Appunti ML (teoria)

Sommario

1. Introduzione

Il concetto di macchine che possono imparare nel 1950 con Alan Turing, sei anni dopo nel ’56 viene creato il campo dell’intelligenza artificiale.

Soli 10 anni dopo vengono formulati i primi algoritmi di apprendimento automatico basati su reti neurali.

Nel 1990 nasce il machine learning moderno basato su approcci indirizzati ai dati (data driven). Con l’avanzare della tecnologia, è stato possibile utilizzare modelli più complessi. In seguito nel 2010 nasce il **deep learning**

Con l’avvento e lo sdoganamento di Internet al pubblico, la quantità di dati è diventata rilevante al punto di causare la diffusione del machine learning su larga scala (difatti un modello addestrato su un dataset di ***sole*** 100 tuple potrebbe non essere adatto all’uso, mentre addestrato su un dataset con il numero di tuple nell’ordine dei milioni è sicuramente più affidabile e potrebbe essere adatto all’uso)

I modelli ML presentano vari modelli:

- Correttezza dei dati
- Logica di sviluppo
- Aging del modello (modello vecchio o meno)

Quindi necessariamente il modello va monitorato ed eventualmente vanno eseguite operazioni di correzione, aggiornamento, pulizia ecc

2. Cos’è il machine learning

Il machine learning è quindi il processo che porta alla generazione di intelligenza dai dati (valorizzazione e analisi dei dati).

Per queste analisi bisogna partire dalla scelta tra le varie categorie di approccio, differenti e categorizzabili tramite i seguenti criteri:

- **Livello di supervisione umana**
  - **Apprendimento supervisionato**
    - La macchina segue le direzioni e i modelli scelti dall’umano
  - **Apprendimento non supervisionato**
    - La macchina ha come unica interazione con l’umano quella di input di dati, il resto lo realizza lei cercando di categorizzare i dati e analizzarli secondo modalità scelte dalla macchina
  - **Apprendimento per rinforzo**
    - Una via di mezzo tra un apprendimento supervisionato e non supervisionato. Basato sul modello a feedback, ovvero in base alle analisi autonome della macchina, questa riceve bonus o punizioni imposte dall’utente
- **Capacità di apprendere incrementalmente**
  - **Batch**
    - Durante la fase di esercizio la macchina resta “ferma” sui dati su cui è stato addestrato, non si aggiorna ma ha un addestramento molto importante sui dati
  - **Online learning**
    - Ha un addestramento più breve ma ha il vantaggio di essere aggiornato e di adattarsi al cambiamento di tendenze e nuovi dati
- **In base a come effettuano le previsioni**
  - **Instance-based**
    - Sfruttano i dati su cui sono stati addestrati per classificare e analizzare i nuovi dati

- **Model-based**
  - Dopo la fase di addestramento verranno presi i dati di addestramento e trasformati in una funzione così da essere geometricamente comparabili

2.1. Livello di supervisione umana

2.1.1. Supervisionato

I dati di training sono etichettati con la soluzione corretta. Uno dei task tipici di apprendimento supervisionato è quello della classificazione, in cui la macchina deve assegnare dei label a degli oggetti; analogamente troviamo la regressione in cui la macchina deve restituire un valore numerico

2.1.2. Non supervisionato

I dati di training non contengono la soluzione, il sistema deve apprendere e rispondere da solo.

I tipici task sono:

- **Clustering**(trovate un sottoinsieme di dati simili tra loro associandole e raggruppandole in un cluster)
- **Riduzione della dimensionalità**(trovare il sottoinsieme di attributi necessari e caratterizzanti nei dati)
- **Estrazione di association rule**(ricerca delle *associazioni*, o delle tendenze nei dati cercandone varie correlazioni)

2.1.3. Semisupervisionato

Simile al supervisionato, nei dati di training sono presenti le risposte, ma solo in piccola parte (pochi dati con le risposte e molti senza)

2.1.4. Per rinforzo

Un agente osserva l'ambiente (dati) ed esegue delle azioni, ricevendo di conseguenza ricompense o penalità. L'agente impara ad ottimizzare le risposte per ottenere il massimo numero possibile di ricompense

2.2. Tipologia di apprendimento

2.2.1. Batch learning (offline learning)

Il sistema viene addestrato utilizzando tutti i dati a disposizione, il modello va in funzione senza essere più addestrato, restando fermo sui dati dell'addestramento. Non si aggiorna ma è più semplice da realizzare e richiede computazioni meno complesse pur richiedendo più spazio

2.2.2. Online learning

Il sistema apprende incrementalmente, ricevendo nuove istanze singolarmente o in gruppo (nuovi dati). Il modello, quindi, va in funzione aggiornandosi di continuo adattandosi ai nuovi dati. È molto più complesso da realizzare, più costoso e più oneroso in termini di computazioni, pur richiedendo meno spazio di quello batch. Molto influenzato da eventuali outliers

2.3. Tipologia di generalizzazione

1. Model-Based

Utilizza i dati di addestramento per ricavare una funzione per distinguere e predirre i nuovi dati, esempio predirre di che forma sarà una nuova figura in base alla sua posizione in un grafico cartesiano conoscendo che in una determinata area si ha una distribuzione maggiore di una certa figura

2. Instance-Based

Il sistema mantiene sempre i dati di training, li sfrutta per predire. Seguendo l'esempio di prima possiamo identificare una nuova figura in uno spazio cartesiano sfruttando la distanza di ogni elemento utilizzato per l'addestramento, se questa nuova figura sarà più vicina a determinate figure x il sistema predirà che sarà una figura x

Algoritmi Model Based e Instance Based:

- Come algoritmi **instance-based**, ricordiamo il ***k-Nearest Neighbors regression*** che stima un valore in base ai suoi k vicini
- Mentre uno dei più comuni **model-based** ricordiamo la **regressione lineare** (si cerca una curva che rappresenti i dati), con conseguente **analisi dei resti** per verificarne la bontà di raffigurazione.
    - Nel caso la bontà di rappresentazione non sia così alta è possibile utilizzare una regressione polinomiale

3. Quando usare il ML

Il machine learning è un approccio utilizzato per apprendere pattern complessi da dati esistenti, sfruttando questa conoscenza per eseguire predizioni sui nuovi dati.

3.1. Apprendere

Il machine learning quindi torna utile specialmente per apprendere tramite pattern **complessi** ma soprattutto **devono essere utili** poiché un sistema di machine learning è molto complesso per essere inutilmente utilizzato per pattern semplici, perciò è necessario anche dover fornire un dataset utile per l'individuazione di questi pattern

Il successo del machine learning è direttamente proporzionale alla qualità e alla quantità dei **dati disponibili**(**dataset**); questi rendono il modello proporzionalmente più preciso, fornendo la possibilità di **generalizzare** (fare predizioni sui nuovi dati) meglio, potendo anche scoprire pattern nascosti o non previsti

3.1.1. Zero shot learning

È un sistema che riesce a fare buone predizioni senza essere stato addestrato su dati relativi a quel compito, ma comunque addestrato su dati legati ad un task correlato (ovvero se un modello è in grado di eseguire predizioni su un altro task o un altro problema legato a quello interessato, questo potrebbe anche produrre soluzioni buone per il task richiesto)

3.1.2. Continual learning

Analogamente, il modello non è addestrato per bene ma imparerà dai dati in arrivo, questo ovviamente comporta rischi

Esistono poi sistemi che non presentano un addestramento precedente e senza continual learning che seguono un approccio **fake it till you make it**

3.2. Predizioni

I modelli di ML producono predizioni. Rendendoli quindi utili solo per problemi che si prestano a predizioni, ovvero problemi che giovano di stime anche se approssimative

3.3. Dati non visti

I modelli di ML producono predizioni. E questi su cui vengono fatte predizioni hanno dei pattern comuni ai dati di training; perciò, i dati di training e quelli che saranno oggetti di predizioni devono essere di provenienze e distribuzioni simili. Questo si evidenzia eseguendo assunzioni, se queste sono *forti* e risultano corrette fanno da base portante per il modello per fare previsioni. Nel caso queste assunzioni risultino sbagliate bisogna riaddestrare il sistema

Altre caratteristiche

- **Ripetitività**
    - Nel caso il task sia ripetitivo è decisamente più facile trovarne pattern utili
- **Basso costo per predizioni errate**

    - Se la previsione del task, eventualmente fosse sbagliata, non ha un enorme impatto negativo allora è utile poter osare
- **Larga scala**
    - Il machine learning, per la sua complessità, risulta utile solo su una grande quantità di dati o di operazioni.

4. Quando NON usare il ML

- Non è etico
- Esistono soluzioni più semplici
- Non conviene economicamente
- Non conviene utilizzare il ML per via della limitazione delle tecnologie attuali (calcoli troppo complessi)

5. Applicazioni del ML

Consumer

Il ML è molto utilizzato nei **motori di ricerca**, cercando di restituire risultati pertinenti in base alla quantità delle interazioni e al feedback degli utenti.
Altri usi sono quelli di **profilazione** per suggerire contenuti rilevanti basati sulle preferenze individuali dell'utente (es. Netflix)
Un'altra applicazione è il **predictive writing**, ovvero suggerire e prevedere parole prima che vengano scritte, basandosi sullo storico delle parole utilizzate.
Analogamente, i **sistemi di traduzione automatic**a utilizzano sistemi di linguaggio basati sull'analisi della grammatica
Sistemi di **smart home** basandosi sulle abitudini dell'utente, adattandosi di conseguenza
Sistemi di **riconoscimento facciale**, basandosi su immagini o video. Apprendendo le caratteristiche del volto di una persona, estraendone i particolari, riuscendo a riconoscerlo

Business

Per le aziende, migliorare anche solo lo 0,5% la precisione del riconoscimento facciale (per esempio) può significare risparmi di migliaia o milioni di dollari, dando molta più rilevanza alla precisione delle previsioni del ML. Alcuni esempi possono essere:
Rilevamento di frodi,
Price optimization,
Profilazione dei clienti,
Analisi dei nuovi potenziali clienti,
Churn prediction, ovvero, il task di identificare o predire quanto un cliente sta per abbandonare i prodotti dell'azienda
Brand monitoring, ovvero, l'analisi di quanto sia popolare o interessante il prodotto

6. ML nella ricerca e nella produzione

I due ambiti possiedono varie **differenze**, le principali sono:

- **Requisiti**
    - La ricerca preferisce le performance
        - Spesso si creano sistemi anche troppo complessi da poter essere concretamente utilizzati

    - La produzione possiede requisiti dati dai vari stakeholder
        - Puntano a massimizzare la performance minimizzando la complessità, preferendo la seconda ad un sistema troppo complesso e non utilizzabile
- **Priorità**
    - La ricerca necessita di un training rapido e alto troughput (per massimizzare i frutti dell'analisi)
    - La produzione invece inferenza rapita e bassa latenza (per massimizzare l'efficienza)
- **Dati**
    - La ricerca possiede dati statici
    - La produzione costantemente in evoluzione
- **Fairness(imparzialità) e interpretabilità**
    - La ricerca non se ne interessa
    - La produzione invece dovrebbe considerarla per il proprio sistema di ML
        - In ambito di produzione l'interpretabilità dei dati prodotti dal sistema di ML è necessaria per evitare di essere ricaduti in bias o di aver prodotto risultati errati
- **Tempo impiegato**
    - La ricerca impiega più tempo nella creazione di modelli e algoritmi
    - La produzione invece impiega più tempo nella qualità e nella pulizia dei dataset

Entrambi peccano però di un'eccessiva attenzione alla fase di sviluppo non concentrandosi sulla face di deployment e manutenzione. Queste fasi sono importanti in quando il principale rallentamento di un modello ML è il suo training durante lo sviluppo, in seguito a questo, il collo di bottiglia più grande dopo il rilascio è l'inferenza (predizione)

Una delle questioni rilevanti del ML è proprio l'imparzialità, poiché il sistema di machine learning non può predire con certezza ma analizzare il passato e i dati così da cercare di predire un evento futuro, sono possibili errori per via di un modello GiGo (Garbage In, Garbage Out), ovvero, se i dati di addestramento sono rozzi, influenzati da bias ecc… il sistema di ML allora produrrà risultati falsati o frutti di bias

Es. *https://www.cbsnews.com/news/mortgage-discrimination-black-and-latino-paying-millions-more-in-interest-study-shows/*

7. Che obiettivo ha il ML?

Lo sviluppo di un sistema di ML dev'essere motivato da obiettivi precisi, per esempio, per un'azienda quello di aumentare i profitti (che sia direttamente attraverso aumenti di vendite, o indirettamente aumentando la soddisfazione del cliente).

