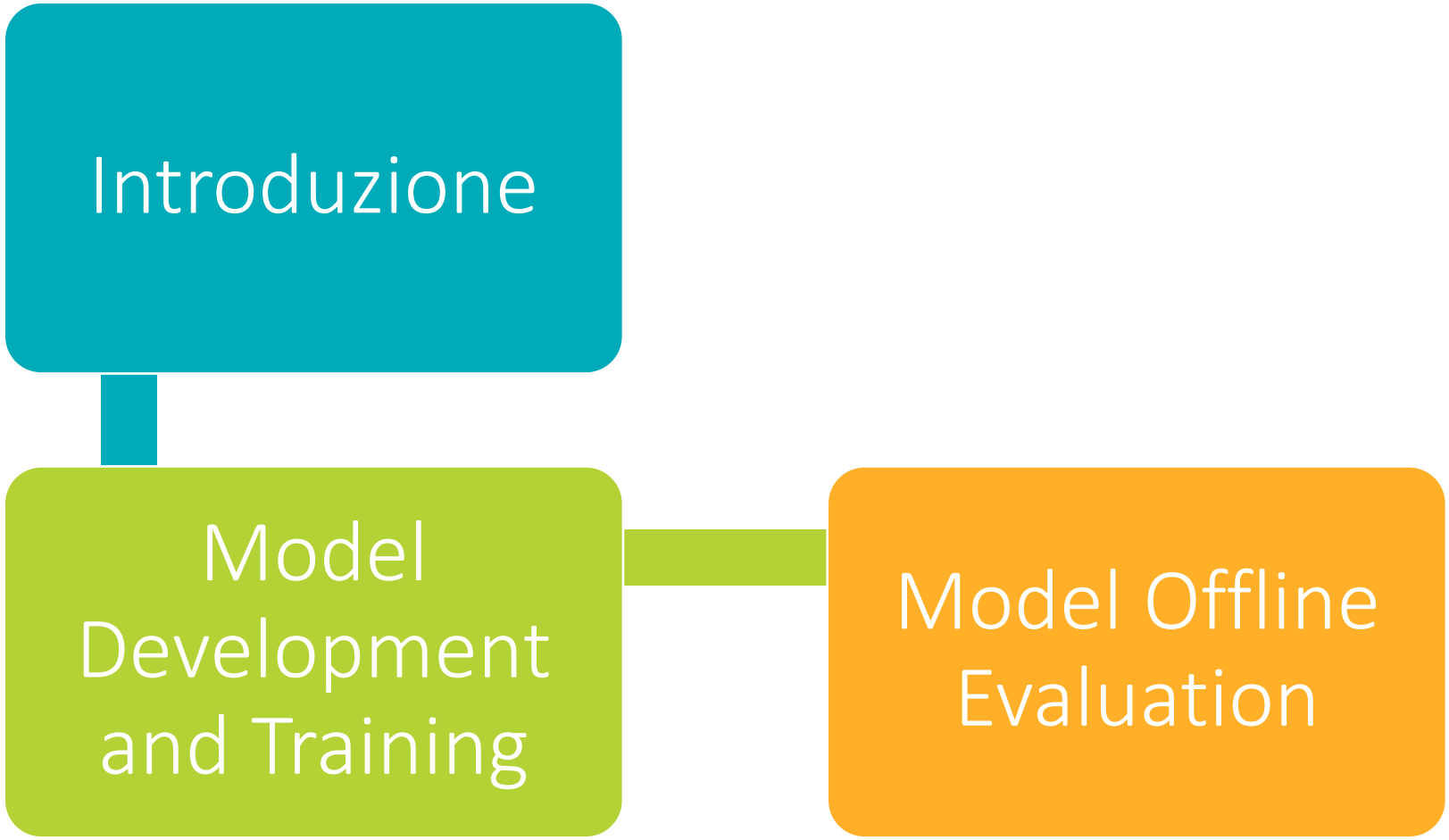
Machine Learning

5 - Model Development and Offline Evaluation

Prof. Giuseppe Polese Prof.ssa Loredana Caruccio

Outline


Introduzione
Model Development and Training
Model Offline Evaluation

Introduzione

1

Introduzione

- Nelle precedenti lezioni abbiamo trattato la creazione dei dati di addestramento e abbiamo discusso su come definire delle feature a partire da tali dati
- Una volta costruito un insieme di feature iniziale, possiamo iniziare lo studio degli algoritmi di ML integrati all'interno dei sistemi
- Esistono molti algoritmi di ML e altri sono in fase di sviluppo
- In base a diversi aspetti si possono identificare i migliori algoritmi da adottare per la risoluzione di uno specifico task

Argomenti

- In questa lezione analizzeremo diversi aspetti dello sviluppo di un modello:
  - **Debug**
  - **Experiment Tracking**
  - **Versioning**
  - **Addestramento distribuito**
  - **AutoML**
- Inoltre, studieremo le migliori tecniche per valutare le performance di un modello prima di distribuirlo in produzione

Model Development and Training

2

Valutazione dei modelli di ML

- Dato un task che può sfruttare un modello di ML per la sua risoluzione, come scegliamo quale algoritmo utilizzare?
  - Utilizziamo un algoritmo che conosciamo già?
  - Proviamo un modello che dovrebbe essere il nuovo stato dell'arte per la risoluzione del task?
- Se il tempo e la potenza di calcolo fossero illimitati, la cosa più razionale da fare sarebbe provare tutti gli algoritmi e vedere quale fornisce i migliori risultati
  - Tuttavia, il tempo e la potenza di calcolo sono risorse limitate e bisogna essere strategici nella scelta dei modelli

Valutazione dei modelli di ML

- Dato che la maggior parte dei progressi dell'IA nell'ultimo decennio è avvenuta grazie al Deep Learning (DL) e a reti neurali sempre più grandi e profonde, si potrebbe pensare che il DL stia sostituendo gli algoritmi di ML classici
- Tuttavia, sebbene il DL stia trovando sempre più casi d'uso, gli algoritmi di ML classici non stanno scomparendo
  - Ad esempio, molti sistemi di raccomandazione si basano ancora sul filtraggio collaborativo e sulla fattorizzazione delle matrici
  - Molti task di classificazione con requisiti di latenza rigorosi utilizzano ancora algoritmi basati su alberi, ad esempio i gradient-boosted trees

Valutazione dei modelli di ML

- Anche nelle applicazioni in cui vengono impiegate le reti neurali, i classici algoritmi di ML vengono ancora utilizzati in tandem
- Ad esempio, le reti neurali e i decision tree possono essere utilizzati insieme in un ensemble
- Oppure, un modello di clustering k-means può essere utilizzato per estrarre le feature da inserire in una rete neurale
- Viceversa, una rete neurale pre-addestrata potrebbe essere utilizzata per generare rappresentazioni vettoriali da inserire in un modello di regressione logistica

Esempi

- Quando bisogna selezionare un modello per un problema, non si sceglie tra tutti i modelli possibili, ma ci si concentra su un insieme di modelli adatti al problema
- Ad esempio, se il vostro capo vi dice di costruire un sistema per rilevare i tweet tossici, sapete che si tratta di un problema di classificazione del testo – dato un testo, classificare se è tossico o meno – e i modelli comuni per la classificazione del testo sono
  - **Naive Bayes**
  - **Regressione Logistica**
  - **Reti Neurali Ricorrenti**
  - **…**

Esempi

- Se il vostro cliente vuole che costruiate un sistema per rilevare transazioni fraudolenti, sapete che questo è riconducibile al classico problema di rilevamento delle anomalie – le transazioni fraudolenti sono anomalie che volete rilevare – e i modelli comuni per questo problema sono molti, tra cui
  - **K-Nearest Neighbors**
  - **Reti Neurali**
  - **Clustering**
  - **…**
- La conoscenza dei task comuni di ML e degli approcci tipici per risolverli è essenziale in questo processo

Valutazione dei modelli di ML

- I diversi tipi di algoritmi richiedono un numero diverso di etichette e una diversa potenza di calcolo
- Alcuni richiedono un addestramento più lungo di altri, mentre altri impiegano più tempo per fare previsioni
- Gli algoritmi classici di ML tendono a essere più interpretabili (ad esempio, quali feature hanno contribuito maggiormente a classificare un'e-mail come spam) rispetto alle reti neurali

Valutazione dei modelli di ML

- Quando si valuta quale modello utilizzare, è importante considerare non solo le performance, misurate tramite metriche come l'Accuracy e l'F1-score, ma anche altre proprietà, come
  - **La quantità di dati, il calcolo e il tempo necessari per l'addestramento**
  - **La latenza dell'interferenza**
  - **L'interpretabilità**
- Ad esempio, un semplice modello di regressione logistica potrebbe avere un'accuratezza inferiore rispetto a una rete neurale complessa, ma richiede meno dati etichettati, è molto più veloce da addestrare, è molto più facile da implementare ed è anche molto più facile spiegare perché sta facendo certe predizioni

