abbiamo disegnato è una buona retta? Lo possiamo sapere attraverso la misura di **Loss**. La Loss ci dice in sostanza quanto bene la nostra retta è brava a predire, dato un qualsiasi dato di cui si conoscono soltanto le features e non la label. Per saperlo quindi abbiamo la necessità di avere un dataset labellato, dove poter fare delle predizioni sui singoli dati per poi verificare quanto la predizione si discosta dal valore reale. Il calcolo della Loss infatti prevede l'osservazione della differenza tra la predizione di un certo valore e il suo valore reale.

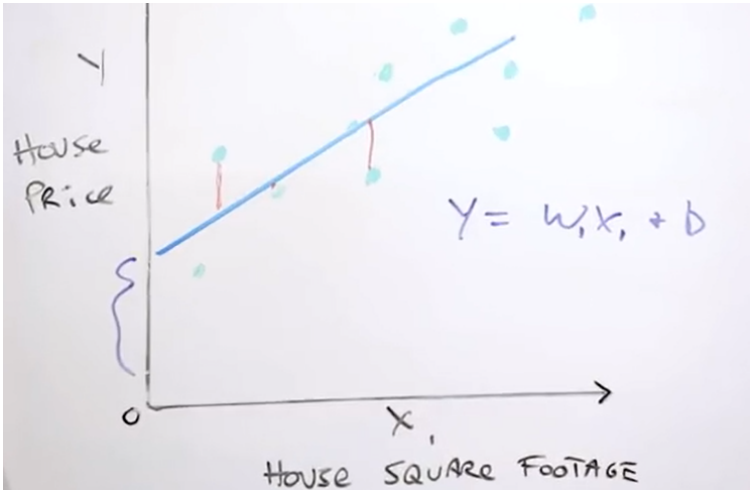

Figure 3.2: Loss - intuizione

Valori di Loss vicino allo 0 ovviamente sono preferiti.

3.1.1 L2 Loss

La misura per stabilire la distanza tra il valore predetto e il valore reale può essere definita in diversi modi. Ogni modo diverso da vita a differenti tipi di Loss. Quella che tratteremo è la misura di Loss più facile con la quale iniziare.

La Loss più conosciuta è la L_2 *Loss* chiamata anche errore quadratico. Si chiama in questo modo perchè per calcolarla basta fare $(y - y')^2$ ovvero la differenza tra il valore predetto e il valore reale al quadrato. Ovviamente, man mano che ci allontaniamo dal valore reale, l'errore quadratico aumenta. Quando andiamo ad allenare il modello, non ci importa minimizzare la Loss soltanto su un dato, ma benssì vogliamo minimizzare la Loss sull'intero dataset. Dunque formalmente vogliamo minimizzare l'errore quadratico medio (MSE):

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - prediction(x))^2$$

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - y')^2$$

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - (b + w_1 x_1))^2$$

MSE stà per Mean square error. Essa è la media degli scarti quadratici. Dove:

- (x, y) è un dato dove
  - x è una o un insieme di features, che il modello usa per fare le predizioni.
  - y è la label del dato

- $prediction(x)$ è l'equazione della retta che abbiamo visto prima, $y' = b + w_1 x_1$ la quale ci da appunto una predizione/label y'.
- D è un data set contenente diversi dati labellati rappresentati come coppie (x, y).
- N è il numero dei dati in D

Sebbene MSE è comunemente usata nel machine learning, essa non è ne la l'unica funzione di loss praticabile e nemmeno la migliore per tutte le circostanze.

3.1.2 Training e Loss

Allenare un modello significa apprendere quali sono dei buoni valori per tutti i pesi e per l'intercetta, a partire dai dati labellati presenti nel dataset. Nell'apprendimento supervisionato, un algoritmo di machine learning costruisce un modello esaminando diversi esempi e tentando di trovare il modello che rende la loss minima. Questo processo si chiama **minimizzazione empirica del rischio**.

Loss è un numero che indica quanto cattiva era la predizione su un singolo dato. Se la predizione fatta dal modello è perfetta, la loss è 0. Questo significa che y e y' coincidono. L'obiettivo nell'allenare un modello è trovare un insieme di pesi e di intercette che rendono la loss bassa, in media, su tutti i dati. Per esempio in questa figura mostriamo un modello con una loss alta (sulla sinistra) ed un modello con una loss bassa (sulla destra). Le frecce rosse rappresentano la loss, le rette blu invece le predizioni.