Molte aziende, quindi, hanno creato le proprie metriche per valutare il sistema ML usato, piuttosto che seguire una metrica standard. Per esempio, Netflix misura il proprio sistema di ML tramite una metrica chiamata "Take-rate"

8. Quali requisiti richiede il ML?

Generalmente sono questi:

- Affidabilità
- Scalabilità
- Manutenibilità
- Adattabilità

Ovviamente i requisiti variano da contesto a contesto

8.1. Affidabilità

Si definisce affidabile un sistema che continua svolgere correttamente la funzione (in quanto a prestazioni desiderate) anche in presenza di avversità. Queste avversità possono accadere *silenziosamente*, ovvero, senza avvisi espliciti. È quindi necessario un controllo per rilevarli

8.2. Scalabilità

Si definisce scalabile un sistema di ML che può adattarsi all'aumento di:

- Complessità
    - Rimpiazzare il modello con uno più complesso adattandosi all'evoluzione dei dati
- Volume del traffico
    - All'aumentare del traffico il sistema deve riuscire ad elaborare con prestazioni accettabili
- Numero di modelli
    - Aggiungendo funzionalità al sistema potrebbe essere necessario ed utile aggiungere nuovi modelli

Indipendentemente da questo qualsiasi tipo di crescita va gestita per bene, per esempio:

> in una farm di GPU potrebbero esserci momenti di richiesta di 100 GPU e momenti lunghi di downtime in cui la richiesta è solo di 10 GPU, tenere accese altre 90 GPU è un costo notevole da dover risolvere.

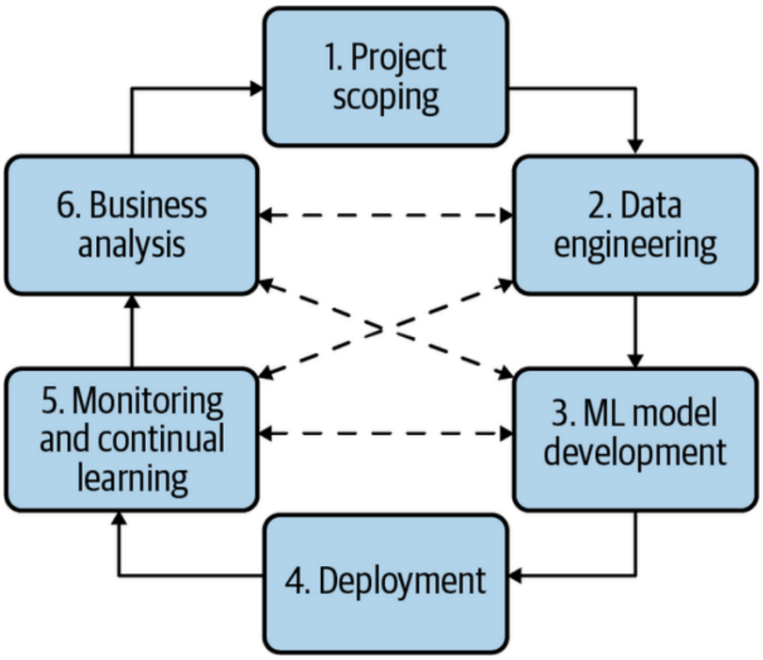
Analogamente, all'aumentare dei modelli, il controllo manuale di monitoraggio non basterà più come nel caso di un modello singolo. Potrebbe essere necessario avere procedure automatizzate di riaddestramento per modelli non più efficienti

8.3. Manutenibilità e Adattabilità

Coinvolgendo figure con competenze diverse e varie nello sviluppo del sistema ML, è necessario che ognuno possa contribuire al sistema.

9. Come si sviluppa un sistema ML?

Il sistema ML, per definizione, è iterativo e molto spesso ***senza fine***, richiedendo costante attenzione, monitoraggio con conseguenti aggiornamenti. Generalmente il ciclo di vita di un sistema ML è:



1. Project scoping = definizione degli obiettivi, dei vincoli e stimando le risorse necessarie coinvolgendo gli stakeholder
2. Data Engineering = Elaborazione dei dati con conseguente gestione delle fonti creando l'insieme dei dati di addestramento
3. Model Development = Estrazione delle feature significative dai dati di addestramento e sviluppo modello
4. Deployment = messa in funzione del sistema
5. Monitoring = Monitoraggio delle performance del modello (secondo la metrica dell'azienda) e fasi di aggiornamento
6. Business analysis = valutazione delle performance rispetto agli obiettivi aziendali

Come descritto dal grafico, le fasi sono influenzate tra loro; ovvero, al termine della business analysis potrebbe essere necessario ritornare al data engineering

9.1. Definizione del problema (project scoping)

Prima di tutto in un sistema di ML è necessario definire il problema, che questo sia un problema:

- **Regressione**
- **Classificazione**
  - **Binaria**
    - La classificazione deve avvenire tra due tipi di risposte
  - **Multiclasse**
    - La risposta è sempre una sola ed unica per oggetto (una sola label per campione)
    - Più di due tipi di risposte si dividono in:
      - **Alta cardinalità**
        - Ovvero quando il numero di classi è **molto elevato** (anche più di mille).
        - Uno dei **problemi** è la **complessità**, perché, per ogni possibile classe nel dataset devono esserci tanti esempi per ogni tipo di classe di risposta (es. se servono 100 esempi per classe e abbiamo 1000 classi, serviranno 10000 esempi)
      - **Bassa cardinalità**
        - Quando il numero di classi è superiore di due ma non troppo superiori, massimo ordine della decina
  - **Multilabel**
    - Task più complesso
    - A differenza delle altre che hanno un solo risultato (ovvero sia nella multiclasse che nella binaria il risultato sarà uno solo), in questo caso il risultato sarà l'unione di più risposte (la risposta è composta da più label per campione)

La definizione del problema può passare per un processo di "semplicifazione" del problema, nel caso di multiclasse e multilabel è possibile suddividere il task proposto in sottotask più semplici e piccoli

9.1.1. Classificazione per task multiclasse ad alta cardinalità

Gerarchica:

- È una tecnica che distingue le istanze in diversi **macrogruppi**. Successivamente ulteriori **classificatori** vengono utilizzati per classificare i sottogruppi

Addestrare un modello binario per ogni label esistente:

- Uno dei **problemi** di questa classificazione è proporre risultati divergenti, creando difficoltà
  - Una delle soluzioni possibili è proporre il risultato "*più probabile*"

9.1.2. Classificazione per task multilabel

Per i task **multilabel** è possibile:

- **Considerare un output come un array**
  - Dove le celle dell'array sono i vari label di risposta.
    - (*es. con 4 label possibili, la risposta sarà composta da un array [0,0,0,0]*)
- **Addestrare un modello binario per ogni label esistente:**
  - In questo caso però è complesso distinguere in quanto chi monitora i risultati può avere **opinioni contrastanti**
    - (*es. con **4 label** sarebbe necessario addestrare **4 modelli binari** che produrranno ognuno 4 risposte corrispondenti ai 4 label possibili*)
  - Uno dei **problemi** di questa classificazione è proporre risultati divergenti, creando difficoltà

- Modernamente si utilizzano un numero dispari di modelli, sfruttando un sistema a votazione (a maggioranza)

Generalmente è più utile ed efficiente riportare il task in un problema di regressione.

- Questo può essere semplificato scomponendo il task ed eseguendo una regressione per ogni componente del task
  - Es. predire quale app ha più probabilità di essere aperta

10. Funzione obiettivo

Si definisce funzione obiettivo (o funzione di loss), una funzione che definisce il grado di accuratezza che il modello sta assumendo nella fase di training (indicano le inferenze errate).

L'obiettivo del ML è minimizzare le funzioni di loss, bisogna quindi monitorare il modello durante il training e le funzioni di loss associate.

Tra le più comuni: **cross-entropy**, **Root Mean Square Error** (**RMSE**), **Mean Absolute Error** (**MAE**)

Cross entropy:

Il calcolo di questa funzione di loss è dato da:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

dove, y indica il valore effettivo e y^ il valore predetto

A livello pratico, le funzioni di loss sono necessarie per comprendere la bontà del modello di ML.

Queste possono essere concatenate, per esempio in un social, possiamo combinare una funzione di loss che calcola quanto un post sia di qualità e un'altra che indica quanto sia coinvolgente. Con i risultati delle due precedenti funzioni di loss possiamo crearne una nuova, data da un parametro alfa per la prima funzione e un parametro beta per la seconda. Questo rende possibile "aumentare" o "diminuire" una funzione di loss. In tal caso però è necessario riaddestrare il modello

Considerando l'esempio precedente, è necessario creare un modello che valuta la qualità e minimizza la funzione di loss associata, e un altro modello che valuta il coinvolgimento e minimizza il loss associato

11. Trade-off sui dati

A livello business, la creazione di un modello ML è necessario un trade-off tra la quantità e qualità dei dati oppure progettare modelli nuovi ed efficienti. E generalmente entrambi è difficile da massimizzare, bisogna quindi creare un modello bilanciato e apposito per i task assegnati.

Molti sottolineano che è necessario dare estrema priorità a modello e algoritmi intelligenti a scapito della quantità e qualità di dati

Altrettanti però, supportano la priorità alla quantità e qualità di dati, che a detta loro possono essere utilizzati per generalizzare correttamente il problema del task, magari riuscendo a cogliere pattern più complessi di quelli che un modello con pochi dati potrebbe mai riconoscere.

Attualmente non esiste una risposta univoca, anche se è facile affermare che i dati siano essenziali per il funzionamento di un sistema ML. I dati raccolti e organizzati devono comunque passare per un percorso di pulizia e rilevazione di errori o anomalie.

12. Data Engineering

Data l'enorme mole di dati prodotta negli anni (soprattutto dall'impiego comune di Internet), i sistemi di machine learning sono stati agevolati per quantità di dati; il problema nasce però proprio nel gestirli, nascono quindi algoritmi complessi.

Ad oggi l'ordine di grandezza dei byte è cresciuto nell'ordine degli yottabyte. Si definiscono così i ***Big Data***, il termine nasce da **Micheal Cox** e **David Ellsworth**. L'enorme impiego di Internet (dall'IOT ai trasporti, passando per i social network e settori sanitari/scientifici/militari)

I Big Data, per motivi di complessità, necessitano l'uso di DBMS NoSQL. Poiché una delle caratteristiche dei Big Data è la variabilità, questo implica perciò che il DB che gestisce i suddetti dati dev'essere in grado di "evolvere" la propria struttura per adattarsi a cambiamenti nei dati

Le fasi del ciclo di vita dei Big Data sono:

- **Capture**
  - Ovvero, come ottenere i dati
- **Extraction**
  - Estrarre i dati da quelli raccolti nella fase di cattura, questo produce "dirty data"
- **Curation**
  - Pulizia dei dati, avviene tramite algoritmi di pulizia (o di cura – ***Data Curation***).
  - Dopo questa fase i dati diventano effettivamente utilizzabili poiché ripuliti da dati di bassa qualità
- **Storage**
- **Search**
- **Sharing**
- **Querying**
- **Analysis**
  - L'analisi con l'avvento dei Big Data non è più "manuale" come su pochi dati, questa avviene calcolando statisticamente il trend, l'andamento, il plot ecc
- **Visualization**
  - Questa fase è utile soprattutto come supporto per il management dei big data

Generazione dei dati

Si definisce *sorgente* un'entità che produce dati. Nel caso dei dati generati da utenti, possiamo assumere che questi saranno:

- Molto eterogenei
- Con probabilità di errore semantici o sintattici (es. testo al posto di numeri)

La qualità, quindi, è da verificare e garantire nel caso dei dati generati da utenti.