Sei consigli per la selezione dei modelli

- **Evitare la trappola dello stato dell'arte:** I ricercatori spesso valutano i modelli solo in ambito accademico, per cui un modello migliore allo stato dell'arte spesso significa che ha prestazioni migliori rispetto ai modelli esistenti su alcuni set di dati statici
  - Ciò non implica che il modello è abbastanza veloce o economico da poter essere implementato
  - Inoltre, non è detto che il modello avrà prestazioni migliori rispetto ad altri modelli sui vostri dati
- Sebbene sia essenziale tenersi aggiornati sulle nuove tecnologie, la cosa più importante da fare quando si risolve un problema è trovare le soluzioni che possono risolverlo. Se esiste un modello che è molto più economico e semplice di modelli più avanzati, utilizzate la soluzione più semplice

Sei consigli per la selezione dei modelli

- **Iniziare con i modelli più semplici:** La semplicità serve a tre scopi.
  - In primo luogo, i modelli più semplici sono più facili da distribuire e la distribuzione precoce del modello consente di convalidare che la pipeline di predizione sia coerente con la pipeline di addestramento
  - In secondo luogo, iniziare con qualcosa di semplice e aggiungere componenti più complessi passo dopo passo rende più facile la comprensione del modello e il suo debug
  - In terzo luogo, il modello più semplice serve da riferimento per confrontare i modelli più complessi

Sei consigli per la selezione dei modelli

- **Evitare i pregiudizi:** Immaginate che a un ingegnere del vostro team venga assegnato il compito di valutare quale modello sia migliore per il vostro task: un decision tree o una rete neurale. Dopo due settimane, questo ingegnere annuncia che la rete neurale ha performance migliori rispetto al decision tree. Il vostro team decide di utilizzare la rete neurale
- Qualche mese dopo, però, un ingegnere esperto si unisce al vostro team. Decide di esaminare nuovamente le performance e scopre che il decision tree supera la rete neurale. Che cosa è successo?

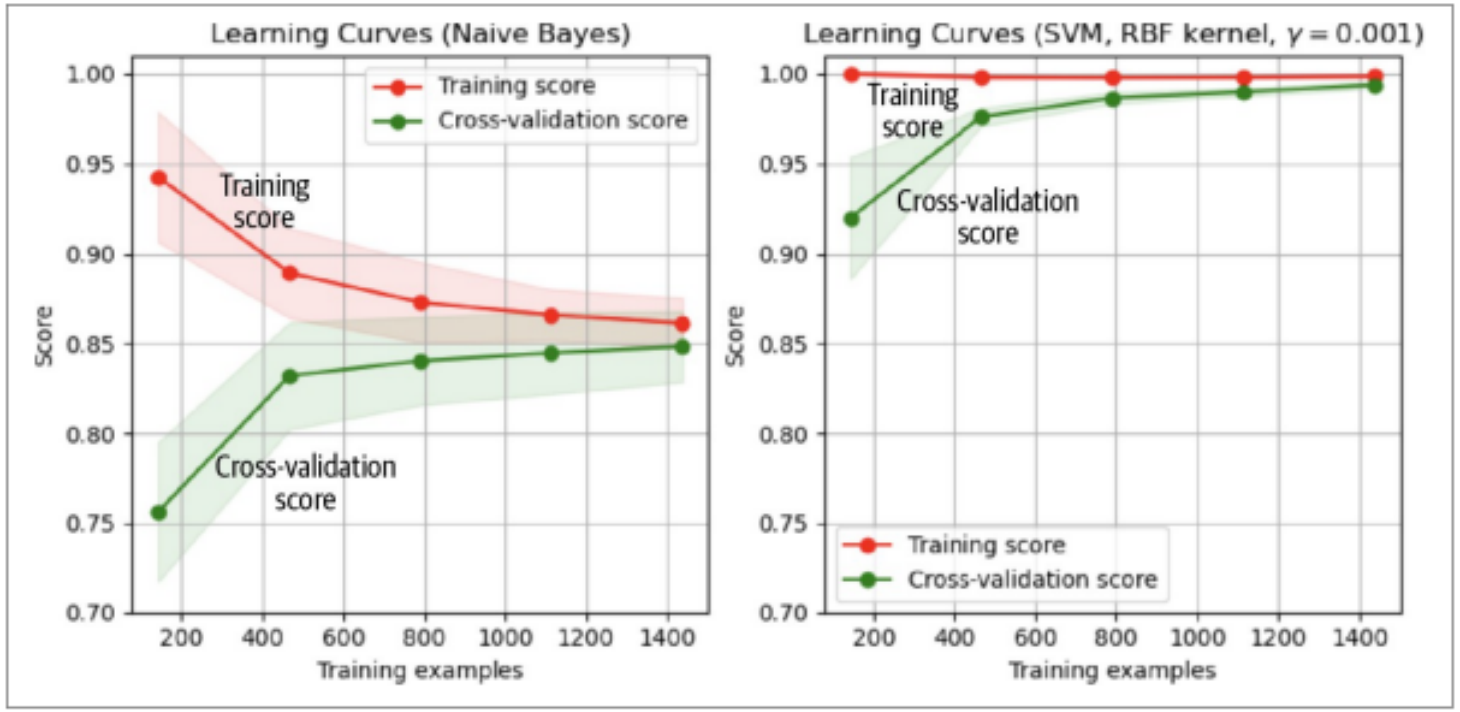
Sei consigli per la selezione dei modelli

- Ci sono molti pregiudizi umani nella valutazione dei modelli. Parte del processo di valutazione di un'architettura di ML consiste nello sperimentare diverse feature e diversi set di iperparametri per trovare il modello migliore per tale architettura. Se una persona è più entusiasta di un'architettura, probabilmente passerà molto più tempo a sperimentarla, il che potrebbe portare a modelli più performanti per quell'architettura
- Quando si confrontano architetture diverse, è importante confrontarle con configurazioni simili. Se si eseguono 100 esperimenti per un'architettura, non è corretto eseguire solo un paio di esperimenti per l'architettura che si sta valutando. Potrebbe essere necessario eseguire 100 esperimenti anche per l'altra architettura
- Poiché le prestazioni di un'architettura dipendono fortemente dal contesto in cui viene valutata (ad esempio, il task, i dati di addestramento, i dati di test, gli iperparametri, ecc.), l'affermazione potrebbe essere vera in un contesto, ma improbabile per tutti i contesti possibili

Sei consigli per la selezione dei modelli

- **Valutare le performance attuali rispetto a quelle future:** Il modello migliore ora non è detto che sia il modello migliore tra due mesi. Ad esempio, un modello ad albero potrebbe funzionare meglio ora perché non avete una quantità elevata di dati di addestramento, ma tra due mesi potreste essere in grado di raddoppiare la quantità di dati e una rete neurale potrebbe funzionare molto meglio
  - Un modo semplice per stimare come le performance di un modello variano è quello di utilizzare le curve di apprendimento

Sei consigli per la selezione dei modelli



- Una curva di apprendimento illustra graficamente come variano le performance di un modello all'aumentare del numero di campioni di addestramento utilizzati

Sei consigli per la selezione dei modelli

- **Valutare i compromessi:** Quando si scelgono i modelli, si devono fare molti compromessi. Capire cosa è più importante per le performance del vostro sistema di ML vi aiuterà a scegliere il modello più adatto. Esempi classici di compromessi sono:
  - Falsi positivi e falsi negativi: Ridurre il numero di falsi positivi potrebbe aumentare il numero di falsi negativi e viceversa. In un task in cui i falsi positivi sono più pericolosi dei falsi negativi, ad esempio l'accesso a strutture tramite impronte digitali (le persone non autorizzate non dovrebbero essere classificate come autorizzate e avere accesso), si potrebbe preferire un modello che produce meno falsi positivi
  - Requisiti di calcolo e accuratezza: Un modello più complesso potrebbe offrire un'accuratezza più elevata, ma potrebbe richiedere una macchina più potente per generare previsioni con una latenza di inferenza accettabile
  - Interpretabilità e performance: Un modello più complesso può fornire performance migliori, ma i suoi risultati sono meno interpretabili

Sei consigli per la selezione dei modelli

- **Comprendere le ipotesi di un modello:** Il mondo reale è estremamente complesso e i modelli possono solo approssimarlo utilizzando delle ipotesi. Ogni singolo modello ha le sue ipotesi. Capire quali sono le ipotesi di un modello e se i vostri dati le soddisfano può aiutarvi a valutare quale sia il modello migliore per il vostro caso d'uso. Ipotesi comuni sono
  - **IID:** Le reti neurali presuppongono che gli esempi siano indipendenti e identicamente distribuiti, il che significa che tutti gli esempi sono tratti in modo indipendente dalla stessa distribuzione congiunta
  - **Confini:** Un classificatore lineare presuppone che i confini decisionali siano lineari