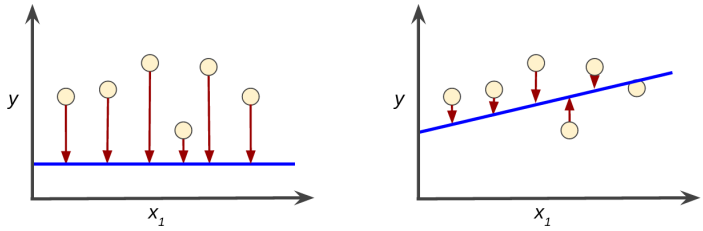

Figure 3.3: Loss alta vs. Loss Bassa

3.2 Gradient Descent

Per allenare un modello, abbiamo bisogno di un buon modo per ridurre la loss. L'approccio largamente usato è quello iterativo. Esso risulta semplice ed efficiente. Per trovare la loss minima, abbiamo bisogno di trovare un insieme di iperparametri che la minimizza. Nel caso della retta significa trovare dei valori per l'intercetta b e per il coefficiente angolare w che renda MSE minimo. Questi iperparametri li dobbiamo cercare all'interno dello **spazio degli iperparametri**, ovvero l'insieme dei possibili valori assumibili dai nostri iperparametri rispetto all'errore quadratico medio (MSE). Piuttosto che muoverci in questo spazio senza una meta, sarebbe comodo avere una guida che ci indichi dove andare per trovare un insieme di iperparametri che ci dia una loss più bassa di quella attuale.

Uno dei modi per ottenere questa guida è computare il **gradiente**, definito come la derivata della funzione di loss calcolata su tutti i punti del dataset. Una derivata positiva ci dirà che proseguendo in quella direzione, MSE aumenterà, mentre una derivata negativa ci dirà che proseguendo in quella direzione

invece MSE diminuirà. Quando la derivata vale 0 o un valore prossimo ad esso, allora significa che abbiamo raggiunto il punto di minimo. Le coordinate di questo punto ci danno i valori da assegnare all'intercetta e ai vari pesi per ottenere l'iperpiano che rende MSE minima. Dato che la direzione del gradiente indica sempre il verso in cui la funzione cresce, noi ci muoviamo nel verso opposto ad essa. La curva che descrive la relazione tra la loss e gli iperparametri dell'iperpiano da luogo ad un grafico **convesso**. Il fatto che sia convesso è molto importante perché ci dice che esiste un solo punto minimo che è eventualmente possibile raggiungere. Il Gradient Descent è un approccio **iterativo**, dunque quello che facciamo è fare dei piccoli step, ripetutamente, nella direzione che minimizza la loss. Questi step li chiamiamo **Step del gradiente**. Ad ogni step del gradiente otteniamo dei nuovi valori per l'intercetta b e per i pesi $w_1, ..., w_n$, i quali, applicando dinuovo la funzione di loss, ci restituiranno un nuovo valore per MSE e calcolando la derivata di questa funzione scegliamo la nuova direzione dove dirigerci. Questa strategia iterativa si chiama **Discesa del gradiente** perchè usiamo il gradiente per scendere fino al punto di minimo di questa curva a forma di coppa. Di solito, si itera finchè la loss complessiva smette di cambiare o cambia estremamente poco. Quando questo accade, diciamo che il modello ha raggiunto la **convergenza**.

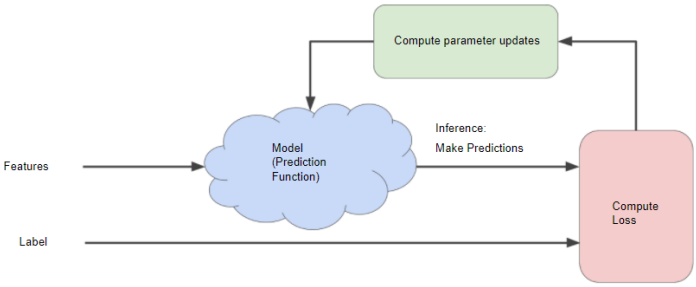

Figure 3.4: Gradient Descent - Walkthrough

Questo approccio iterativo consta in sostanza dei seguenti passi:

1. Input di tutti i dati
2. computiamo il gradiente derivando la funzione di loss applicata a tutti i dati.
3. Il gradiente negativo ci dice in che direzione ci dobbiamo muovere per aggiornare i parametri del modello e diminuire la loss.
4. Facciamo un passo in quella direzione e abbiamo così una nuova versione del modello sul quale possiamo ricalcolare il gradiente, reiterando.