Nel caso invece, di dati prodotti da sistemi (generalmente detti log), possiamo essere quasi certi che questi saranno ben formattati. Seppur questi hanno un'altissima probabilità di essere ben formattati e ben riempiti bisogna garantire un'elaborazione rapida (devono poter essere interpretati) e sicuramente è necessario trattarli per evitare di rilasciare dati sensibili degli utenti che usano quel sistema che produce log.

I database aziendali sono quelli più utili perché già organizzati e puliti, pronti per essere usati per un sistema di ML. Necessitando comunque un'analisi per risolvere eventuali incomprensioni date dalle convenzioni usate che potrebbero essere diverse.

I dati possono essere categorizzati in:

- Di prime parti
    - Ovvero l'azienda recupera dati direttamente nel suo sistema per i suoi clienti e utenti
- Di seconde parti
    - Dati collezionati da altre aziende sui propri clienti che vengono resi disponibili da queste
- Di terze parti
    - Dati collezionati da aziende preposte alla raccolta dati, su utenti che non sono loro clienti

13. Dati di training

Come già detto in precedenza, i dati per un sistema di ML sono il fondamento del sistema stesso. Questi devono essere di qualità ed essere puliti. Bisogna quindi costruire dataset adatti al sistema che si vuole costruire.

Per esempio, questi potrebbero essere complessi, disordinati, protetti o (quasi sempre) affetti da bias (pregiudizi). Bisogna quindi, effettuare una fase di pre-processing di pulizia di questi.

Una delle principali fasi è quella del **campionamento**, si divide in:

- Creazione dati di addestramento
- Divisione di dati di training, validation e di test
- Definizione di tutti gli eventi possibili che possono accadere durante la messa in opera di un sistema di ML

Il campionamento può avvenire secondo due "tecniche"

- Campionamento ***non probabilistico***
    - *Basato su un campionamento deterministico*
- Campionamento ***casuale***
    - *Basato su un campionamento aleatorio e probabilistico*

Campionamento non probabilistico

La tecnica si basa su definizioni deterministiche, e i prinicipali metodi sono campionamento:

- Di convenienza
- A valanga (o palla di neve)
- Per giudizio
- Per quote

Ogni metodo adottato non riflette pienamente il mondo reale, quindi ognuno possiede un bias che porta il sistema a produrre assunzioni errate che non rappresentano la realtà (**bias di campionamento**)

Campionamento di convenienza

Prevede che i campioni vengano selezionati in base alla loro semplicità di acquisizione (disponibilità), si utilizza questa tecnica quando la quantità di dati a disposizione è enorme e testare tutti i membri di questa popolazione risulterebbe complicato o impossibile (per tempo o risorse)

PRO:

- Basso costo
- Semplice
- Rapido

CONTRO:

- Mancanza di varietà
- Validità limitata
- Errori difficili da individuare
- Bias di campionamento

Campionamento a palla di neve (a valanga)

Come una valanga, si parte da un piccolo campione e poi si ingrandisce il set di dati.

È una tecnica di tipo incrementale ed è utilizzata quando non si ha un accesso immedato a tutta la popolazione dei dati.

Si definiscono quindi una piccola popolazione composta da pochi campioni noti, quindi si scelgono dei campioni facilmente accessibili (come il campionamento di convenienza) e poi man mano si aumentano i dati del set

PRO:

- Scalabile
- Disponibilità di accesso a dati che non erano noti o che erano difficilmente raggiungibili
- Riduzione di tempi e costi
    - Saranno i dati del piccolo campione ad aiutare o fornire nuovi dati

CONTRO:

- Bias rete sociale (di selezione)
    - I dati potrebbero non rappresentare correttamente tutta la popolazione dei dati, si basano sui campioni raccolti (che potrebbero essere affetti da un bias)
- Forte dipendenza dai dati del campione scelto nelle fasi iniziali
- Errori di campionamento difficilmente valutabili

Campionamento per giudizio

Si differenzia da quelle precedenti per la selezione dei dati, questi non vengono più scelti “a caso” bensì vengono selezionati dati affini a quelli cercati (quelli che rispettano determinate caratteristiche). Si utilizzano in sistemi in cui è necessario ottenere dati di ottima qualità rispetto alla precisione statistica nella previsione per tutta la popolazione

PRO:

- Ottima qualità di dati
- Precisione molto alta nel campo specifico dei dati (utilizzo in contesti qualitativi)

CONTRO:

- Bias di selezione
- Bassa precisione statistica per la previsione di tutta la popolazione (campioni non rappresentativi)
- Non adattabile ad altri contesti
    - Sarebbe necessario riaddestrare il sistema con dati diversi

Campionamento per quote

Si basa sull’individuazione di *quote* per determinate porzioni di dati, basando la tecnica di selezione su queste quote. Risulta utile nel campionamento di dati molto limitati (es. per malattie rare)

PRO:

- Utile per campionare dati molto limitati nel caso non ci siano altri metodi disponibili

CONTRO:

- Presenza di bias
- Bassa rappresentatività statistica

Campionamento casuale

Si utilizza come “evoluzione” del campionamento non probabilistico, cercando di rimuovere o ridurre la presenza di bias. Queste tecniche si basano su una raccolta di campioni basandosi su distribuzioni di probabilità, queste si dividono in:

- Casuale semplice
- Stratificato
- Pesato
- A Serbatoio
- Per Importanza

Campionamento casuale semplice

Tutti i campioni hanno uguale probabilità di essere scelti (equiprobabili), questa estrema semplicità nella selezione può portare però ad alcuni problemi, come, nel caso ci sia una piccola parte della popolazione è molto probabile che questa venga trascurata. Dando equa probabilità ad ogni elemento della popolazione verranno selezionati principalmente quelli più ricorrenti o che compongono la maggior parte della popolazione

PRO:

- Semplice
    - Implementazione
    - Criteri semplici

CONTRO:

- Svantaggia classi rari nella popolazione
- Problematico con carenza di dati
- Non considera correlazioni di dati
- Può risultare costoso in popolazioni molto grandi

Campionamento stratificato

È un’evoluzione del campionamento semplice, aggiungendo una suddivisione preventiva della popolazione in gruppi (organizzati per elementi affini tra loro secondo una caratteristica – classi) , e poi, il campionamento avviene indipendentemente in ogni gruppo.

È utile per analizzare popolazioni eterogenee o che presentano classi rare. Per esempio in una popolazione di persone, possiamo dividerla in 3 gruppi (strati) ognuno composto da persone di range di età differenti, e poi da ognuno di questi strati si possono selezionare 50 persone a caso.

PRO:

- Aumenta la rappresentatività
- Precisione nel campionamento
- Efficiente
- Permette confronti diretti tra strati

CONTRO:

- Difficile scegliere il criterio di raggruppamento
    - In alcuni casi è addirittura impossibile
- Complesso e costoso

Campionamento pesato

Si basa sull'assegnazione di un peso ad ogni elemento (o classe di elementi), questi rappresentano la probabilità che quel campione venga scelto

È utile nel rappresentare popolazioni che presentano classi di dimensioni molto diverse tra loro, ottimizzando l'uso di risorse a disposizione.

Permette di risolvere il **bias di non risposta** (ovvero alcune entità che non vengono campionate perché rare o poco probabili)

PRO:

- Risolve il bias di non risposta
- Efficiente con le risorse fornite
- Flessibilità

CONTRO:

- Complesso
- Difficile analisi dei risultati
- Possibile presenza di bias
- Gestione difficile dei pesi

14. Database NoSQL

La principale differenza tra un DB SQL e quello NoSQL che spinge alla scelta del secondo è la possibilità di:

- Trattare grossi volumi di dati
- Tramite transazioni BASE, garantire le ***ACID***

Questi DB si basano su:

1. Garanzia di disponibilità dei dati
    a. Anche in presenza di fallimenti multipli (sistemi distribuiti)
2. Abbandono del requisito della consistenza deei modelli ACID
3. Consistente

Un sistema distribuito però può garantire solo due tra le caratteristiche elencate, allora, da caso a caso si decide di utilizzare DB diversi in base alle esigenze, (es. per un sistema che sia consistente e disponibile si preferisce MySQL sfavorendo la tolleranza ai fallimenti)

Alcune delle difficoltà affrontate dai DB NoSQL sono:

- Maturità
    - Intesa come "anzianità" e supportabilità, difatti, essendo nati molto in seguito ai DB SQL per molti non sono ancora ritenuti "maturi"
- Supportabilità
    - Legato alla questione di prima, i sistemi NoSQL spesso sono open source, mentre quelli SQL no
- Amministrazione
    - NoSQL non necessità un grande supporto amministrativo
- Expertise
    - È più difficile trovare un esperto di sistemi NoSQL piuttosto che uno di sistemi SQL (per la questione di maturità)

L'utilità di NoSQL nasce in determinati casi:

- Struttura non definibile a priori
- I dati nel sistema sono molto affini, causando una scarsa efficienza nelle operazioni di JOIN (che è un'operazione dispendiosa)
- Necessità di prestazioni più elevate

Poiché NoSQL non è sempre con un'unica raffigurazione o struttura, è possibile dividere le varie categorie in:

- Document data store
- Key-value data store
- Graph based data store
- Column oriented data store

Document Data-Store (mongoDB)

- Memorizza dati **non strutturati**
- È **schema-less**
- Un documento è identificato da una chiave primaria
- Garantisce la scalabilità orizzontale (quantità di dati)

La rappresentazione è quindi basata su oggetti dette **documenti**, ognuno ha un certo numero di proprietà **ma** non devono necessariamente seguire una struttura fissa

L'identificazione è come quella di un record in cui i valori per il record è il documento.

Ogni documento può essere messo in relazione con altri tramite dei riferimenti, concedendo la possibilità di creare relazioni tra documenti

Key-value Data-Store (redis)

Si utilizza un "associative array" (chiave-valore) per lo storage. Tramite le chiavi è possibile effettuare operazioni di storage, update e ricerca.

Sono costruiti come **mappe** o **dizionari** in cui ogni chiave identifica il valore a lei associata.

Graph-based Data-Store (neo4j)

Utilizza nodi, proprietà ed archi per rappresentare entità, attributi e relazioni

Rende semplice ed intuitivo il modello logico.

Ogni elemento possiede un puntatore all'elemento suo adiacente, permettendo la navigazione tra gli archi (per ricercare le varie relazioni o cercare dati)

È utilizzato spesso nella rappresentazione di reti sociali. I dati sono quindi memorizzati sia nei nodi che negli archi, questi ultimi nel caso sia necessario memorizzare informazioni che riguardano una specifica relazione.

L'utilità di questa rappresentazione nasce nella logica che porta con se l'utilizzo dei grafi, tramite algoritmi di navigazione

Column-oriented Data-Store (Cassandra)

I dati sono memorizzati in termini di colonne, massimizzando la distribuzione dei dati. Un insieme di queste colonne è detta ***famiglia***. Vi è quindi una analogia con il modello relazionale, solo che la rappresentazione concedono una semplice distribuzione, una scalabilità ed un'alta performance.

Concedono quindi una rapidità di aggregazione in base ad attributi, questo rende i DB column-oriented apprezzati da aziende che trattano milioni e miliardi di dati

Trade-off sulla scelta di NoSQL

Più è complessa la struttura e più sarà scarsa la capacità di memorizzazione; perciò, è necessario determinare il sistema più adatto all'applicazione. In ogni caso un qualsiasi sistema NoSQL è sempre meglio di un qualsiasi sistema SQL, solo che eccellono in uno solo dei campi di **capacità di memorizzazione** e **complessità**

15. Dataflow

Un flusso di dati è il canale di trasferimento da un sistema all'altro, le principali modalità di dataflow sono:

- Attraverso i Database
- Attraverso servizi (tipo richieste POST/GET)
- Attraverso un trasferimento in tempo reale

Attraverso il Database

È il passaggio più semplice, in cui un processo A che deve comunicare con un processo B, scrive in un database che verrà poi letto dal processo B.