Ensemble

- Quando si considera una soluzione di ML per il proprio problema, si potrebbe iniziare con un sistema che contiene un solo modello
- Dopo aver sviluppato un singolo modello, si potrebbe pensare a come migliorare le performance del sistema
- Un possibile approccio è l'utilizzo di un ensemble di più modelli per fare previsioni
- Ogni modello dell'ensemble è chiamato 'base learner'
- Ad esempio, per prevedere se un'e-mail è SPAM o NON SPAM, si potrebbero avere tre modelli diversi. La previsione finale per ogni e-mail si potrebbe ricavare tramite un voto a maggioranza dei tre modelli
- I metodi di ensembling sono meno favoriti in produzione perché sono più complessi da distribuire e più difficili da manutenere. Tuttavia, sono ancora comuni per le attività in cui un piccolo aumento delle prestazioni può portare un enorme guadagno economico, come la previsione del tasso di clic per gli annunci pubblicitari

Ensemble - Esempio

- Immaginiamo di avere tre classificatori di spam via e-mail, ciascuno con un'accuratezza del 70%. Supponendo che ogni classificatore abbia la stessa probabilità di fare una previsione corretta e che i classificatori non siano correlati, tramite l'utilizzo del voto a maggioranza possiamo ottenere un'accuratezza del 78,4%
- Per ogni e-mail, ogni classificatore ha il 70% di probabilità di essere corretto. L'ensemble sarà corretto se almeno due classificatori sono corretti. La seguente tabella mostra le probabilità dei diversi risultati possibili dell'ensemble per un'e-mail

Outputs of three models	Probability	Ensemble's output
All three are correct	0.7 * 0.7 * 0.7 = 0.343	Correct
Only two are correct	(0.7 * 0.7 * 0.3) * 3 = 0.441	Correct
Only one is correct	(0.3 * 0.3 * 0.7) * 3 = 0.189	Wrong
None are correct	0.3 * 0.3 * 0.3 = 0.027	Wrong

- Pertanto, l'ensemble avrà un'accuratezza di 0,343 + 0,441 = 0,784, ovvero del 78,4%

Ensemble - Esempio

- Tuttavia, il precedente calcolo è valido solo se i classificatori di un ensemble non sono correlati
- Se tutti i classificatori sono perfettamente correlati – tutti e tre calcolano la stessa previsione per ogni e-mail – l'ensemble avrà la stessa accuratezza di ogni singolo classificatore
- Quando si crea un ensemble, minore è la correlazione tra i base learner, migliore sarà l'ensemble
- Per questo motivo, è comune scegliere tipi di modelli molto diversi per un ensemble

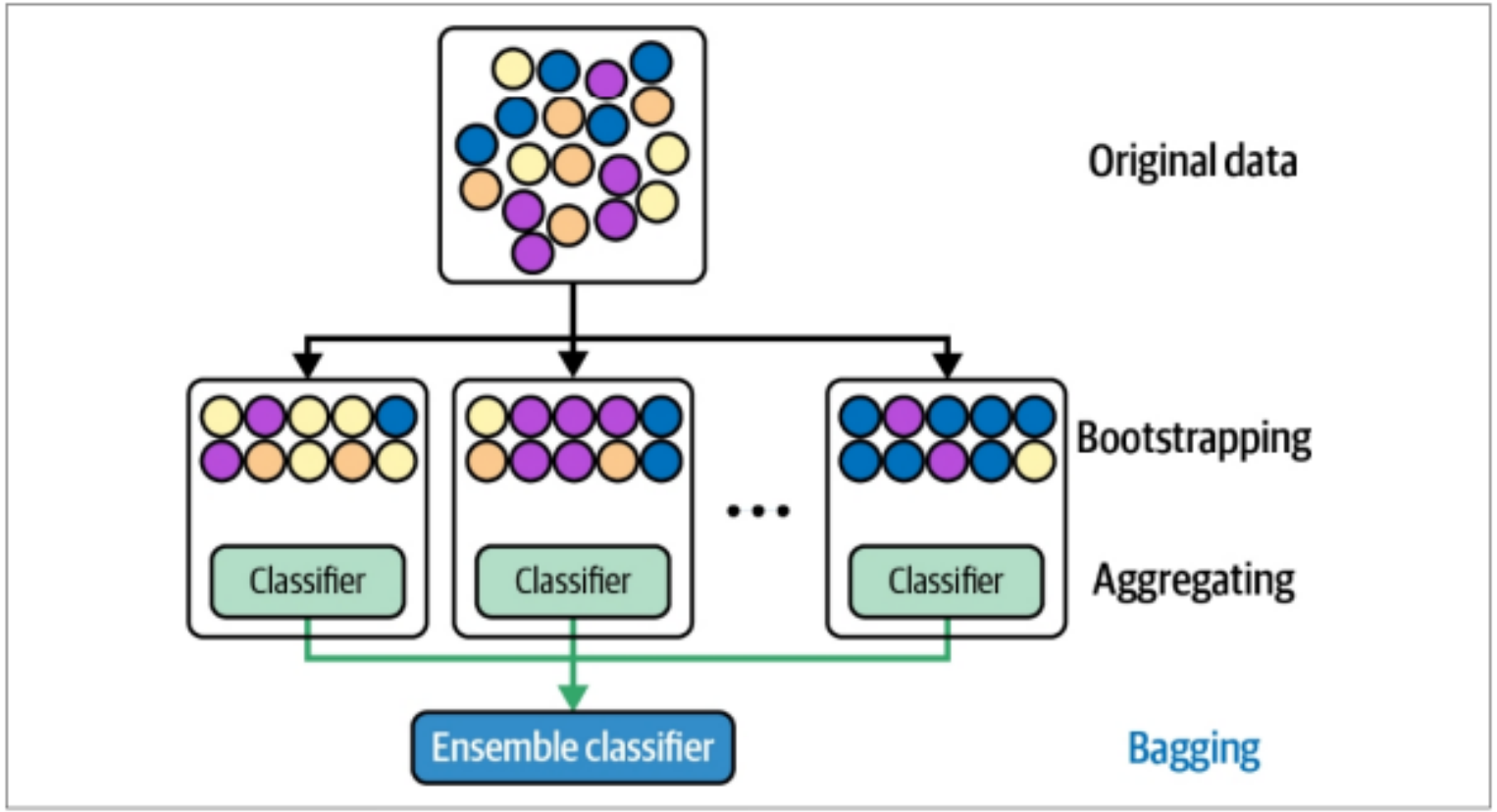
Ensemble

- Esistono tre modi per creare un ensemble
  - **Bagging**
  - **Boosting**
  - **Stacking**
- Oltre a contribuire ad aumentare le performance, secondo diversi studi, i metodi di ensemble come il boosting e il bagging, insieme al ricampionamento, hanno dimostrato di essere utili in caso di insiemi di dati sbilanciati

Bagging

- Bagging, abbreviazione di bootstrap aggregating, è stato progettato per migliorare la stabilità dell'addestramento e l'accuratezza degli algoritmi di ML
- Esso permette di ridurre la varianza e aiuta ad evitare l'overfitting
- Dato un set di dati, invece di addestrare un classificatore sull'intero set di dati, si campiona con sostituzione per creare diversi set di dati, chiamati bootstrap
- Il campionamento con sostituzione garantisce che ogni boostrap sia creato indipendentemente

Bagging


Original data
Classifier
Classifier
Classifier
Bootstrapping
Aggregating
Ensemble classifier
Bagging