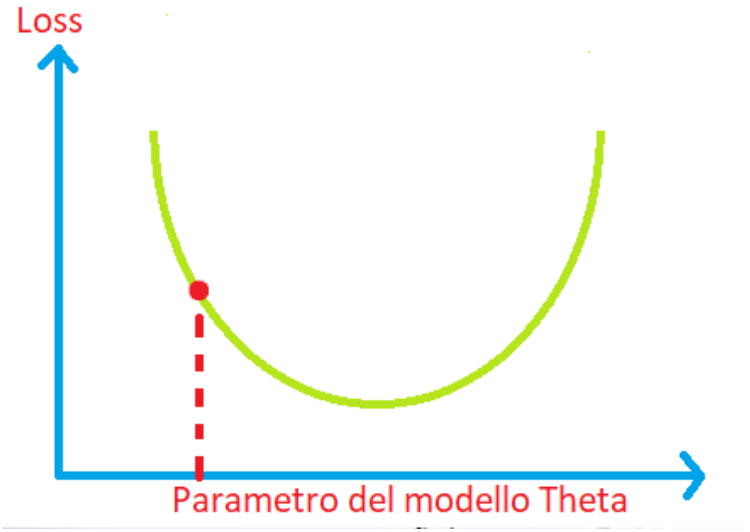

Figure 3.5: Gradient Descent

Prendiamo questo esempio, dove il parametro da ottimizzare per ridurre la loss è soltanto uno, Theta. In questa rappresentazione vediamo come è mappato il parametro Theta con la loss. Se iniziamo con un valore casuale di Theta, otteniamo una certa loss. Computando il gradiente negativo capiamo in che direzione fare lo step del gradiente. Una volta fatto uno step in quella direzione otteniamo una nuova loss. Man mano che facciamo sempre più step si arriverà ad un punto in cui abbiamo passato il minimo locale in cui il gradiente negativo ci dirà di tornare indietro nella direzione da cui siamo venuti.

3.2.1 Learning Rate

L'ampiezza di questi step del gradiente la decidiamo noi. Essa prende il nome di **learning rate** ovvero tasso di apprendimento.

- Step piccoli - sono richiesti molti step per raggiungere il minimo insieme a molto più tempo computazionale
- Step grandi - si rischia con uno step di andare troppo oltre il punto di minimo locale. Questo comporta la possibilità di ottenere una loss peggiore di quella attuale e anche di incappare in un loop, andando avanti ed indietro all'infinito, non potendo arrivare al minimo.

3.2.2 Problemi convessi e non convessi

Per i problemi convessi sappiamo che la loro rappresentazione ricorda la forma di una coppa. Questo ci dice che:

- c'è un solo minimo
- Non importa il valore iniziale dei pesi. finchè iniziamo a muoverci lungo questa coppa, adoperando degli step di dimensione ragionevole e seguendo il gradiente negativo, saremo eventualmente in grado di giungere a questo minimo.

Purtroppo, molti problemi di machine learning non sono convessi. Per esempio le reti neurali sono notoriamente Non convesse. Non convesso significa che:

- C'è più di un solo minimo locale, e di conseguenza un solo minimo globale.
- C'è una forte dipendenza sui valori iniziali dei pesi
- Il problema ha una forma come delle montagne russe.

3.2.3 Stocastic Gradient Descent & Mini-Batch Gradient Descent

Quando computiamo il gradiente della funzione di loss, la matematica ci suggerisce che dovremmo calcolare il gradiente su tutti i dati del nostro dataset e nel Gradient Descent si procede così. Questo è l'unico modo per garantire che i nostri step del gradiente avengono nella direzione giusta. Però, se abbiamo un data set con milioni o miliardi di dati, questo significherebbe molto tempo di computazione per performare ogni step all'interno dello spazio degli iperparametri per trovare una nuova configurazione. Empiricamente è stato trovato che invece di usare l'intero dataset, se si calcola il gradiente della funzione di loss su un singolo dato random, o su un sottoinsieme random del dataset, questo funziona lo stesso, anche se nel complesso sono richiesti più passi. Il carico computazionale risulta minore. Questa viene chiamata la **discesa del gradiente stocastica** nel caso del calcolo del gradiente sulla base di un solo dato, mentre **discesa del gradient mini-batch**, se usiamo un piccolo sottoinsieme del dataset. La Stocastic Gradient Descent è utile se il dataset ha molti dati ridondanti, ma generalmente è preferibile usare il Mini-Batch Gradient Descent.