Non funziona sempre perché:

- Tutti i processi che richiedono i dati devono essere in grado di accedervi
- Devono poter leggere/scrivere sul database (che sono operazioni costose)

Attraverso servizi

È un tipo di dataflow che sfrutta servizi di trasferimento dei dati. Questo avviene attraverso una rete, quindi, un processo A manda una richiesta una precisa richiesta dei dati di cui necessita al processo B sulla rete, questo manderà ad A una risposta contenente i dati richiesti

È necessariamente legata alla gestione delle richieste

Trasferimento in tempo reale

È un tipo di dataflow che si basa sulla replicazione dei dati prodotti da un processo con gli altri processi, questo è molto efficiente IN SCENARI SEMPLICI, quando troppi processi devono essere interconnessi tra loro è necessario creare altrettanti flussi di dati per ognuno.

Il trasferimento in tempo reale è basato su **eventi**. Si definisce evento un dato trasmesso a un trasporto in tempo reale, questi eventi sono visibili ai processi. Esistono due caratteristiche di trasmissione in tempo reale:

- **Publish-Subscribe**
  - Ogni servizio può *pubblicare* un evento e qualsiasi servizio può essere un *consumer*, ovvero, analizza e legge gli eventi pubblicati dagli altri.
  - Questa struttura si basa su una politica di conservazione di dati per una determinata finestra temporale
- **Message Queue**:
  - Un evento, alla sua produzione, ha un insieme di consumatori previsti e la coda dei messaggi è responsabile di indirizzare il messaggio pubblicato a questi consumer

Batch Processing vs Stream Processing

Si definiscono dati storici i dati che arrivano a motori di archiviazione database (data lake, data warehouse) diventando ***storici***

Si contrappongono ai dati in **streaming**, ovvero, i dati un continuo arrivo senza una prospettiva di fine di questa.

Elaborazione Batch

Per i dati storici, sono predisposti dei **batch job**, ovvero task di analisi sui dati. Questa avviene periodicamente su dati già memorizzati, quest'analisi può produrre risultati interessanti per l'ottimizzazione del sistema.

L'elaborazione batch è stata oggetto di ricerca per molti anni, creando sistemi come MapReduce e Spark

Elaborazione in Streaming

Vengono avviati sia come processo periodico **sia** come **richiesta**

Infatti, l'analisi e l'elaborazione deve accadere in tempo reale, permettendo un'esecuzione sui dati appena prodotti dall'analisi. Il periodo di invocazione dell'analisi dev'essere molto breve.

La sostanziale differenza tra Streaming e Batch è la latenza di elaborazione:

- In Batch è necessario ricalcolare con tutti i dati presenti
    - Alta latenza, inadatto alla produzione efficace di dati
- In Streaming, si esegue un calcolo basato sugli ultimi dati e un precalcolo fornito
    - Bassa latenza

La differenza tra le due elaborazioni è anche la ***maturità*** ovvero, quello batch p supportato da tanta ricerca, mentre quello streaming è "recente"

Anche in questo caso è necessario un trade-off, quella batch può produrre risultati molto precisi nel caso i dati da analizzare siano molto affini. Mentre quella streaming consente un'adattabilità alla volubilità dei dati.

Questa è la differenza tra batch learning e online learning

16. Etichettatura

In seguito al campionamento (e alla creazione dei dati di training) il passo successivo è (nel caso di approcci a sistemi supervisionati) l'etichettatura. Questo passaggio è essenziale per preparare finalmente i dati di addestramento. Essa consiste nell'assegnazione di un'etichetta (o più etichette) ad ogni entry nel dataset.

L'etichettatura può avvenire *naturalmente* (ovvero etichettati automaticamente) e etichettatura manuale.

16.1. Etichettatura manuale

È la tecnica qualitativamente superiore, in cui un supervisore manualmente etichetta le entry.

Pro:

- Qualità superiore

Contro:

- Poco efficienti
    - Economicamente e temporalmente
        - Può essere risolto quello temporale a scapito di quello economico (assumendo più etichettatori)
            - Assumere più annotatori però può portare ad altri problemi di ambiguità sulle etichette, questo è detto ***molteplicità delle etichette*** (***label ambiguity***)
        - Il tempo che richiede l'etichettatura penalizza di molto la capacità di adattamento del modello

- Problemi di privacy
    - Non tutti i dati del dataset sono rilasciabili o leggibili da chiunque
- Problemi di bias
    - Es. **label ambiguity**
        - Questo può essere ridotto fornendo delle regole comuni da seguire, così da dare delle direttive precise

16.2. Data Lineage

Considerando i problemi dell'etichettatura manuale, nello specifico l'ambiguità delle etichette date dai bias degli etichettatori (o in generale dalle ***diverse sorgenti***).

In questa tecnica è importante tenere traccia di queste sorgenti e dei dati che forniscono

Pro:

- Permette di individuare potenziali bias
- Permette di fare debug del modello
    - Conoscendo le sorgenti dei dati che potrebbero essere problematiche è più facile risolvere questo problema (**Data Inspection**)

16.3. Etichettatura Naturale

Esistono dati che non hanno necessità di essere etichettate a mano. Ovvero dati di previsione del modello che vengono confrontate con dati che non provengono da un'etichettatura naturale, es. Maps confronta il tempo e il percorso minimo tra due punti previsto con quello effettivamente rilevato dalla posizione dell'utilizzatore.

Altri esempi di etichettatura naturale sono:

- Un e-commerce potrebbe considerare i prodotti più cliccati come positivi e quindi favoriti dal sistema di raccomandazione, mentre potrebbe considerare in modo negativo (e da non consigliare) il *non cliccare* dell'utente su un prodotto

Si definiscono poi:

- Etichette implicite
    - Non necessita di un'interazione effettiva, per esempio l'e-commerce che considera negativamente un prodotto non molto cliccato
- Etichette esplicite
    - Interazione con l'utente, per esempio l'e-commerce che raccomanda i prodotti più cliccati

Queste etichette però non possono essere assolute e pregiudicative per tutto il sistema per sempre, va definito quindi un lasso di tempo massimo, detto "**Lunghezza del ciclo di feedback**", che definisce quindi "*quanto deve durare l'etichetta*". È necessario quindi un trade-off tra **lunghezza del ciclo di feedback** e **accuratezza**.

16.4. Etichettatura Programmatica

L'etichettatura di alta qualità è tutt'oggi un problema importante, esistono varie tecniche per affrontare il problema, queste sono:

- Weak Supervision
    - Sfrutta euristiche e regole precise per generare etichette
    - Non necessita di set di partenza, però, potrebbe essere molto utile averne anche uno piccolo per poter guidare lo sviluppo delle euristiche
- Semi-supervision
    - Sfrutta assunzioni strutturali per generare etichette
    - È necessario un set di partenza per generare le etichette
- Transfer Learning
    - Come lo zero-shot learning, sfrutta modelli preaddestrati su altri task che potrebbero combaciare e funzionare anche in altri contesti
    - Nel caso di un'affinatura del modello preaddestrato si necessita di un set iniziale
- Active Learning
    - L'etichettatura avviene sui campioni del dataset più importanti e utili al sistema

16.4.1. Weak Supervision

È un approccio popolare nell'etichettatura perché la velocizza tantissimo
Si basa soprattutto sulla definizione delle euristiche, queste devono assolutamente essere precise e fatte bene.
Esistono varie librerie per la weak supervision, una di queste è Snorkel che ha il fine di *codificare* l'euristica (tipo blocchi if-then-else)
Si possono definire vari tipi di Euristiche:

- **Keyword**
    - Sfrutta l'individuazione di parole chiave all'interno del campione
- **Espressioni Regolari**
    - Cercano un match con l'espressione regolare fornita
- **Database Lookup**
    - Si utilizza una lista di concetti, e si cerca un match con almeno una di queste nel campione
- **Output di altri modelli**

Le euristiche (e quindi le etichette prodotte) non sono immuni dagli errori, questi sono definiti come **rumore**. Questo può essere ridotto combinando diverse euristiche o ripesando diverse keyword
La weak supervision risulta migliore dell'etichettatura manuale sotto ogni punto di vista:

- Meno costosa
- Maggiore privacy
- Più veloce
-

16.4.2. Semi-Supervision

Esistono tre diverse tecniche di semi-supervision:

- Self training
- Similarity based
- Perturbation based

Self-training

È la più classica tecnica di semi-supervision, ovvero sul set iniziale di dati etichettati. Si utilizza il modello addestrato su questo piccolo dataset per fare predizioni (ipoteticamente corrette), a questo set possiamo poi aggiungere in seguito nuove etichette in modo che il modello potrà eseguire nuove predizioni che potrà poi sfruttare per poter riaddestrare il modello

Similarity-based

La tecnica si basa sull'assunzione che dati simili condividano la stessa etichetta. Per esempio nell'assegnazione di un topic ad una serie di hastag, possiamo assumere che:

- Tutti gli hastag nello stesso tweet siano dello stesso topic (similarità semplice)
- Si suddivisono gli hastag in gruppi di similarità più piccoli (similarità complessa)

Perturbation-based

È una tecnica che si basa sull'aggiunta volontaria del rumore nei dati.

Il fine di questo passaggio è quello di verificare la robustezza delle etichette e del modello, per esempio in un sistema di riconoscimento di immagini possiamo aggiungere della noise (rumore) oppure in un sistema che lavora su numeri interi possiamo aggiungere dei numeri casuali. Con queste lievi perturbazioni le etichette non dovrebbero cambiare

Underfitting e Overfitting

Uno dei problemi dei sistemi semi-supervision è quello **dell'overfitting**:

Si definisce overfitting l'eccesso di adeguamento ai dati,
questo crea delle assunzioni errate basandosi su dati non inerenti
(es: in un sistema che prevede i voti di un esame, il sistema considera pure il meteo)

L'underfitting invece, è il problema della formulazione di assunzioni troppo generiche e poco precise, date da uno scarso numero di dati ed etichette.
Bisogna quindi cercare un equilibrio tra queste due.

16.4.3. Transfer learning

Si definisce la tecnica di riutilizzo di un modello già sviluppato per altri task (simili), questo viene sfruttato come punto di partenza per il lavoro su un altro task. In alcuni tipi di problemi il modello può essere usato così com'è (**zero-shot learning**), in altri invece è necessario affinare il modello per lo specifico task (**fine-tuning**).

Il fine-tuning è un ulteriore addestramento del modello pregresso però su dati che sono rilevanti per il task che dovrà eseguire

La potenza del transfer-learning risiede nella possibilità di utilizzare un sistema preaddestrato quando si hanno pochi dati per addestrare un sistema da zero. (GPT e BERT si basano su questo paradigma)

16.4.4. Active Learning

È una tecnica basata su un'interazione tra il modello (detto active learner) e un operatore. Al modello vengono forniti dati non etichettati, questo poi invia query (dati non etichettati ma ritenuti più utili) ad un etichettatore che si occuperà di etichettarli; il modello userà poi i dati etichettati
La metrica più utile per questa tecnica è la *misura di incertezza*, ovvero, su campioni in cui il modello non ha problemi a predire correttamente non verranno etichettati altri dati, mentre l'etichettatura manuale avverrà su predizioni in cui il modello ha le maggiori incertezze
Un'altra euristica molto usata è il **query-by-committee**, ovvero:

- Si compone un *comitato* formato da diversi modelli (o con lo stesso ma su parti diversi dello stesso set)
- Si effettuano predizioni sulle etichette e si verificherà uno dei tre casi:
    - Tutti d'accordo

- Tutti disaccordo
- Pareri contrastanti
  - Si convoca un etichettatore per risolvere la contesa

L'active learning è sfruttato soprattutto nei contesti dei dati real-time perché riescono ad adattarsi molto facilmente al cambiamento dei dati

17. Bilanciamento delle classi

Uno dei principali problemi del ML è la presenza di uno sbilanciamento delle classi, questa è definitea come una grande differenza in termini di numeri di campioni tra le classi dei dati di training e questo porta a seri problemi di classificazione e di regressione.