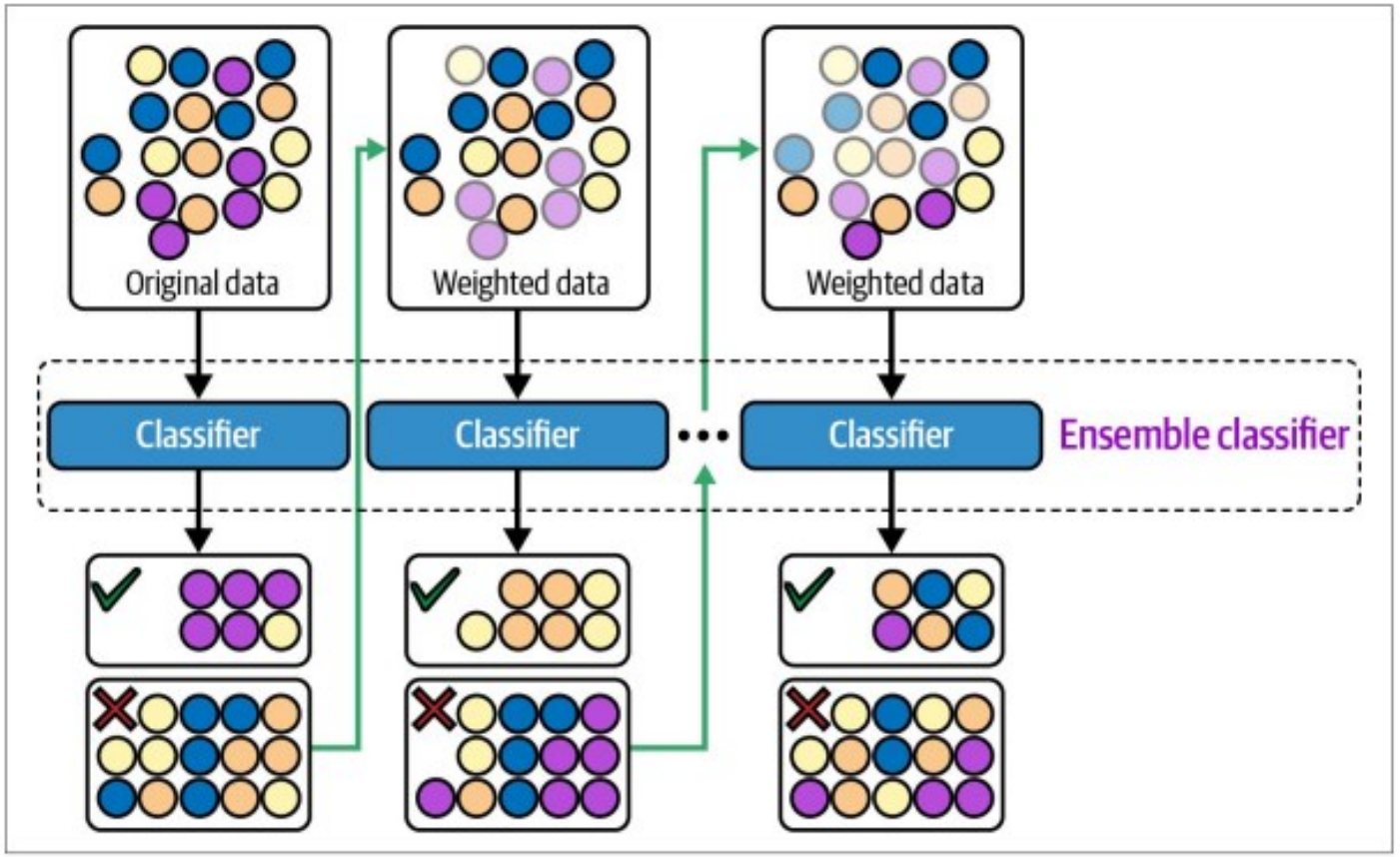
Bagging

- Il bagging migliora generalmente i metodi instabili, come le reti neurali, i classification e regression tree, e la selezione di sottoinsiemi nella regressione lineare
- Tuttavia, può degradare leggermente le performance di metodi stabili come il KNN
- Un random forest è un esempio di bagging. In particolare, esso è un ensemble composto da un insieme di decision tree costruiti sia con il bagging che con la tecnica feature randomness, dove ogni albero può scegliere solo da un sottoinsieme casuale di feature da utilizzare

Boosting

- Il boosting è una famiglia di algoritmi di ensemble iterativi che convertono i learner deboli in learner forti
- Ogni learner di questo ensemble è addestrato sullo stesso insieme di campioni, ma essi sono ponderati in modo diverso nelle iterazioni
- Di conseguenza, i futuri learner deboli si concentrano maggiormente sugli esempi che i precedenti learner deboli hanno classificato in modo errato

Boosting


Original data
Weighted data
Weighted data
Classifier
Classifier
Classifier
Ensemble classifier

Boosting

- Il boosting prevede le seguenti fasi:
  1. Si inizia addestrando il primo classificatore debole sul set di dati originale
  2. I campioni vengono pesati in base al grado di classificazione del primo classificatore, ad esempio, ai campioni mal classificati viene attribuito un peso maggiore
  3. Viene addestrato il secondo classificatore sul set di dati pesato. L'ensemble è ora composto dal primo e dal secondo classificatore
  4. I campioni sono pesati in base all'accuratezza con cui l'ensemble riesce a classificarli
  5. Viene addestrato il terzo classificatore sul set di dati pesato e viene aggiunto all'ensemble
  6. Tali step vengono ripetuti per un certo numero di iterazioni
  7. Si genera un classificatore forte come combinazione pesata dei classificatori esistenti: i classificatori con errori di addestramento minori hanno pesi maggiori

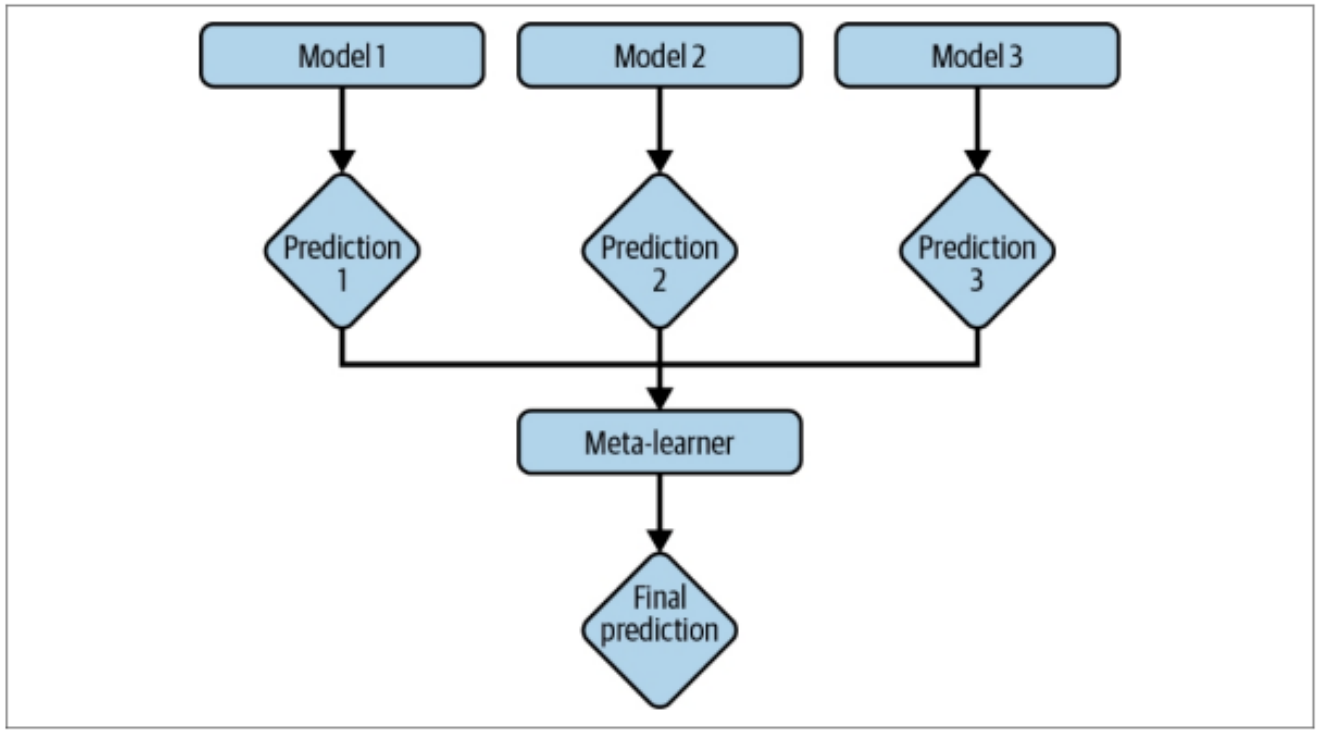
Boosting

- Un esempio di algoritmo di boosting è il Gradient Boosting Machine (GBM), il quale produce un modello di previsione a partire da decision tree deboli
- Costruisce il modello in modo graduale, come fanno altri metodi di boosting, e li generalizza consentendo l'ottimizzazione di una funzione di perdita differenziabile arbitraria

Stacking

- Lo stacking consiste nell'addestrare dei base learner sui dati di addestramento e poi creare un meta-learner che combina i risultati dei base learner per produrre le previsioni finali
- Il meta-learner può essere molto semplice: si prende il voto della maggioranza (per i task di classificazione) o il voto medio (per i task di regressione) di tutti i base learner
- Oppure può essere un altro modello, come un modello di regressione logistica o un modello di regressione lineare, addestrato sulle predizioni dei base learner

Stacking


Model 1
Model 2
Model 3
Prediction 1
Prediction 2
Prediction 3
Meta-learner
Final prediction

Experiment Tracking e Versioning

- Durante il processo di sviluppo di un modello, è spesso necessario sperimentare molte architetture e molti modelli diversi per scegliere il migliore per il problema
- Alcuni modelli potrebbero sembrare simili e differire solo per un iperparametro, ottenendo però performance drammaticamente diverse
- È importante tenere traccia di tutte le informazioni necessarie per ricreare un esperimento e i relativi artefatti
  - Un artefatto è un file generato durante un esperimento: ad esempio, i file che mostrano la curva di perdita o i risultati intermedi di un modello durante il processo di addestramento
- Il confronto di diversi esperimenti può aiutare a capire come piccole modifiche influenzano le performance di un modello, il che, a sua volta, offre una maggiore visibilità sul funzionamento dello stesso

Experiment Tracking e Versioning

- Il processo di tracciamento dei progressi e dei risultati di un esperimento prende il nome di experiment tracking
- Il processo di memorizzazione di tutti i dettagli di un esperimento allo scopo di riprodurlo in seguito o di confrontarlo con altri esperimenti prende il nome di versioning