Nota: Il termine "Stocastica" sta ad intendere che il singolo dato, o i dati contenuti nel sottoinsieme del dataset preso in considerazione, sono presi in modo casuale.

3.2.4 Note sul gradiente

Nota che il gradiente è un vettore, quindi ha le stesse caratteristiche di un vettore:

- una direzione
- una magnitudine

Cerchiamo di chiarire meglio questo ultimo punto. Un gradiente punta sempre nella direzione di crescita maggiore della funzione di loss. L'algoritmo di discesa del gradiente fa i passi nella direzione del gradiente negativo in modo da ridurre la loss il più velocemente possibile. Per determinare il prossimo punto lungo la curva della funzione di loss, l'algoritmo di discesa del gradiente moltiplica il gradiente per uno scalare chiamato **learning rate** (chiamato a volte anche **step size** (ampiezza del passo)) per determinare il prossimo punto. Per esempio, se la magnitudine del gradiente è 2.5 e il learning rate è 0.01, allora l'algoritmo di discesa del gradiente prenderà il prossimo punto ad una distanza di 0.025 dal punto precedente. Questo passo che facciamo è, come abbiamo già visto, lo step del gradiente. La tecnica della discesa del gradiente ripete questo processo, avvicinandosi sempre di più al minimo.

Nota: Quando performiamo la discesa del gradiente, generalizziamo il processo appena descritto per calibrare tutti i parametri del modello contemporaneamente. Per esempio, per trovare il valore ottimale sia per w_1 che per b, calcoliamo il gradiente rispetto sia W_1 che b. In seguit, modifichiamo i valori sia di W_1 che di b in base ai rispettivi gradienti. Poi ripetiamo questi step finchè non raggiungiamo la loss minima.

4 TensorFlow

4.1 Intro a TensorFlow

TensorFlow è una piattaforma open source end-to-end per il machine learning. TensorFlow è un sitema molto ricco per gestire tutti gli aspetti di un sistema di machine learning; tuttavia, questo corso si focalizza soltanto su una particolare API di TensorFlow per sviluppare ed allenare i modelli di machine learning.

Le API di TensorFlow sono organizzate in modo gerarchico, con le API di più alto livello costruite sopra quelle di più basso livello. I ricercatori nel campo del machine learning usano le API di basso livello per creare ed esplorare nuovi algoritmi di machine learning. In questo corso, useremo invece soltanto una API di alto livello chiamata **tf.keras** per definire ed allenare i modelli di machine learning e fare predizioni. tf.keras è la variante di TensorFlow dell'API open-source Keras.

La seguente figura mostra la gerarchia dei toolkit di TensorFlow:

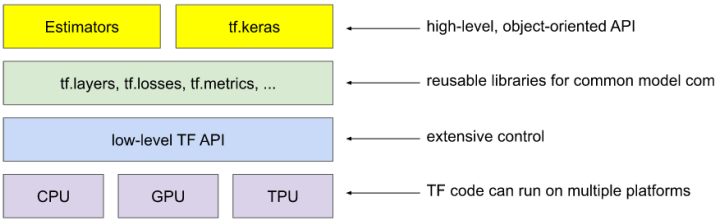

Figure 4.1: TensorFlow - Gerarchia

A questo punto ci sono i due esercizi sulla regressione lineare fatti con tf.keras

5 Overfitting

La **generalizzazione** del modello è la sua abilità di addattamento a nuovi dati non visti in precedenza, prelevati dalla stessa distribuzione usata per creare il modello.