Questo perché l'ideale di un sistema di machine learning è una distribuzione equa del numero dei campioni, in caso contrario è possibile parlare di *few-shot learning*, ovvero il caso in cui il sistema avrà pochi dati (di una classe) per imparare a riconoscere o ad inferire una determinata classe.

In alcuni casi però lo sbilanciamento è *volontario* per ottenere in modo più preciso possibile per inferire in contesti di problemi pesantemente sbilanciati.

Si possono gestire dei database sbilanciati tramite tre principali strategie:

- Metrics-level
- Data-level
- Algorithm-level

Questi hanno il compito di mitigare l'effetto dello sbilanciamento (non sempre risolvere del tutto)

17.1. Metrics-level

È una strategia basata sull'adozione di metriche scelte, utili per valutare lo sbilanciamento presente nei dati. Questo procedimento ovviamente dipende dalla scelta della metrica corretta.

Le più comuni *accuracy* e *errore rate* utilizzate per valutare un sistema di ML non sono tanto utili nella valutazione di un database sbilanciato, questo avviene perché esse trattano tutte le classi equamente e conseguentemente la classe maggioritaria influenzerà pesantemente questa metrica.

17.1.1. Precision, Recall, F1-Score

Le performance del modello (dati le sue inferenze) possono essere calcolate tramite Precision, recall e F1-Score. Precision e recall sono in controtendenza e valutano ognuno un tipo di precisione. F1-Score è utile invece per valutare in modo assoluto un sistema:

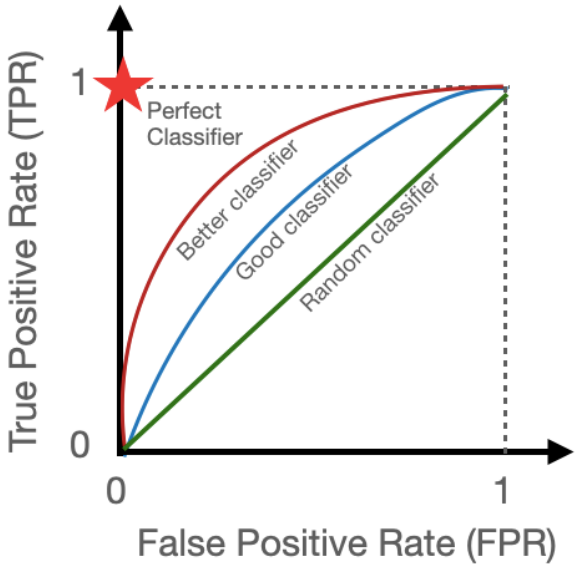
- $Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$ → Quante di quelle positive sono effettivamente positive
- $Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$ → Quanti campioni positive inferiti ci sono
- $F1\ Score = 2 * \frac{Recall * Precision}{Recall + Precision}$ → Valuta in modo adimensionale (assoluto) un sistema

17.1.2. ROC Curve

Un altro dei parametri molto usato è la **ROC Curve**, ovvero una tecnica basata sul trattare il task di classificazione come task di regressione, definendo (ed inferendo) la ***probabilità*** che un campione ha di

appartenere ad una classe. Per esempio in un problema di classificazione *positivo/negativo* è possibile definire una **soglia** oltre la quale si definisce positivo/negativo un campione.

Attraverso la ROC Curve è possibile osservare i cambiamenti di performance al cambiare della soglia detta **threshold**, questa è plottata come curva tra un indice di Recall e un indice di False Alarm. Chiaramente quando la **Recall** è pari ad 1 il modello è perfetto perché riesce ad individuare (e classificare) correttamente tutti i positivi:



17.2. Data-level

Questa tecnica si basa sul ricampionamento, le principali sono:

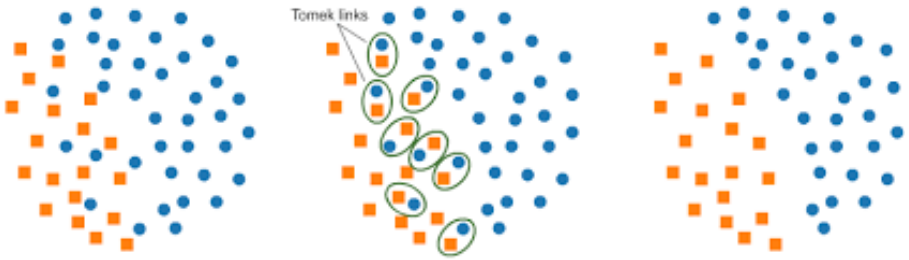
- Sottocampionamento
- Sovracampionamento
- Apprendimento a due fasi
- Campionamento dinamico

Queste si basano sull'aggiunta o la rimozione di campioni dal dataset con l'obiettivo di bilanciarlo.

17.2.1. Sottocampionamento

Si rimuovono dei campioni (randomicamente o tramite tecnica Tomek) dalla classe maggioritaria, riducendo la differenza con la classe minoritaria.

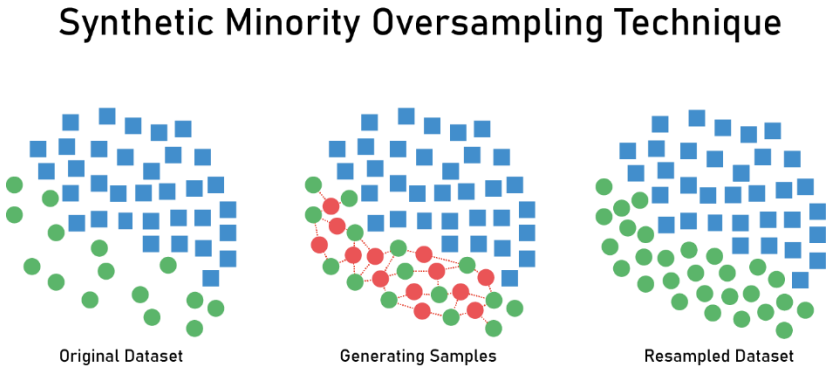
La tecnica **Tomek** (lavora bene su dati a bassa dimensionalità) si basa sulla generazione di coppie di campioni di classi opposte sulla base della loro vicinanza e si rimuove il campione della classe maggioritaria per ogni coppia:



17.2.2. Sovracampionamento

Si aggiungono sample artificiali della classe minoritaria, riducendo la differenza con la classe maggioritaria. Questo avviene randomicamente oppure tramite tecnica **SMOTE** (lavora bene con dati a bassa dimensionalità).

La tecnica SMOTE si basa sulla generazione di nuovi campioni campionando e generando nuovi campioni simili (interpolati) a quelli della classe minoritaria:



17.2.3. Apprendimento a due fasi

Sovracampionamento e sottocampionamento causano però (rispettivamente) overfitting e il rischio di perdere dati importanti, questi effetti possono essere mitigati dall'apprendimento a due fasi.

L'apprendimento a due fasi si basa su:

1. Sottocampionare il dataset fino ad avere N campioni per ogni classe,
2. Addestrare il modello con il dataset ricampionato,
3. Fine tuning del modello usando il dataset originale.

17.2.4. Campionamento dinamico

È un'altra tecnica che cerca di superare le limitazioni del sovracampionamento e del sottocampionamento.

Si basa sul "mostrare meno" la classe che ha già imparato a predirre, e conseguentemente vengono mostrati più campioni su cui si sono riscontrate più difficoltà

17.3. Algorithm-level

Queste tecniche lavorano sulle ***loss function***, cercando di minimizzare questa funzione di costo.

Si cerca di aggiustare dinamicamente i valori della loss function con l'obiettivo di dare maggiore priorità alle previsioni che producono una perdita maggiore. Si assegnano dei pesi maggiori ai campioni con maggiore priorità

18. Data Augmentation

Il processo di data augmentation è utile per la creazione di altri dati (sintetici) utilizzati per l'addestramento nel caso in cui il dataset coinvolga pochi dati. La causa dei pochi dati nel dataset ha cause esterne.

Quelle di data augmentation sono quindi tecniche per accrescere il dataset di addestramento; queste però hanno comunque necessità di un dataset come fondamento per sintetizzare dei nuovi dati.

Tra le tecniche più utilizzate troviamo:

- Simple label-preserving transformations
  - Questa si basa sulla preservazione dei label nel dataset
- Perturbation
  - Si basa sull'introduzione di perturbazioni volontarie
- Data synthesis

18.1. Simple Label-Preserving Transformations

È una delle tecniche più semplici utilizzate spesso in computer vision.

Si basa sulla modifica randomica dell'immagine preservandone però la sua etichetta, le modifiche possono essere di ritaglio, rotazione, inversione dei colori, cancellazione di parti dell'immagine.

Si distorce quindi l'immagine (contenuta nel dataset) conservandone l'etichetta. Questo processo produce un nuovo elemento nel dataset che avrà un aspetto "estetico" diverso ma un'etichetta uguale.

La medesima tecnica è utilizzata anche nel generare frasi per task di Natural Language Processing

18.2. Perturbation

La perturbazione è una tecnica che si basa sull'aggiunta di dati "rumorosi" a quelli già presenti nel dataset. Il fine di questa tecnica è quella di affinare il *decision boundary* (ovvero un confine utilizzato per la classificazione di un elemento). Esistono due tipi fondamentali per la perturbation:

- Simple
- Adversarial
    - Consiste nell'attaccare il modello (si cerca di confonderlo), con l'obiettivo di mettere in difficoltà il modello, aspettandosi che questi migliorino imparando come reagire a determinati tipi di dati

La tecnica però è da utilizzare in modo consapevole e ponderato perché può portare a perdite di performance del modello.

18.2.1. Adversarial augmentation

È una tecnica che si basa sull'individuazione del minimo noise injection necessario per confondere il sistema di ML. Questa tecnica non è però utilizzata nel NLP ma è più affine ai task di computer vision, questo perché un'immagine distorta può essere ancora riconosciuta mentre una frase con caratteri errati, o rumore in generale è molto più difficile da riconoscere.

Alcuni sistemi utilizzano questa tecnica per apprendimento piuttosto che per accrescere il dataset (BERT è uno di questi)

18.3. Data Synthesis

È una tecnica che si basa sulla generazione di dati sintetici, si utilizza quando la raccolta di dati reali è complessa, onerosa o anche per problemi di privacy. È una tecnica utilizzata in task di NLP o anche in task di computer vision.

Nel caso di una frase NLP possiamo utilizzare dei template tipo:

find me a {CUISINE} restaurant within {NUMBER} miles of {LOCATION}

E dato questo template, si generano delle frasi da utilizzare come dati.