Experiment Tracking

- Gran parte dell'addestramento di un modello di ML consiste nell'assistere i processi di apprendimento
- Durante il processo possono verificarsi diverse problematiche, tra cui l'overfitting, underfitting, esaurimento della memoria
- È importante tenere traccia di ciò che accade durante l'addestramento non solo per individuare e risolvere questi problemi, ma anche per valutare se il modello sta imparando qualcosa di utile
- Inoltre, il tracciamento consente il confronto tra gli esperimenti. Infatti, osservando il modo in cui la modifica di un componente influisce sulle performance di un modello, si può capire cosa fa quel componente

Experiment Tracking

- Ecco un elenco di aspetti da tracciare per un esperimento:
  - Le metriche di prestazione del modello che vi interessano sui set di test e validazione, come accuratezza, F1, perplessità
  - Il log del campione, della previsione e dell'etichetta di verità corrispondente. È utile per le analisi ad hoc e per il controllo della correttezza
  - La velocità del modello, valutata dal numero di step al secondo o, se i dati sono testuali, dal numero di token elaborati al secondo
  - Misurazioni delle prestazioni del sistema, come l'utilizzo della memoria e della CPU/GPU. Sono importanti per identificare i colli di bottiglia ed evitare di sprecare le risorse del sistema
  - I valori nel tempo di tutti i parametri e iperparametri le cui variazioni possono influenzare le performance del modello, come ad esempio il learning rate

Versioning

- I sistemi di ML sono in parte codice e in parte dati, quindi non è sufficiente effettuare il versioning solo del codice, ma bisogna farlo anche dei dati
- Ci sono alcuni motivi per cui il versioning dei dati è una sfida. Uno di questi è che, poiché i dati sono spesso molto più grandi del codice, non possiamo usare la stessa strategia che di solito si usa con il codice
- Ad esempio, il versioning del codice viene effettuato tenendo traccia di tutte le modifiche apportate a una base di codice
- Una modifica è nota come diff, abbreviazione di difference
- Ogni modifica viene misurata con un confronto per righe

Versioning

- Una riga di codice è di solito abbastanza corta affinché il confronto per righe abbia senso
- Tuttavia, una riga di dati, soprattutto se memorizzati in formato binario, può essere indefinitamente lunga
- Gli strumenti di versioning del codice consentono agli utenti di tornare a una versione precedente mantenendo le copie di tutti i vecchi file
- Tuttavia, un set di dati potrebbe essere così grande che duplicarlo più volte potrebbe non essere fattibile
- Gli strumenti di versioning del codice consentono a più persone di lavorare contemporaneamente sullo stesso codice, duplicandolo sul computer locale di ciascuno
- Tuttavia, un set di dati potrebbe essere così grande che duplicarlo più volte potrebbe non essere fattibile

Versioning

- In secondo luogo, c'è ancora confusione su cosa costituisca esattamente un diff quando si effettua il versioning dei dati. I diff significano modifiche al contenuto di qualsiasi file nel repository? Solo quando un file viene rimosso o aggiunto? O quando il checksum dell'intero repository è cambiato?
- A partire dal 2021, gli strumenti di versioning dei dati come DVC registrano una differenza solo se la checksum della directory è cambiata e se un file viene rimosso o aggiunto
- Un'altra confusione riguarda la risoluzione dei conflitti di fusione: se lo sviluppatore 1 usa la versione X dei dati per addestrare il modello A e lo sviluppatore 2 usa la versione Y dei dati per addestrare il modello B, non ha senso fondere le versioni X e Y dei dati per creare Z, dato che non esiste un modello corrispondente a Z

Versioning

- In terzo luogo, se si utilizzano i dati degli utenti per addestrare il modello, le normative come il Regolamento generale sulla protezione dei dati (GDPR) potrebbe rendere complicata la gestione delle versioni dei dati. Ad esempio, le normative potrebbero imporre la cancellazione dei dati degli utenti su richiesta, rendendo legalmente impossibile il recupero delle versioni precedenti dei dati
- L'experiment tracking e il versioning consentono la riproducibilità, ma non la garantiscono
- I framework e l'hardware utilizzati potrebbero introdurre il nondeterminismo nei risultati dell'esperimento, rendendo impossibile la riproduzione del risultato di un esperimento senza conoscere tutto ciò che riguarda l'ambiente in cui l'esperimento viene eseguito

Debug dei modelli di ML

- Il debug è parte integrante dello sviluppo di qualsiasi software. I modelli di ML non fanno eccezione
- Esso può essere particolarmente complesso per tre motivi
  - I modelli di ML falliscono silenziosamente. Il codice viene compilato. La perdita diminuisce come dovrebbe. Vengono invocate le funzioni corrette. Le previsioni vengono fatte, ma sono sbagliate
  - La lentezza con cui si può verificare se un bug è stato risolto. Quando si esegue il debug di un programma software tradizionale, è possibile apportare modifiche al codice e testarlo immediatamente. Tuttavia, quando si apportando modifiche a un modello di ML, è necessario riqualificare il modello e attendere che converga per verificare se il bug è stato risolto, il che può richiedere ore/giorni
  - Complessità interfunzionale. In un sistema di ML ci sono molti componenti (dati, algoritmi, ecc.), pertanto quando si verifica un errore potrebbe essere difficile stabilire chi lo ha causato

Debug dei modelli di ML

- Ecco alcuni degli elementi che possono causare il fallimento di un modello di ML
  - Vincoli teorici. Ogni modello ha le proprie ipotesi sui dati e sulle feature che utilizza. Un modello potrebbe fallire perché i dati da cui apprende non sono conformi alle sue ipotesi. Ad esempio, si utilizza un modello lineare per dati i cui confini decisionali non sono lineari
  - Scelta inadeguata degli iperparametri. Dato un modello, un insieme di iperparametri può fornire un risultato all'avanguardia, ma un altro insieme potrebbe far sì che il modello non converga mai. Il modello si adatta perfettamente ai dati e la sua implementazione è corretta, ma una serie inadeguata di iperparametri potrebbe rendere il modello inutile
  - Problemi di dati. Campioni di dati ed etichette non correttamente accoppiati, etichette rumorose, feature normalizzate usando statistiche obsolete, ecc. potrebbero influenzare negativamente il training di un modello
  - Scarsa scelta di feature. I modelli possono avere molte feature da cui apprendere. Un numero eccessivo di feature potrebbe far sì che un modello si adatti eccessivamente ai dati, mentre un numero insufficiente potrebbe mancare di potere predittivo per consentire a un modello di fare buone previsioni

Debug dei modelli di ML

- Il debug deve essere sia preventivo che curativo. È necessario disporre di pratiche sane per ridurre al minimo le opportunità di proliferazione dei bug e di una procedura per individuarli, localizzarli e risolverli
- Purtroppo non esiste ancora un approccio scientifico al debugging in ML
- Tuttavia, sono state pubblicate diverse tecniche di debug collaudate da ingegneri e ricercatori esperti di ML

Debug dei modelli di ML

- **Iniziare in modo semplice e aggiungere gradualmente altri componenti:** Iniziate con il modello più semplice e poi aggiungente lentamente altri componenti per vedere se aiutano o danneggiano le prestazioni. Ad esempio, se si vuole costruire una rete neurale ricorrente, si inizia con un solo livello prima di sovrapporne più di uno
- **Training su un singolo batch:** Dopo aver ottenuto una semplice implementazione del modello, provate a fare un training su una piccola quantità di dati di addestramento ed eseguite una valutazione sugli stessi dati per assicurarvi che il modello raggiunga la perdita più piccola possibile. Se il modello non è in grado di eseguire il training su una piccola quantità di dati, potrebbe esserci qualcosa di sbagliato nell'implementazione
- **Impostare un seme casuale:** Sono moli i fattori che contribuiscono alla casualità del modello: inizializzazione dei pesi, dropout, ecc. L'aleatorietà rende difficile confrontare i risultati tra diversi esperimenti: non si può sapere se la variazione delle prestazioni è dovuta a una modifica del modello o a un seme casuale diverso. L'impostazione di un seme casuale garantisce la coerenza tra le diverse esecuzioni

Apprendimento distribuito

- Poiché i modelli sono sempre più grandi e richiedono più risorse, le aziende si preoccupano molto di più dell'addestramento su scala
- Quando i dati non possono essere memorizzati in memoria, gli algoritmi di pre-processing, rimescolamento e batching dei dati devono essere eseguiti fuori dal core e in parallelo