5.1 Overfit

Quello che vogliamo fare è usare i dati che abbiamo nel dataset per allenare il modello, in modo tale che se prendiamo un altro dato mai visto dal modello, esso sia in grado di classificarlo correttaemente. Se alleniamo il modello su tutti i dati presenti nel dataset, il modello diventa troppo complesso, ovvero descriverà troppo accuratamente le caratteristiche dei dati presenti. Esso performerà bene sui dati su cui si è allenato, ma molto male sui dati che non ha mai visto (**overfitting**). La soluzione è utilizzare il principio del **Rasoio di Ockham** il quale dice che un modello dovrebbe essere il più semplice possibile. Nella pratica per evitare di creare un modello troppo complesso e quindi di andare in overfitting, dividiamo il dataset in due porzioni. Una porzione chiamata **Training set** dove usiamo questi dati per allenare il modello, ed una porzione chiamata **Test set**. Per valutare il potere di predizione del modello, usiamo i dati presenti nel Test set, i quali sono dati che il modello non ha incontrato in fase di allenamento. Questi dati, avendoli presi dal dataset sono labellati e di conseguenza possiamo confrontare le predizioni del modello su ciascun dato, rispetto al valore reale della

propria label. Se il modello dimostra di avere un buon potere predittivo, allora questo ci da fiducia che esso sarà in grado di predire accuratamente dei valori a partire da un dato mai visto in precedenza.

Ci sono da sottolineare però delle cose:

1. Estraiamo dei dati **indipendentemente ed identicamente (i.i.d)** in modo radom dalla distribuzione. Questo significa che i dati non si influenzano l'un l'altro.

2. La distribuzione è **stazionaria**, ovvero la distribuzione non cambia all'interno del dataset.

3. Estraiamo i dati sempre dalla **stessa distribuzione**: inclusi gli insiemi di training, validazione e test.

Nella pratica, a volte violiamo queste assunziooni. Per esempio:

- Considera un modello che sceglie le pubblicità da mostrare. L'assunzione i.i.d. sarebbe violata se il modello basa le scelte delle pubblicità, in parte, su quali pubblicità l'utente ha visto in precedenza.

- Considera un dataset che contiene delle informazioni sulle vendite di un anno. Gli acquisti di un utente cambiano stagionalmente, il che potrebbe violare la stazionarità.

Quando sappiamo qualunque delle precedenti tre assunzioni di base venga violata, dobbiamo prestare molta attenzione alle metriche.

5.2 Training e Test set

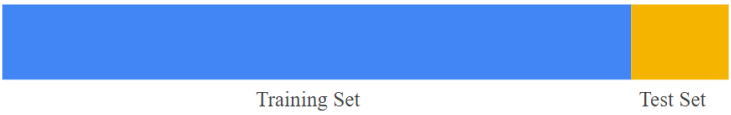

Figure 5.1: Suddivisione del dataset

Quando dividiamo il dataset in training set e test set dobbiamo assicurarci che il test set soddisfi due condizioni:

1. Esso sia grande abbastanza per portare a dei risultati statistici significativi.
2. E' rappresentativo del dataset nell'insieme. In altre parole, non bisogna prendere un test set con delle caratteristiche differenti rispetto il training set.

Mai allenare il modello sui dati di test. Se vediamo dei risultati troppo buoni nelle metriche di valutazione, potrebbe essere un segno che abbiamo fatto del training sul test set. Per esempio, una accuratezza alta potrebbe indicare che i dati di test sono entrati in qualche modo nel training set.

Per esempio, considera un modello che predice se una email è spam o non spam, usando l'oggetto dell'email, il suo corpo e l'indirizzo del mittente come features. Facciamo dunque uno splitting del dataset in training set e test set nelle percentuali di 80-20. Dopo l'allenamento, il modello raggiunge una precisione del 99% sia sul training set e sia sul test set. Noi ci aspetteremmo invece una precisione minore sul test set, quindi diamo un'altra occhiata ai dati e scopriamo che molti dati nel test set sono dei dati duplicati anche nel training set. Inavvertitamente, abbiamo allento il modello su alcuni dei dati di test, e

come risultato, non stiamo più misurando accuratamente quanto il nostro modello si generalizza su nuovi dati.