Nei task di computer vision, si possono produrre dei mixup date le immagini nei dati di training (con label discrete), per creare delle label continue. Es:

Dati x1 e x2 etichettati rispettivamente come 0 e 1, si genera x' combinando x1 e x2:

$$x' = k\,x1 + (1-k)x2$$

L'obbiettivo di questa tecnica è quella di migliorare le performance, intese come:

- Aumento capacità di generalizzazione

- Ridurre problemi legati ad etichette corrotte
    - Sia per un'errata assegnazione che per un'errata rilevazione tipo tramite sensori
- Aumentare la robustezza del modello contro gli "esempi avversari"

Negli ultimi anni però sono stati sviluppati sistemi basati sull'uso di reti neurali per la generazione di dati (es. cycleGAN per la generazione di immagini)

19. Feature Engineering

Un sistema ML si basa non solo sulla raccolta di dati, bisogna estrapolare le caratteristiche o *feature* necessarie al sistema per le inferenze. Le fasi di feature engineering sono:

- Gestione Missing Value
- Scaling
    - Uniforma i fattori di scala dei singoli modelli
- Discretizzazione
    - Rendere discreti valori continui, rendendoli utili per la fase di apprendimento
- Feature Crossing
    - Da un insieme di feature è possibile combinarle per ridurle in numero
- Codifica Feature Crossing
    - Codifica (in senso di trasformazione) numerica delle feature, rendendo "vicine" le feature vicine
- Perdita dei dati
- Rilevamento perdita dei dati

Con features engineering si intende l'insieme di processi di selezione manipolazione e trasformazione di variabili dai dati grezzi. Rendendo possibile l'utilizzo di funzionalità per l'addestramento e previsione. Schematicamente:

- Creazione di funzionalità
    - Creazione di nuove variabili per il modello di ML (aggiungendo o rimuovendo funzionalità)
- Trasformazione
    - Trasformare le feature da un tipo di rappresentazione ad un'altra, aumentando o riducendo le funzionalità dal sistema
- Estrazione delle funzionalità
    - Processi atti ad estrarre features da un set di dati, con l'obiettivo di identificare informazioni utili
    - Con particolare attenzione al non distorcere le relazioni originali, riducendo la quantità di dati rendendoli più facilmente elaborabili dagli algoritmi
- Analisi esplorativa dei dati
    - Ispezione dei dati, tramite istogrammi o altri tipi di diagrammi, con il fine di ottenere e verificare l'andamento dei dati e la loro distribuzione

19.1. Tipi di valori mancanti

È l'operazione che prepara i dati per l'apprendimento automatico, i valori mancanti risultano una minaccia per l'apprendimento, è necessario quindi gestire queste situazioni.

Innanzitutto, si possono riscontrare 3 casi:

- Mancanti non casuali (MNAR)
- Mancanti casuali (MAR)
- Mancanti del tutto casuali (MCAR)

19.1.1. Mancanti non casuali (MNAR)

È il caso più complesso da analizzare, poiché non sono note le cause, si suppone che possa essere legato al valore stesso. Per esempio, un reddito mancante può indicare che questo è più alto della media o più basso o anche semplicemente nascosto per questioni di privacy

19.1.2. Mancanti casuali (MAR)

Il dato è legato ai dati osservati ma non a quelli non osservati, il motivo quindi non è legato al valore stesso ma legato ad un'altra variabile osservata

19.1.3. Mancanti casuali (MCAR)

In questo caso la mancanza di dati è indipendente dal dato osservato e da qualsiasi altro dato osservato.

In questo caso la mancanza di dati non ha una ragione specifica

19.2. Metodi di imputazione

L'imputazione è il processo tramite il quale si cerca di inferire i valori mancanti, con l'obbiettivo di migliorare il dataset il più possibile. Questi processi sono possibili tramite l'assunzione che i dati mancanti siano di tipo MAR (Missing At Random)

Esistono due classi principalmente:

1. Imputazione numerica (per valori numerici)
    a. Metodi deduttivi
    b. Metodi deterministici e stocastici
        i. Basati quindi su calcolo delle probabilità
2. Imputazione categorica (per valori categorici)

In qualsiasi caso, non esiste una tecnica perfetta, l'imputazione può sbilanciare il dataset e generare rumore, peggiorando il dataset. Mentre un'altra tecnica (**cancellazione**) può portare all'accentuazione di bias

19.2.1. Imputazione Numerica

19.2.1.1. Metodi deduttivi

Si sfruttano informazioni contenenti nel dataset per inferire il valore mancante, questo deducendo eventuali dipendenze funzionali possibili tra i dati disponibili, tramite valutazioni soggettive. (es. di un comune di residenza può essere dedotto che sarà legato alla provincia di residenza).

Sono stati costruiti metodi deduttivi per la data imputation. Questi però si basano comunque su una valutazione soggettiva legata alla conoscenza del dominio applicativo

19.2.1.2. Metodi Deterministici e Stocastici

Questo metodo, dati alcuni valori di statistica descrittiva (media, mediana, moda, … ecc) sostituisce tutti i valori mancanti con uno di questi. Questo si basa proprio sull'assunzione della mancanza randomica, inserendo infatti la media, la moda, la mediana o altri dati descrittivi. Il dataset non sarà troppo diverso in caratteristiche. I dati, però, non devono essere in scale diverse o in unità di misura diverse, non devono riferirsi a variabili diverse ecc.

Questo metodo, può anche essere sostituito dall'assegnazione di un valore di default al campo mancante

19.2.2. Imputazione Categorica

Il valore mancante tramite questa tecnica, viene sostituito dal valore con maggiore frequenza, questo però porta a problemi di sbilanciamento del dataset. Un altro approccio è utilizzare un'ulteriore categoria a cui assegnare tutti i valori mancanti.

20. Ridimensionamento (Scaling)

Il ridimensionamento è uno degli aspetti più importanti del ML, poiché inficia molto sull'efficienza.
Lo scaling è il processo che mira a garantire che tutte le funzionalità siano comparabili tra loro distribuendoli su una scala comune. Una delle applicazioni è ridimensionare il dataset da dati troppo frequenti che portano ad un'influenza enorme nell'apprendimento del sistema di ML.
L'utilità è quella di evitare problemi di Overflow o di Underflow in presenza di dati sbilancaiti e su scale completamente diverse.
Esistono due tecniche:

- Normalizzazione
- Standardizzazione

Un esempio di necessità di scaling è un dataset in cui in una colonna sono memorizzate le età (in cui l'età massima presente è 40) e una colonna in cui sono memorizzati i redditi annui (in cui il massimo è 150.000), il sistema non può distinguere la semantica di questi numeri, e potrebbe dare più importanza a 150.000 perché più grande, inferendo incorrettamente.

20.1. Normalizzazione

Una soluzione è **normalizzare**:

- Scaling nel range [0,1]
  - Data una variabile x, i suoi valori possono essere ridimensionati per rientrare nell'intervallo [0,1]. Questo avviene tramite:
  - $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$ in cui se x è il massimo dei dati allora x'=1, se x è minimo x'=0
- Scaling in range arbitrari [a,b]
  - $x' = \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$ in cui se x è il massimo dei dati, allora x' = b, se x è minimo x'=a

20.2. Standardizzazione

Un altro processo può essere quello di standardizzare, questo si fa supponendo che le variabili possano segurie una distribuzione normale. Questo avviene:

- $x' = \frac{x - \bar{x}}{\sigma}$

20.3. Processi di trasformazione

20.3.1. Log transformation

Altre tecniche di ridimensionamento sono quelle che correggono (o mitigano) l'asimmetria della distribuzione, una di queste è la **log transformation**, la quale applica la funzione logaritmica alle variabili della distribuzione

20.3.2. Discretizzazione

È una tecnica che consiste nel raggruppamento di valori continui (o discreti) di variabili in intervalli contigui.
È un processo a perdita, in quanto causa la perdita di informazioni.
È una tecnica che risulta utile per migliorare la convergenza, determinando il minimizzando il numero di intervalli possibili massimizzando però la preservazione delle informazioni nei dati.
È quindi un processo che da un dati continui (o discreti) estrae dati discreti, definendo i limiti degli intervalli.

20.3.2.1. Vantaggi

- Alcuni modelli di regressione funzionano meglio con valori discreti
- Accelera il processo di addestramento
- Riduce al minimo l'influenza dei valori anomali, posizionandoli negli intervalli inferiori (o superiori) insieme ai restanti valori della distribuzione

Esistono varie tecniche di discretizzazione:

- Discretizzazione ad uguale ampiezza
- Discretizzazione ad uguale frequenza
- Approcci non supervisionati
- Approcci supervisionati

20.3.2.2. Discretizzazione ad uguale ampiezza

È una tecnica che consiste nel dividere l'intervallo di valori continui in k intervalli di uguale dimensione,

es. dato un range che va da 0 a 100, se vogliamo dividere in k=2 intervalli, allora gli intervalli: (0-50),(50-100)

Questo tipo di discretizzazione non altera in modo drammatico la distribuzione delle variabili, questa può essere verificata confrontando istogrammi (variabile continua) e grafici a barre

20.3.2.3. Discretizzazione ad uguale frequenza

È una tecnica che consiste nell'ordinare la variabile continua in intervalli con lo stesso numero di osservazioni, la larghezza di questi intervali è determinata dai quantili. Risulta particolarmente utile in variabili asimmetriche

20.3.2.4. Approcci non supervisionati

Si tratta di tecniche non supervisionate perché queste determinano i limiti degli intervalli in modo autonomo, senza direzioni del progettista. Vengono utilizzati algoritmi di clustering (**k-means**), questo *partiziona* i valori in base alla loro similarità. La discretizzazione con k-means richiede quindi un unico parametro (ovvero quanti k cluster devono essere prodotti), anche se può lavorare senza una direzione di quanti cluster sono necessari

20.3.2.5. Approcci supervisionati

Sono tecniche basate sui **Decision Tree**, questo perché rendono possibile determinare automaticamente i punti di taglio e il numero ottimale di divisioni. Queste tecniche si basano sulla costruzione di un albero di decisione, partendo dalla valutazione di tutti i possibili valori di una caratteristica, partizionando ulteriormente (punti di taglio) tramite una metrica (misura di impurità) come **l'entropia** o **l'impurità di Gini**. Queste metriche rendono possibile identificare in modo "efficiente" (anche se approssimato) i punti di taglio. Questa suddivisione viene ripetuta su ogni nodo dell'albero, fino al raggiungimento di un **criterio di arresto**. La costruzione di questo albero, però, si basa sulla disponibilità di un dataset etichettato.

21. Feature Crossing

È un processo che si basa sull'incrocio di due o più funzionalità, generando nuove caratteristiche. Risulta utile nel modellare le relazioni non lineari tra nuove caratteristiche. Risulta poi essenziale per modelli che non sono validi per l'apprendimento di relazioni non lineari (come per esempio la regressione lineare, regressione logistica e modelli basati su alberi). Questo processo rende possibile a questi modelli di inferire su questi dati, facendoli "apparire" come una relazione lineare e quindi permettendo a questi di studiarli.

Bisogna però prestare attenzione al feature crossing, poiché si moltiplicano il numero di valori possibili, peggiorando l'algoritmo invece di aiutarlo a convergere.

Questo aumento di valori può portare a problemi di overfitting o underfitting

22. Naive Bayes (Esercitazione)

Dato un data sample con n feature (x1,x2,xn). Calcola la probabilità che questo sample appartenga ad ognuna delle k classi (y1,y2,yk) . Si considera la probabilità P(yk|x) a cui si applica il teorema di bayes

P(yk) rappresenta come sono distribuite le classi, senza conoscere ulteriori caratteristiche, è detto **prior**

Il prior può essere predeterminato (es. ogni classe è equiprobabile) oppure può essere appreso dai dati di apprendimento

P(yk|x) è detta **posterior** ovvero una conoscenza aggiuntiva dovuta all'osservazione

P(x|yk) è detta **likelihood** ovvero la distribuzione congiunta di n feature considerando che il sample appartenga ad yk

Questi calcoli diventano ovviamente più difficili all'aumentare delle feature. Però per la proprietà naive, consideriamo tutte le feature indipendenti tra loro, quindi:
P(x|yk) = P(x1|yk) * P(x2|yk) * … * P(xn|yk)

P(x) è detta **evidenza** e dipende esclusivamente dalla distribuzione complessiva delle feature

Per il teorema di bayes possiamo dire che il **posterior** è proporzionale al **prior** e alla **likelihood**

Nel caso le osservazioni siano nulle, si può utilizzare un processo di smooting, per esempio quello di laplace(in un caso specifico):

$$P(f = 1|N) = \frac{\boldsymbol{P(f = 1 \cap N) + a}}{\boldsymbol{a + (a * k)}}$$

Consideriamo il numeratore, il valore nullo sarebbe quello di P(f=1 inters N), a questo si somma un alfa arbitrario

Al denominatore moltiplichiamo l'alfa scelto con il numero di classi (per esempio 2 se la classificazione è 0 o 1)

23. Algoritmi Tree-Based

23.1. Caratteristiche Generali

Le feature che possono essere contenute in un dataset possono essere di vario tipo:

- Qualitative
  - Feature che caratterizzano tramite un attributo (es. Rosso, Giallo, Blu)
- Quantitative
  - Si utilizza il significato numerico dei dati (es. 0,1 12 10,4)
- Categoriche
  - Può utilizzare valori numerici che vanno però interpretati non solo per il loro valore "nominale"

Gli alberi di decisione, quindi, sono grafi radicati che possono essere sfruttati data la loro forma per illustrare tutte le possibili alternative decisionali e i risultati corrispondenti, questo tramite il percorso dalla radice alle foglie passando per dei nodi intermedi detti ***decision node*** utilizzando rami di collegamento chiamati ***branch***. Esistono quindi tanti branch (per nodo) quante sono le possibili risposte, questo dipende dal ***criterio di split***.