Parallelismo dei dati

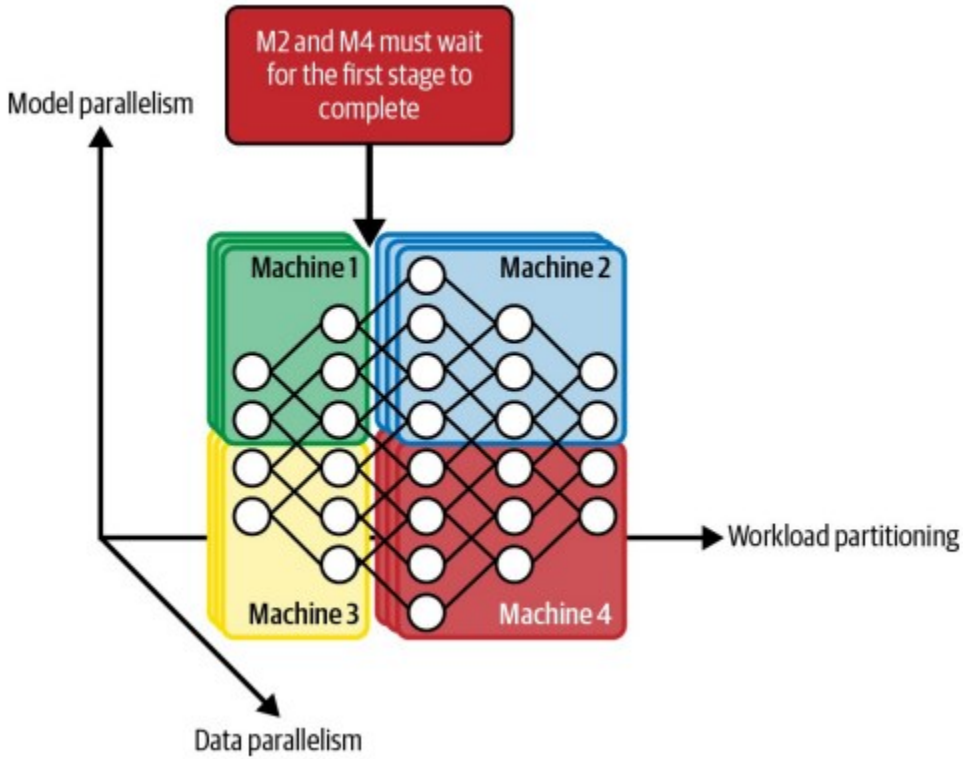
- Il metodo di parallelizzazione più comune supportato dai moderni framework di ML è il parallelismo dei dati: si dividono i dati su più macchine, si addestra il modello su tutte e si accumulano i gradienti
- Il problema significativo è come accumulare in modo accurato ed efficace i gradienti provenienti da macchine diverse
  - Poiché ogni macchina produce il proprio gradiente e bisogna attendere che tutte abbiano terminato l'esecuzione, i ritardi causano il rallentamento dell'intero sistema, con conseguente spreco di tempo e risorse
  - Se il modello aggiorna il peso utilizzando il gradiente di ogni macchina separatamente, lo stallo del gradiente potrebbe diventare un problema perché i gradienti di una macchina hanno fatto cambiare i pesi prima che arrivassero i gradienti di un'altra macchina
- Tuttavia, esistono molti algoritmi che affrontano efficacemente questo problema

Parallelismo del modello

- Con il parallelismo dei dati, ogni worker ha la propria copia dell'intero modello ed esegue tutti i calcoli necessari per la propria copia del modello
- Il parallelismo del modello si ha quando i diversi componenti del modello vengono addestrati su macchine diverse

Parallelismo del modello

- Ad esempio, la macchina 0 gestisce il calcolo dei primi due livelli mentre la macchina 1 gestisce i due livelli successivi, oppure alcune macchine possono gestire il forward pass mentre altre gestiscono il backward pass



Parallelismo del modello

- Il parallelismo del modello può essere fuorviante perché in alcuni casi il parallelismo non significa che le diverse parti del modello su macchine diverse vengono eseguite in parallelo
- Ad esempio, se il modello corrisponde ad una matrice molto grande divisa a metà su due macchine, queste due metà potrebbero essere elaborate in parallelo
- Tuttavia, se il modello è una rete neurale e il primo layer è collocato sulla macchina 1 e il secondo layer sulla macchina 2, e il layer 2 ha bisogno degli output del layer 1 per essere elaborato, la macchina 2 deve aspettare che la macchina 1 termini l'esecuzione

Parallelismo della pipeline

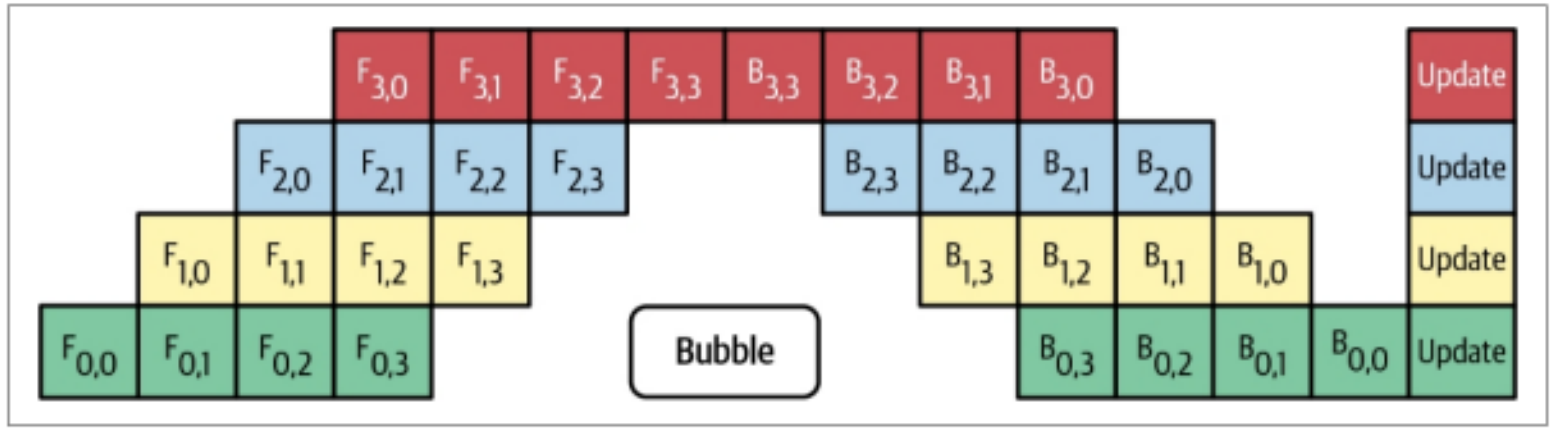
- Il parallelismo della pipeline è una tecnica intelligente per far funzionare in parallelo i diversi componenti di un modello su macchine diverse
- Esistono diverse varianti, ma l'idea chiave è quella di suddividere il calcolo di ogni macchina in più parti
- Quando la macchina 1 termina la prima parte del calcolo, passa il risultato alla macchina 2, che prosegue con la seconda parte e così via
- La macchina 2 può ora eseguire il suo calcolo sulla prima parte mentre la macchina 1 esegue il suo calcolo sulla seconda parte

Parallelismo della pipeline - Esempio

- Per concretizzare questo concetto, si consideri di avere quattro macchine diverse e che il primo, il secondo, il terzo e il quarto livello si trovino rispettivamente sulla macchina 1, 2, 3 e 4
- Con il parallelismo della pipeline, ogni mini-batch viene suddiviso in quattro micro-batch
- La macchina 1 calcola il primo livello sul primo micro-lotto, quindi la macchina 2 calcola il secondo livello sui risultati della macchina 1, mentre la macchina 1 calcola il primo livello sul secondo micro-lotto e così via

Parallelismo della pipeline - Esempio

- La seguente figura mostra l'aspetto del parallelismo della pipeline su quattro macchine; ogni macchina esegue sia il forward pass che il backward pass per un componente della rete neurale



Apprendimento distribuito

- Il parallelismo del modello e il parallelismo dei dati non si escludono a vicenda
- Molte aziende utilizzano entrambi i metodi per un migliore utilizzo dell'hardware, anche se la configurazione per utilizzare entrambi i metodi può richiedere un notevole sforzo di progettazione

AutoML

- AutoML è stato introdotto da Jeff Dean al TensorFlow Dev Summit nel 2018 e si riferisce all'automatizzazione del processo di ricerca di algoritmi di ML per risolvere problemi del mondo reale
- L'idea alla base è: Invece di pagare un gruppo di ricercatori/ingegneri di ML per 'giocare' con vari modelli e alla fine selezionarne uno non ottimale, perché non usare quei soldi per risorse computazionali necessarie per cercare il modello ottimale?