5.3 Validation set

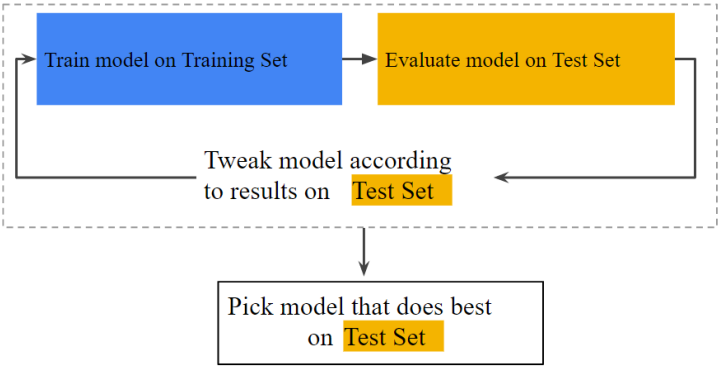

Figure 5.2: ciclo di test e training

Partizionare un dataset in test set e training set ci permette di giudicare se il modello generalizzerà bene su nuovi dati. Tuttavia, usando soltanto due partizioni potrebbe portare a il modello a soffrire di overfitting. Questo accade perchè abbiamo la necessità di effettuare delle modifiche negli iperparametri e andiamo ad allenare il modello in continuazione. Così facendo però, iniziamo ad over-fittare le peculiarità dei dati presenti nel test set.

La soluzione è creare una terza partizione nel dataset chia-

Figure 5.3: suddivisione del dataset in test, training e validation set

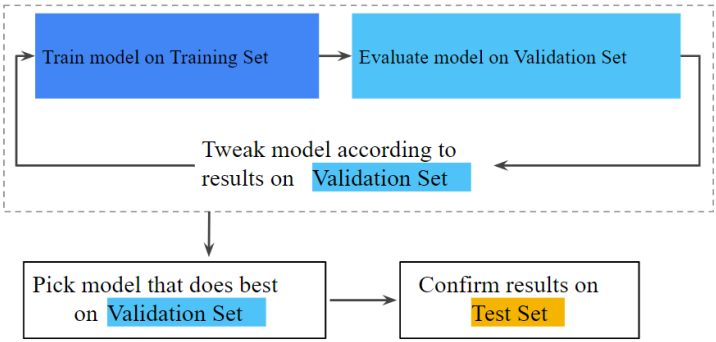

Figure 5.4: ciclo di allenamento, validazione e testing

mata **validation set**. L'approccio iterativo di apprendimento condotto dal modello cambia leggermente. Quando iteriamo, alleniamo il modello sul training set e valutiamo il potere predittivo solo sul validation set, tenendo da parte il test set completamente inutilizzato. Solo così possiamo essere sicuri che il modello non abbia davvero mai visto questi dati. In questo modo siamo liberi di effettuare tutte le modifiche che vogliamo agli iperparametri finchè non otteniamo dei buoni risultati sul validation set.

Soltanto alla fine, andiamo ad usare i dati nel test set per misurare il reale potere predittivo del modello assicurandoci che le performance ottenute sui dati di test siano uguali a quelle ottenute sui dati di validazione. Se non è così, questo è buon segnale che forse stavo over-fittando il validation set.

Esercizio di programmazione: a questo punto c'è un piccolo esercizio di programmazione

6 Rappresentazione

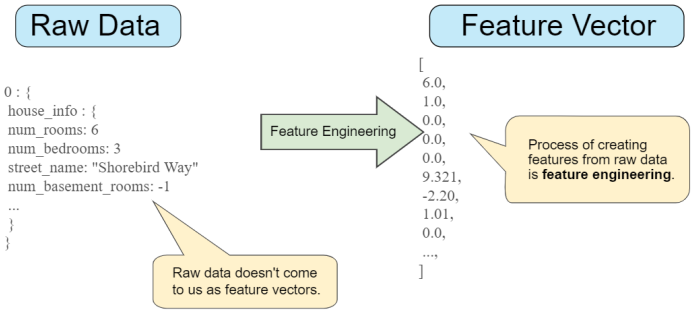

Figure 6.1: rappresentazione dei dati

Un modello di machine learning non può usare i dati grezzi in input così come sono. Dobbiamo **rappresentare** i dati per permettere al modello di poterci lavorare su.

Quando lavoriamo con dei dati provenienti dal mondo reale, essi non sono già rappresentati come dei **vettori di features** formattati. Bensì essi ci perevengono sotto diverse forme. Il processo di estrazione delle features a partire dai dati grezzi si chiama **feature engineering**. Questo è il punto critico dove la maggior parte degli sviluppatori spende all'incirca il 75% del tempo. Molti modelli di machine learning devono rappre-

sentare le features come dei valori reali dato che questi valori devono essere moltiplicati con i pesi del modello.