Un classificatore a Decision Tree, opera sottoforma di albero decisionale, quindi:

1. Mappa le osservazioni in assegnazioni di classe
   a. Nodi foglia
2. Attraverso una serie di test
   a. Nodi intermedi
3. Mappa le condizioni corrispondenti utilizzate per classificare
   a. Rami

23.2. Costruzione di un albero di decisione

Il processo segue questi passaggi:

- Si suddividono i campioni di addestramento in sottoinsiemi
- Si ripete il processo in modo ricorsivo
- Per ogni partizione creata vengono eseguiti test per verificarne l'accuratezza

La suddivisione in branch può avvenire in più modi, questa può essere binaria nel caso i risultati dei test per ogni nodo intermedio siano di ordine due, ma ne esistono di ordine n (specialmente per le quantitative e qualitative numeriche). La suddivisione termina quando ulteriori suddivisioni non migliorano la purezza.

23.3. Algoritmi per Alberi di Decisione

Esistono vari algoritmi per costruire correttamente un albero decisionale:

- ID3
- C4.5
- CART
- CHAID

Questi algoritmi eseguono scelte greedy per la costruzione dell'albero sfruttando ognuna una metrica precisa

23.3.1. Iterative Dichotomiser 3 (ID3)

Effettua una ricerca greedy top-down. Selezionando il miglior attributo su cui dividere il set di dati a ogni iterazione, senza backtracking. Effettua quindi, passo passo, la scelta di decisione in decisione.

23.3.2. C4.5

Effettua una ricerca greedy top-down, risulta simile a ID3 solo che implementa il backtracking, permettendo la rivalutazione del branch nel quale si trova.

23.3.3. CHAID

Crea combinazioni di criteri di selezione, valutando tutte le possibili combinazioni e ne valuta anche l'efficienza tramite modelli statistici (Test del chi quadro)

23.3.4. CART

Si costruisce ponendo condizioni IF-ELSE su ogni nodo, creando split binari, si differenzia da ID3 solo che CART può produrre solo split binari. L'algoritmo cerca in modo greedy la combinazione più significativa di una caratteristica e del suo valore, queste combinazioni vengono testate utilizzando funzioni di misurazione.

L'algoritmo divide il set utilizzando quindi delle condizioni di split binarie, ovvero, che ricadono in una caratteristica o nel suo opposto.

23.4. Criteri di Arresto

I criteri di arresto possono essere di due tipi:

- Numero minimo di campioni
    - Quando la distribuzione dei campioni nelle classi non può essere migliorata da ulteriori split
- Profondità massima dell'albero
    - Può capitare di specializzare troppo dei branch, adattando troppo il modello ai dati di addestramento, causando sicuramente overfitting.

Un'altra condizione di arresto si verifica nel caso in cui un nodo senza rami diventa foglia, e la classe dominante di questo verrà considerato come risultato

23.5. Metriche per misurare lo split

Queste metriche vengono utilizzate per valutare ogni split, per quanto riguarda i decision tree si usano principalmente:

- Impurità di Gini
- Information Gain

23.5.1. Impurità di Gini

Misura il tasso di **impurità**, per un insieme di dati con K classi, supponiamo che i dati della classe k occupino una frazione fk dell'intero insieme di dati, questo tasso si calcola come:

$$\text{Gini Impurity: } 1 - \sum_{k=1}^{K} f_k^2$$

24. Sviluppo dei modelli

Eseguita la fase di scelta e creazione dei dati e dell'insieme di feature, è possibile iniziare finalmente a selezionare il modello di ML adatto al problema, non esiste un algoritmo perfetto per tutto ma bisogna sceglierne uno adatto. La scelta dell'algoritmo però è molto meno importante rispetto alla quantità e qualità dei dati utilizzati per l'addestramento.

24.1. Valutazione dei modelli di ML

La scelta dell'algoritmo per il modello di ML è guidata da alcuni fattori come: la familiarità con un determinato algoritmo, il "trend" del modello del momento, ma soprattutto pesano sulla scelta di questi algoritmi le risorse disponibili (intese in senso economico e di tempo).

Negli ultimi anni il "trend" è quello di utilizzare modelli basati sul deep learning o reti neurali, questo per l'influenza e il miglioramento nel campo dell'intelligenza artificiale.

Nonostante il trend sviluppatosi, spesso gli algoritmi di ML "classici" risultano più efficienti, spesso sono addirittura preferiti per la loro semplicità, in altri casi sono addirittura in simbiosi i due algoritmi (**ensamble**), questi ensamble spesso risultano migliori del "solo" algoritmo. Un esempio di questa simbiosi è l'utilizzo di un modello di clustering k-means per estrarre le feature da un set, utilizzate poi in una rete neurale.

È quindi importante scegliere algoritmi utili alla risoluzione del problema, come per esempio:

- Si ha un problema di identificazione di transazioni fraudolente (problema di anomaly detection), e gli algoritmi più utilizzati sono i k-nearest neighbors, reti neurali o clustering o altre.

La scelta avverrà quindi tra questi algoritmi "indicati"

Come già anticipato, caratteristiche tipo quella della potenza di calcolo richiesta, o il numero di etichette o in generale le risorse che l'algoritmo richiede, sono necessarie per la scelta di questi. In aggiunta però, è necessario prendere in considerazione anche ***l'interpretabilità*** del risultato inferito dal sistema, e in questo gli algoritmi classici sono superiori (ovvero mostrano quanta influenza ha un certo dato sul risultato)

Oltre all'interpretabilità, è possibile valutare il sistema scelto in base ai risultati prodotti. Si utilizzano quindi indici di accuratezza (accuracy) e latenza di inferenza (velocità di predizione)

24.2. Sei consigli per la scelta del modello

- **Non seguire esclusivamente "lo stato dell'arte" o l'ultimo trend** in quanto ad algoritmi o modelli,
    - questi spesso vengono testati solo in ambito accademico e di ricerca e potrebbe non essere efficiente in ambito aziendale o di produzione,
    - spesso è stato testato su dati di una determinata fattura ottimizzati per il funzionamento di questo sistema e non su dati che si potrebbero trovare in ambito aziendale o in contesti reali
- **Iniziare con modelli semplici**, questo perché:
    - sono più facili da distribuire e quindi di effettuare calcoli in modo efficiente,
    - perché sono più facili da incrementare e da comprendere
    - è utile come riferimento nel caso di passaggio ad altri modelli (utile come paragone di base)
- **Evitare i bias**
    - Bias molto frequenti possono essere per esempio la preferenza dello sviluppatore per un determinato modello con conseguente sviluppo molto più accurato degli altri (rendendo il confronto difficile)
    - Scelta dei pesi dei dati sul risultato
- **Valutare le performance attuali rispetto a quelle future**, si intende:

    - È utile valutare la precisione e le performance in generale del modello con i dati attuali, queste valutazioni possono essere rappresentate con le curve di apprendimento
- **Valutare i compromessi**
    - Intesi come tolleranza per falsi positivi e falsi negativi, quindi quanto dev'essere preciso e cosa importa di più tra falsi positivi e falsi negativi, per esempio in un sistema di riconoscimento di impronte digitali sono molto più importanti da minimizzare i falsi positivi piuttosto che i falsi negativi poiché potrebbe dare permesso di accesso per errore a chi non deve avere accesso, o in ambito medico è più importante minimizzare i falsi negativi, in quanto è meglio avere dei falsi positivi che possono essere risolti
    - Altri compromessi possono essere su interpretabilità e accessibilità
    - Compromessi sulle risorse
- **Comprendere le ipotesi di un modello**
    - È importante capire le ipotesi che il modello crea per rappresentare la realtà di interesse, come per esempio dov'è posto il confine in un classificatore lineare

24.3. Ensamble

Si definisce ensamble l'unione e la collaborazione in simbiosi di più modelli per la risoluzione di un problema. Ogni modello presente in questo insieme è detto **base learner**, la creazione di questi agglomerati parte dallo sviluppo di un singolo sistema e opportunamente altri sistemi.

Spesso questo tipo di soluzione non è preferito in ambito di produzione in quanto sono molto più complessi da distribuire e da manutenere, anche se, effettivamente risulta molto efficace soprattutto in problemi in cui anche se il minimo miglioramento è importante.

Per esempio, utilizzando un ensamble con voto a maggioranza, se 3 sistemi hanno tutti la stessa probabilità del 70% di effettuare una predizione esatta e 30% errata, la probabilità che almeno 2 di questi sistemi siano corretti e del 78,4%

dato da $(0{,}7^3) + 3 * (0{,}7 * 0{,}7 * 0{,}3) = 0{,}784$

in cui il primo addendo è il caso in cui tutti e 3 sono corretti e il secondo addendo è il caso in cui 2 dei 3 sistemi sono corretti

Questo si basa sull'assunzione che tutti e 3 i sistemi siano indipendenti l'uno dall'altro

Creare un ensamble

Esistono varie tecniche per la creazione di un ensamble:

- Bagging
- Boosting
- Stacking

24.3.1. Bagging

È l'abbreviazione di ***bootstrap aggregating*** ed è progettato per migliorare la stabilità dell'addestramento e l'accuratezza degli algoritmi di ML, riesce a ridurre la varianza e aiuta ed evitare l'overfitting, consiste nel **campionare con sostituzione** un dataset per creare diversi set di dati detti **bootstrap**, questo tipo di campionamento permette ogni volta la creazione indipendente dei bootstrap. Vengono creati classificatori per ogni bootstrap creato e infine questi vengono aggregati in un classificatore ensamble.

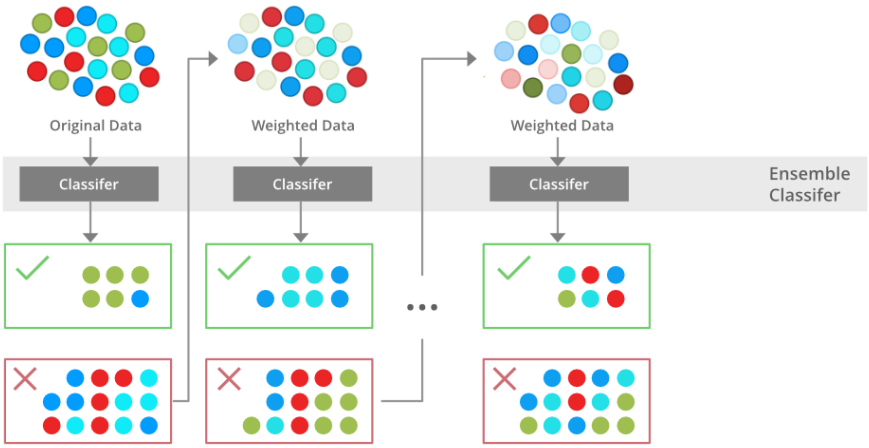
Questa tecnica è molto utile per stabilizzare metodi "instabili" come per esempio le reti neurali, classification e regression tree ecc…

Questa tecnica però può anche portare ad una degradazione delle performance per algoritmi già stabili come il KNN (K-nearest-neighbor). Uno degli esempi più lampanti di Bagging è la **random forest**

Il funzionamento di questa tecnica è quello di evitare correlazioni nei dati dando ad ogni classificatore un set diverso di dati (che però è preso dallo stesso set originario)

24.3.2. Boosting

È una famiglia di algoritmi di ensamble iterativi, che converte learner deboli in learner forti. I vari learner del ensamble viene addestrato sullo stesso set di campioni, ma, per evitare correlazioni ad ogni iterazione vengono cambiati i "pesi" dei vari dati analizzati dai learner. Queste iterazioni sequenziali hanno come obbiettivo quello di fortificare i learner che proveranno ad essere meglio dell'iterazione precedente provando a concentrare il learner indirizzandolo a risolvere gli errori commessi nell'iterazione precedente.