Soft AutoML: Regolazione degli iperparametri

- Una forma leggera, e la più popolare, di AutoML in produzione è la regolazione degli iperparametri
- Un iperparametro è un parametro fornito dagli utenti il cui valore viene utilizzato per controllare il processo di apprendimento, ad esempio il learning rate, la batch size, il numero di hidden layer

Soft AutoML: Regolazione degli iperparametri

- Con diversi set di iperparametri, lo stesso modello può fornire performance drasticamente diverse sullo stesso set di dati
- Melis et al. hanno dimostrato nel loro articolo 'On the State of the Art of Evaluation in Neural Language Models' che i modelli più 'deboli' con iperparametri ben tarati possono superare i modelli più 'forti'
- L'obiettivo della regolazione degli iperparametri è trovare l'insieme ottimale di iperparametri per un dato modello all'interno di uno spazio di ricerca

Soft AutoML: Regolazione degli iperparametri

- Nonostante ne conoscano l'importanza, molti continuano a ignorare gli approcci sistematici alla regolazione degli iperparametri a favore di un approccio manuale, basato sull'istinto
- Il più popolare è probabilmente il Graduate Student Descent (GSD), una tecnica in cui uno studente laureato giocherella con gli iperparametri finché il modello non funziona
- Tuttavia, sempre più persone adottano la regolazione degli iperparametri come parte delle loro pipeline

Soft AutoML: Regolazione degli iperparametri

- I framework di ML più noti sono dotati di utilità integrate o di utilità di terze parti per la regolazione degli iperparametri, ad esempio Scikit-Learn con Auto-Sklearn, TensorFlow con Keras Tuner e Ray con Tune
- Altri metodi noti sono la Random Search, Grid Search e Bayesian Optimization
- Quando si regolano gli iperparametri, bisogna tenere presente che le performance di un modello potrebbero essere più sensibili alla variazione di un iperparametro rispetto a un altro, e quindi gli iperparametri sensibili dovrebbero essere regolati con maggiore attenzione

Hard AutoML: Ricerca dell'architettura

- Alcuni team portano la regolazione degli iperparametri a un livello superiore: se trattassimo altri componenti di un modello o l'intero modello come iperparametri?
- La dimensione di un layer convoluzionale o la presenza di uno skip layer possono essere considerati iperparametri
- Invece di inserire manualmente un layer di pooling dopo un layer convoluzionale o ReLu dopo uno lineare, si forniscono all'algoritmo questi elementi costitutivi e si lascia che sia lui a capire come combinarli
- Quest'area di ricerca è nota come Architectural Search, o Neural Architecture Search (NAS) per le reti neurali, in quanto cerca l'architettura ottimale del modello

Hard AutoML: Ricerca dell'architettura

- Una configurazione NAS è costituita da tre componenti:
  - Uno spazio di ricerca. Definisce le possibili architetture del modello, cioè gli elementi costitutivi tra cui scegliere e i vincoli su come possono essere combinati
  - Una strategia di stima delle performance. Valuta le performance di un'architettura candidata senza dover addestrare ogni architettura candidata da zero fino alla convergenza
  - Una strategia di ricerca. Per esplorare lo spazio di ricerca. Un approccio semplice è la ricerca casuale – scegliere casualmente tra tutte le configurazioni possibili – che è impopolare perché è probabilmente costosa anche per i NAS
    - Gli approcci più comuni includono l'apprendimento per rinforzo e l'evoluzione

Hard AutoML: Ricerca dell'architettura

- Per i NAS lo spazio di ricerca è discreto: l'architettura finale utilizza solo una delle opzioni disponibili e l'utente deve fornire l'insieme dei blocchi costruttivi (ad esempio, convoluzioni di diverse dimensioni, varie attivazioni, pooling, ecc.)
- L'insieme dei blocchi costruttivi varia in base all'architettura di base, ad esempio reti neurali convoluzionali o transformer

Hard AutoML: Ricerca dell'architettura

- In un tipico processo di addestramento di ML, si ha un modello, una procedura di apprendimento, un algoritmo che aiuta il modello a trovare l'insieme di parametri che minimizzano una data funzione obiettivo per un dato insieme di dati
- La procedura di apprendimento più comune per le reti neurali è la Discesa del Gradiente, che sfrutta un ottimizzatore per specificare come aggiornare i pesi del modello in base agli aggiornamenti del gradiente
  - Gli ottimizzatori più diffusi sono: Adam, Momentum, SGD
- In teoria, è possibile includere gli ottimizzatori come blocchi costruttivi in NAS e cercare quello che funziona meglio
- In pratica, questo è difficile da fare, poiché gli ottimizzatori sono sensibili alle impostazioni e gli iperparametri predefiniti spesso non funzionano bene su tutte le architetture

Hard AutoML: Ricerca dell'architettura

- Questo porta a una direzione di ricerca interessante: cosa succede se sostituiamo le funzioni che specificano la regola di aggiornamento con una rete neurale?
- La quantità di aggiornamento dei pesi del modello sarà calcolata da questa rete neurale. Questo approccio consente di ottenere learned optimizer, invece di adottare ottimizzatori progetti 'a mano'
- Poiché i learned optimizer sono reti neurali, devono essere addestrati
- È possibile addestrare un learned optimizer sullo stesso set di dati su cui si addestra il resto della rete neurale, ma questo richiede l'addestramento di un ottimizzatore ogni volta che bisogna risolvere un task

Hard AutoML: Ricerca dell'architettura

- Un altro approccio consiste nell'addestrare una sola volta un learned optimizer su un insieme di task e utilizzarlo per ogni task successivo
- Bisogna utilizzare una perdita aggregata sull'insieme dei task come funzione di perdita e gli ottimizzatori esistenti come regola di apprendimento

Hard AutoML: Ricerca dell'architettura

- Che si tratti di ricerca di architetture o di regole di apprendimento, il costo iniziale dei training è abbastanza costoso e solo poche aziende al mondo possono permettersi di perseguirle
- Tuttavia, è importante che le persone interessate al ML siano a conoscenza dei progressi di AutoML per due motivi
  - In primo luogo, le architetture e i learned optimizer che ne derivano possono consentire agli algoritmi di ML di lavorare in modo autonomo su più task del mondo reale, risparmiando tempo e costi di produzione, sia in fase di addestramento che di inferenza
  - In secondo luogo, potrebbero essere in grado di risolvere molti task del mondo reale precedentemente impossibili con le architetture e gli ottimizzatori esistenti

Model Offline Evaluation

3

Valutazione offline di un modello

- Idealmente, i metodi di valutazione dovrebbero essere gli stessi sia in fase di sviluppo che di produzione
- In molti casi, però, l'ideale è impossibile, perché durante lo sviluppo si dispone di etichette di verità, ma in produzione no
- Per alcuni task, è possibile dedurre o approssimare le etichette in produzione in base al feedback degli utenti
- Ad esempio, per un task di raccomandazione, è possible dedurre se una raccomandazione è buona dal fatto che gli utenti la considerano
- Tuttavia, ci sono molte distorsioni associate a questo metodo
- Per altri task, potreste non essere in grado di valutare direttamente le performance di un modello in produzione e dovrete affidarvi a un monitoraggio approfondito per rilevare i cambiamenti e i guasti nelle performance del vostro sistema

Baseline

- Le metriche di valutazione, da sole, hanno poco significato
- Quando si valuta un modello, è essenziale conoscere la base di riferimento rispetto alla quale lo si sta valutando
- Le baseline dovrebbero variare da un caso d'uso all'altro. Tuttavia, in seguito vengono discusse cinque baseline che potrebbero essere utili in tutti i casi d'uso

Baseline

- Random baseline. Le previsioni sono generate a caso seguendo una distribuzione specifica, che può essere la distribuzione uniforme o la distribuzione delle etichette del task
- Euristica semplice. Dimenticate il ML. Tale soluzione si limita a fare previsioni basate su semplici euristiche
- Zero rule baseline. Questo è un caso speciale di euristica semplice in cui il modello predice sempre la classe più comune
- Human baseline. In molti casi, l'obiettivo del ML è automatizzare ciò che altrimenti sarebbe stato fatto dall'uomo, quindi è utile sapere come si comporta il vostro modello rispetto agli esperti umani
- Soluzioni esistenti. In molti casi, i sistemi di ML sono progettati per sostituire le soluzioni esistenti. È fondamentale confrontare il nuovo modello con queste soluzioni. Il vostro modello non deve sempre essere migliore delle soluzioni esistenti per essere utile. Un modello le cui performance sono leggermente inferiori può comunque essere utile se è molto più semplice o più economico da usare

Metodi di valutazione

- In ambito accademico, quando si valutano i modelli di ML, si tende a concentrarsi sulle metriche delle performance
- Tuttavia, in produzione, vogliamo anche che i nostri modelli siano robusti, corretti, calibrati e, in generale, che abbiano senso

Test di perturbazione

- Idealmente, gli input utilizzati per sviluppare un modello dovrebbero essere simili a quelli con cui il modello dovrà lavorare in produzione, ma in molti casi non è possibile
- Questo è particolarmente vero quando la raccolta dei dati è costosa o difficile e i migliori dati disponibili per l'addestramento sono ancora molto diversi dai dati reali
- Gli input con cui i modelli devono lavorare in produzione sono spesso rumorosi rispetto a quelli utilizzati durante lo sviluppo
- Il modello che funziona meglio sui dati di addestramento non è necessariamente il modello che funziona meglio sui dati che presentano rumore