6.1 Mappare valori numerici e valori categorici

Se i dati grezzi sono interi o numeri reali, tali numeri non richiedono delle codifiche speciali perché possono già essere moltiplicati per i pesi del modello. Possono essere già considerati i valori delle features. Cioè se abbiamo il campo *num_rooms:6* il valore della feature è balanmente *6.0*. Se invece troviamo un valore espresso come stringa, il discorso cambia. Dato che non possiamo moltiplicare delle stringhe per i pesi del modello, dobbiamo fare feature engineering per convertire le stringhe in valori numerici.

6.1.1 Primo tentativo di conversione da Stringhe a numero

Per convertire una stringa in un valore numerico, la cosa più naturale che verrebbe da fare è associare ad ogni vocabolo unico, un valore numerico. Per esempio, se consideriamo una feature chiamata *street_name* che può assumere i seguenti valori:

- 'Charleston Road'
- 'North Shoreline Boulevard'

- 'Shorebird Way'
- 'Regstorff Avenue'

possiamo assegnare ad ogni strada un numero intero:

- 'Charleston Road' al valore 0
- 'North Shoreline Boulevard' al valore 1
- 'Shorebird Way' al valore 2
- 'Regstorff Avenue' al valore 3
- Tutti gli altri termini fuori dal vocabolario a 4

Tuttavia se incorporiamo questi indici direttamente nel nostro modello, questo imporrà alcuni vincoli che potrebbero essere problematici:

- Apprenderemo un singolo peso che si applica a tutte le città. Per esempio, se apprendiamo un peso di 6 per *street_name*, lo moltiplicheremo per 0 se consideriamo Charleston Road, per 1 se consideriamo North Shoreline Boulevard, 2 per Shorebird Way e così via. Considera un modello che predice il valore di una casa usando *street_name* come feature. E' poco verosimile che ci sia una relazione lineare tra il prezzo di una casa e il nome di una strada, e inoltre questo assume che le strade siano state ordinate in base al costo di una casa in media. Il nostro modello richiede la flessibilità di poter apprendere pesi diversi per ogni strada che saranno aggiunti al prezzo stimato usando le altre features.
- Non stiamo prendendo in considerazioni i casi in cui *street_name* potrebbe assumere valori multipli. Per esempio, molte

case sono collocate all'angolo di due strade, e non c'è modo di codificare questa informazione in *street_name* se possiamo fare affidamento su un solo indice.

Per rimuovere entrambi questi vincoli, possiamo invece creare un vettore binario per ogni feature categorica nel nostro modello.

6.1.2 One-hot encoding

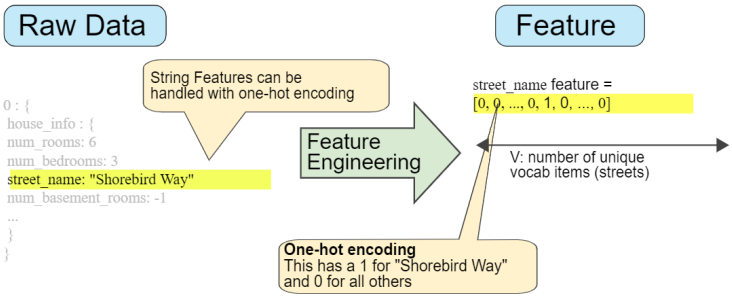

Figure 6.2: One-hot encoding

Possiamo costruire per esempio un feature vector a partire da esso usando l'**hot-encoding**. L'hot encoding prende nota di tutti gli N vocaboli diversi che sono stati incontrati e associa ad ognuno di esso un vettore ad N componenti, con tutti 0 tranne un solo 1. In tal modo ogni vocabolo diverso è codificato come un vettore. Questo approccio crea efficacemente una variabile booleana per ogni valore della feature (per es.*street_name*). Quindi, se una casa per esempio è in Shorebird Way, allora il valore binario sarà 1 soltanto per Shorebird Way. In questo

modo, il modello usa soltanto il peso per Shorebird Way. Similmente, se una casa è ad un angolo dove si incrociano due strade, allora due valori binari sono settati ad 1, e il modello usa entrambi i loro rispettivi pesi. Il One-hot encoding si estende anche ai dati numerici che non vogliamo moltiplicare direttamente per un peso del modello, come per esempio il codice postale.