Nella prima iterazione il classificatore effettuate le sue previsioni, produce un insieme di dati che sono coinvolti in previsioni errate. Da questo insieme poi viene definito un peso diverso per i dai nel dataset così da porre attenzione sui dati "errati" questo dataset "ponderato" viene poi elaborato (nella seconda iterazione e nelle successive) producendo un insieme di errori, ripetendo poi le stesse operazioni

24.3.3. Stacking

Lo stacking consiste nell'uso di un pool di base learner sui dati di addestramento, poi creare un meta-learner che date le previsioni dei base learner produce una previsione finale. Questo meta-learner può essere semplice come un voto a maggioranza o una media ponderata dei risultati dei base learner o anche più complesso nel caso questo possa essere un altro modello che usa i dati prodotti dai base learner per produrre un risultato finale

24.4. Experiment Tracking e Versioning

Nell'ambito del machine learning è molto importante la tracciabilità di un esperimento, queste informazioni devono garantire la ripetibilità dell'esperimento con i relativi artefatti (risultati). Queste informazioni permettono il confronto tra diversi esperimenti così da poter verificare come anche un piccolo cambiamento può definire modifiche grosse. Il processo di tracciamento dei progressi e dei risultati è detto **Experiment Tracking** e il processo di memorizzazione è detto **Versioning**

24.4.1. Experiment Tracking

Come già detto, tenere traccia dei progressi dei vari esperimenti è necessario per ottenere risultati buoni. È in questa fase che è possibile:

- Scegliere una metrica precisa per valutare le prestazioni del modello

- Effettuare logging del campione utilizzato come input al modello
- La velocità del modello (step al secondo o numero di token elaborati al secondo)
- Misurazioni delle prestazioni del sistema (utili per individuare colli di bottiglia)
- Parametri ed iperparametri le cui variazioni possono influenzare le performance del modello (parametri come il learning rate)

24.4.2. Versioning

È necessario, nella fase di versioning non solo effettuare versioning del codice ma anche dei dati per permettere la ripetibilità dell'esperimento. Per la prima, la stringa di codice è relativamente breve, mentre nel caso del versioning dei dati queste possono essere tanto voluminosi. Il versioning del codice permette quindi a più sviluppatori di lavorare sullo stesso codice, è quindi suggeribile utilizzare una fonte condivisa per quanto riguarda i dati piuttosto che clonare un intero dataset ogni volta.

Si definisce **diff** la differenza tra i vari versioning. Negli ultimi anni è stato definito un diff basato sul checksum. Possono quindi presentarsi problemi di merging, questi vengono risolti facendo prevalere uno sull'altro e non misti dato che non esiste un esperimento misto ma solo i due singoli.

Nel caso di dati protetti da privacy e per legge costretti ad essere eliminati dopo un po', alcune versioni potrebbero essere inutilizzabili per motivi legali

24.5. Debug modelli di ML

Il debugging, come per qualsiasi software è necessario. Tuttavia, per i modelli di ML questo processo è molto più complicato:

- **Il fallimento è silenzioso**
  - I fallimenti del modello non sono sintattici o identificabili immediatamente, il modello potrebbe anche diminuire la loss function. Però può produrre comunque previsioni errate
- **È lento verificare che un bug sia stato risolto**
  - Poiché un sistema di ML richiede un lungo processo di addestramento, questo è necessario ad ogni modifica del modello riaddestrarlo. Questo processo appunto richiede tanto tempo
- **Un sistema di ML è molto complesso**
  - Essendo composto di varie componenti è difficile identificare l'elemento che causa il fallimento del modello

Le cause del fallimento di un modello di ML possono essere varie:

1. **Vincoli teorici**
   a. Ogni modello genera previsioni basandosi sulle ipotesi sul quale è basato, queste potrebbero essere errate dal principio, rendendo fallace tutto il sistema
2. **Scelta inadeguata degli iperparametri**
   a. La scelta degli iperparametri(es. profondità dell'albero, numero minimo di campioni) influisce tantissimo sulla convergenza del sistema verso una previsione corretta
3. **Problemi nei dati**
4. **Scelta errata delle feature**

Per il debugging è utile sfruttare l'automazione della famiglia degli algoritmi ***AutoML***

24.5.1. AutoML

È una soluzione sviluppata nel 2018 utilizzata per "giocare" con gli iperparametri di un sistema di ML cercando di ottimizzare il suddetto sistema configurando nel modo più corretto i parametri.

24.5.1.1. Soft AutoML

Si definisce **Soft AutoML** il l'approccio più popolare e semplice(leggero) degli AutoML, questo si occupa della regolazione degli iperparametri. Questo perché, come preannunciato, la scelta di iperparametri diversi su un modello identico può influenzare enormemente l'efficienza del sistema o le sue capacità. È stato dimostrato come modelli "deboli" con iperparametri ben settati possano essere superiori a modelli più "forti"

24.5.1.2. Hard AutoML

A differenza del Soft, l'hard automl permette non solo una verifica dei migliori iperparametri, bensì riesce a valutare anche la propria architettura. Queste possono essere "finte" come un iperparametro per lo skip layer o un layer convoluzionale

24.6. Valutazione offline di un modello

La valutazione di un modello dovrebbe (idealmente) essere la stessa sia in fase di produzione che in fase di sviluppo, questo però spesso non è possibile per vari casi, come per esempio una valutazione "falsata" poiché il modello potrebbe avere dei buoni risultati solo sui dati di test e non sui dati reali. La valutazione, quindi, non può essere effettuata sui dati di testing come in fase di sviluppo. In alcuni casi i dati di test (in produzione) sono ricavabili dal feedback degli utenti, per altri problemi questo non è possibile.

24.6.1. Baseline

Non è utile basarsi solo sulle metriche di valutazione, è importante conoscere la base di riferimento del modello per poterlo valutare correttamente.

Cinque baseline che possono essere utili nella valutazione del modello (possono variare da caso d'uso a caso d'uso):

1. Random baseline:
    a. Le predizioni sono generate a caso seguendo una distribuzione specifica (o uniforme o delle etichette)
2. Euristica semplice
    a. Le previsioni sono basate su semplici euristiche
3. Zero rule baseline
    a. Caso specifico di euristica semplice in cui il modello predice sempre la classe più comune
4. Human baseline
    a. Le predizioni vengono comparate con il giudizio di esperti umani
5. Soluzioni esistenti
    a. Si confrontano le predizioni con i risultati prodotti da soluzioni già esistenti e affermate
        i. Non sempre il modello dev'essere migliore (in performance) delle soluzioni esistenti per essere utile, anche un modello con performance leggermente più scarse può risultare utile

24.6.2. Metodi di valutazione

La valutazione è basata principalmente dall'ambito di applicazione del sistema di ML, in ambito accademico si potrebbe preferire una metrica sulla performance mentre in ambito aziendale si potrebbe scegliere un modello più robusto, semplice ecc

24.6.2.1. Test di perturbazione

Un modello deve saper lavorare correttamente (e in modo ottimale) su dati "puliti", ma dovrebbe poter sopportare e lavorare correttamente anche con dati più disordinati, rumorosi o "sporchi".

L'idea alla base dei test di perturbazione è quello di "rinforzare" e testare il modello utilizzando dei dati di più rumorosi per valutare la robustezza (o sensibilità) del modello. Più questo sarà sensibile e più avrà problemi nel gestire dati rumorosi e più avrà bisogno di manutenzione col passare del tempo.

24.6.2.2. Test di invarianza

A leggere variazioni di input il sistema dovrebbe rispondere con il medesimo output. Se questo non avvenisse il modello potrebbe essere inutilizzabile. Questa caratteristica definisce anche la *fairness* del modello. Il caso di una variazione involontaria (e non-fair) dell'output ad una leggera variazione in input può essere sia un bonus (nel caso sia ricercato e utile) o da risolvere nel caso producano output errati e influenzati da bias.

La soluzione a questo problema è escludere informazioni sensibili (non utili ai fini del modello).

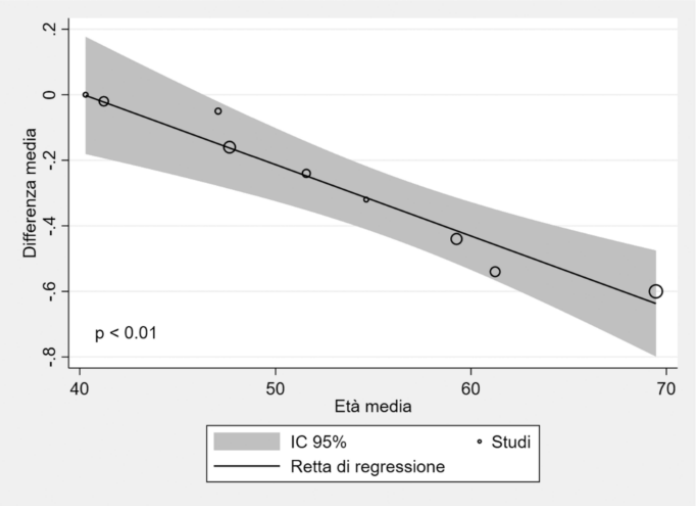
24.6.2.3. Calibrazione del modello

La calibrazione di un modello è molto importante, questa definisce quanto e se le previsioni del modello sono precise. Se questo asserisce di prevedere correttamente nel 70% dei casi, questo deve valere sempre. Quando in seguito ad un testing queste probabilità vengono dimostrate false, il modello si dice *non calibrato*.

La calibrazione di un modello si raffigura plottando sull'asse x la probabilità e sull'asse y la frequenza che la predizione si avvera. Un modello perfettamente calibrato segue la retta y=x

24.6.2.4. Misura di confidenza

Si può considerare la misura di confidenza come soglia entro la quale la previsione può essere utile, es:



Entro questa soglia (raffigurata dall'alone) le predizioni possono essere "utili"

24.6.2.5. Valutazione slice-based

I dati vengono suddivisi in sottogruppi, e per ognuno, si valutano metriche a grana grossa (precision, recall, F1-Score, accuracy, ecc). Questa valutazione è però può causare altri problemi.

Studiando con un esempio, un sottogruppo maggioritario dei dati ottiene una valutazione di accuratezza del 98%, mentre sul secondo gruppo minoritario raggiunge l'80%. Complessivamente questo sistema avrà un'accuratezza del 96,2%

Mentre un secondo modello che valuta correttamente entrambi i sottogruppi al 95% (con valutazione complessiva 95%)

La scelta tra i due modelli è indirizzata principalmente dallo scopo del modello e non solo dalla valutazione complessiva del modello; infatti, per un determinato problema potrebbe essere più importante valutare accuratamente un gruppo rispetto alla valutazione complessiva (preferirebbe quindi una valutazione più alta su un gruppo piuttosto che una valutazione complessiva più alta). **Paradosso di Simpson**

La valutazione Slice-Based avviene secondo tre approcci principali:

- **Basati su euristica**
    - Si suddividono i dati in base alla conoscenza che si ha di questi (richiede un'alta confidenza con i dati)
- **Analisi degli errori**
    - Esaminare manualmente gli errori commessi dal sistema cercando pattern significativi
- **Slice finder**
    - Si generano sottogruppi utilizzando algoritmi come beam search o di clustering