6.1.3 Sparse Representation

Supponiamo di avere 1,000,000 di nomi di strade differenti nel nostro dataset che vogliamo includere come valori per *street_name*. Creare esplicitamente un vettore binario con 1,000,000 elementi dove soltanto 1 o 2 elementi sono true è una rappresentazione molto inefficiente sia in termini di spazio che di tempo computazionale quando si tratta di processare questi vettori. In questa situazione, un approccio comune è usare una rappresentazione sparsa, dove soltanto i valori diversi da 0 sono salvati. Nelle rappresentazioni sparse, viene ancora appreso un peso del modello indipendente per ogni valore di feature, come descritto sopra.

6.2 Cosa rende una feature una buona feature?

Ci sono delle regole generali:

- Ogni feature dovrebbe comparire con un valore diverso da 0 almeno un numero ragionevole di volte nel nostro dataset. Se una feature compare con un valore diverso da 0 raramente, o soltanto una volta, è probabile che non sia

una buona feature da usare e che dovrebbe essere filtrate nei passi di pre-processing.

- Le nostre features dovrebbero avere un significato chiaro ed ovvio. Questo ci permette di fare facilmente del debugging e di verificare che tutto sia ok, per renderci sicuri che le nostre features sono processate correttamente. Per esempio, è molto più facile avere l'età di una casa in anni piuttosto che secondi.

- Le features non devono fare uso di valori innaturali. Per esempio, sarebbe una cattiva idea se avessimo una feature il cui compito fosse quello di dirci da quanti giorni una casa è sul mercato, inserendo -1 se la casa non è mai stata sul mercato. Invece è una buona idea avere una feature che funga da indicatore, dove tramite un valore booleano indicare se i giorni della casa sul mercato sono stati definiti oppure no. In tal modo potremmo avere dei valori booleani true/false o 1/0 da mostrare e allo stesso tempo permettere alla feature *giorni_casa_sul_mercato* di mantenere i suoi valori naturali da 0 ad n.

- I valori delle nostre feature non dovrebbero cambiare nel tempo. Questo fa riferimento a quanto detto sull'avere i dati stazionari.

- L'ultima cosa è che una feature non dovrebbe avere dei valori anomali impazziti. Per esempio nel nostro dataset California housing, se creiamo una feature sintetica, come il numero di stanze per persone, dove prendiamo il numero totale delle stanze e lo dividiamo per la popolazione totale, abbiamo un certo numero di valori tra 0 e 3 o 4 stanze per persone per la maggior parte degli isolati. Ma su alcuni isolati abbiamo dei valori alti fino a 50. Questo

non è possibile, chi ha 50 stanze in un appartamento?. Quindi quello che facciamo è magari limitare il range o trasformare la nostra feature in modo tale che ci possiamo sbarazzare di questi valori anomali. Un altro trick potrebbe essere il **binning trick**.

6.3 Binning trick

Se pensiamo per esempio all'effetto della latitudine sui prezzi delle case in California, non c'è nessun tipo di relazione lineare tra il nord e sud che può essre mappata direttamente sui prezzi delle case. Ma all'interno di qualsiasi latitudine particolare c'è spesso una forte correlazione. E quindi, quello che possiamo fare è prendere la latitudine da nord e sud e suddividerla in contenitori. Questi contenitori diventano delle feature booleani, possiamo usare l'hot-encoding. Essenzialmente quello che otteniamo adesso è per esempio un 1 se ci troviamo nel contenitore specifico che mappa San Diego, o un 1 se ci troviamo in un contenitore che mappa South Bay Area, e 0 in tutti gli altri contenitori. Questo permette al nostro modello di avere un modo economico per mappare alcune di queste non -linearità senza compiere dei trucchi speciali.

6.4 Buone abitudini

Un elenco di buone abitudini potrebbe essere:

- **Conosci** i tuoi dati
- **Visualizza**: Fai il plot di istogrammi, scatter plot, diverse metriche per ranking.

- **Debug**: Ci sono degli esempi duplicati? Ci sono dei valori mancanti? Ci sono dei valori anomali? I dati del Training set e del Validation set sono simili?
- **Monitora**: Quantili delle features. Monitorare i dati col tempo. Solo perchè le fonti di dati si sono rivelate buone ieri non significa che lo possano essere in futuro.