Test di perturbazione

- Per avere un'idea di come il modello potrebbe funzionare con dati che presentano rumore, potete apportare piccole modifiche ai vostri split di test per vedere come queste modifiche influenzano le performance del modello
- Pertanto, si potrebbe scegliere il modello che funziona meglio sui dati perturbati invece di quello che funziona meglio sui dati chiari
- Più il modello è sensibile al rumore, più sarà difficile manutenerlo, poiché se i comportamenti degli utenti cambiano di poco le performance del modello potrebbero degradarsi
- Inoltre, rende il modello suscettibile ad attacchi avversari

Test di invarianza

- Alcune modifiche agli input non dovrebbero portare a cambiamenti nell'output
- Se ciò accade, il modello presenta delle distorsioni che potrebbero renderlo inutilizzabile, indipendentemente dalle sue performance
- Per evitare queste distorsioni, una soluzione consiste nel mantenere gli input invariati ma cambiare le informazioni sensibili per vedere se i risultati cambiano
- Oppure si dovrebbero escludere le informazioni sensibili dalle feature utilizzate per addestrare il modello

Test di aspettativa direzionale

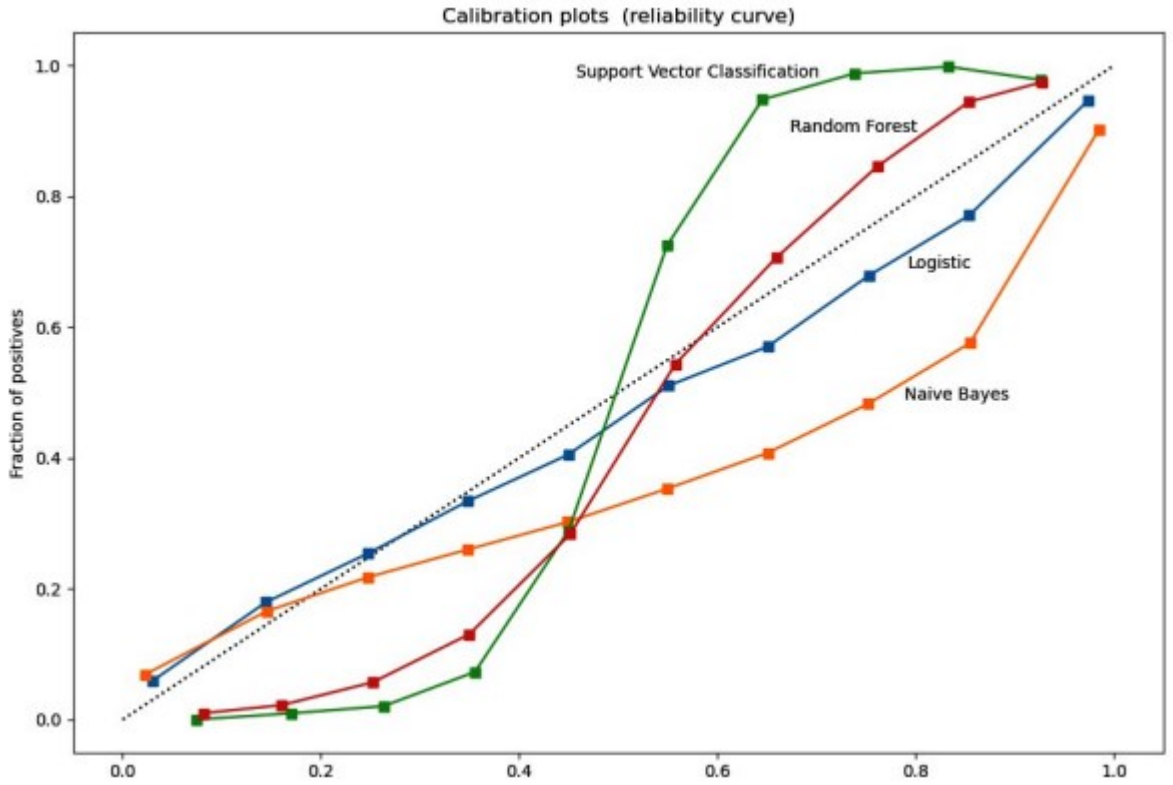
- Alcune modifiche agli input, tuttavia, dovrebbero causare cambiamenti prevedibili negli output
- Ad esempio, quando si sviluppa un modello per prevedere i prezzi delle abitazioni, mantenendo tutte le feature invariate ma aumentando le dimensioni del lotto non dovrebbe diminuire il prezzo previsto, mentre diminuendo la metratura non dovrebbe aumentarlo
- Se i risultati cambiano nella direzione opposta a quella prevista, è possibile che il modello non stia imparando la cosa giusta e che sia necessario indagare ulteriormente prima di distribuirlo

Calibrazione del modello

- La calibrazione del modello è un concetto sottile ma cruciale da comprendere
- Immaginiamo che qualcuno faccia una previsione che qualcosa accadrà con una probabilità del 70%. Questa previsione significa che, tra tutte le volte che viene fatta, il risultato previsto corrisponde a quello effettivo il 70% delle volte
- Se un modello prevede che la squadra A batterà la squadra B con una probabilità del 70% e su 1000 volte che queste due squadre giocano contro, la squadra A vince solo il 60% delle volte, allora diciamo che questo modello non è calibrato
- Un modello calibrato dovrebbe prevedere che la squadra A vinca con una probabilità del 60%

Calibrazione del modello

- Per misurare la calibrazione di un modello, un metodo semplice è il conteggio
- Si conta il numero di volte in cui il modello emette la probabilità *X* e la frequenza *Y* di avveramento della previsione e si traccia il grafico di *X* rispetto a *Y*
- Il grafico di un modello perfettamente calibrato avrà X uguale a Y in tutti i punti dati



Misura della confidenza

- La misura della confidenza può essere vista come un modo per considerare la soglia di utilità di ogni singola previsione
- Mostrare indiscriminatamente agli utenti tutte le previsioni di un modello, anche quelle su cui il modello non è sicuro, può, nel migliore dei casi, causare fastidio e far perdere agli utenti la fiducia nel sistema, come nel caso di un sistema di rilevamento dell'attività sullo smartwatch che pensa che si stia correndo anche se si sta solo camminando un po' velocemente
- Nel peggiore dei casi, può causare conseguenze catastrofiche, come nel caso di un algoritmo che segnala alla polizia una persona innocente come potenziale criminale

Misura della confidenza

- Se si vogliono mostrare solo le previsioni di cui il modello è certo, come si misura questa certezza? Quale è la soglia di certezza a cui mostrare le previsioni? Cosa fare con le previsioni al di sotto di tale soglia: scartarle o chiedere ulteriori informazioni agli utenti?
- Mentre la maggior parte delle altre metriche misura le performance del sistema in media, la misurazione della confidenza è una metrica applicata ad ogni singolo campione
- La misurazione a livello di sistema è utile per avere un'idea delle performance complessive, ma le metriche a livello di campione sono fondamentali quando ci si preoccupa delle performance del sistema su ogni singolo campione

Valutazione Slice-based

- In tale valutazione i dati vengono separati in sottoinsiemi e si esaminano le performance del modello su ciascun sottoinsieme separatamente
- Un errore comune è quello di concentrarsi su metriche a grana grossa, come la F1 complessiva o l'accuratezza sull'insieme dei dati, e non abbastanza su metriche slice-based
- Questo può portare a due problemi

Valutazione Slice-based

- Il primo è che il modello potrebbe comportarsi in modo diverso su diversi sottoinsiemi di dati quando dovrebbe agire allo stesso modo
- Ad esempio, supponiamo di avere due sottogruppi, uno maggioritario e uno minoritario, e il sottogruppo maggioritario rappresenta il 90% dei dati
  - Il modello A raggiunge il 98% di accuratezza sul sottogruppo maggioritario ma solo l'80% sul sottogruppo minoritario, il che significa che la sua accuratezza complessiva è del 96,2%
  - Il modello B raggiunge un'accuratezza del 95% sul gruppo maggioritario e del 95% sul gruppo minoritario, il che significa che la sua accuratezza complessiva è del 95%
- Quale modello sceglieresti?

Valutazione Slice-based

	Majority accuracy	Minority accuracy	Overall accuracy
Model A	98%	80%	96.2%
Model B	95%	95%	95%

- Se un'azienda si concentra solo sulle metriche complessive, potrebbe scegliere il modello A
- Potrebbe essere molto soddisfatta dell'elevata accuratezza di questo modello finché, un giorno, i suoi utenti finali non scoprono che questo modello è prevenuto nei confronti del sottogruppo minoritario che corrisponde a un gruppo demografico sottorappresentato
- Se l'azienda vede le performance dei due modelli in base ai sottogruppi, potrebbe seguire strategie diverse
- Ad esempio, potrebbe decidere di migliorare le performance del modello A sul sottogruppo minoritario, migliorando così le performance complessive del modello
- Oppure potrebbe mantenere entrambi i modelli invariati, ma ora dispone di maggiori informazioni per decidere con maggiore cognizione di causa quale modello impiegare

Valutazione Slice-based

- Il secondo è che il modello potrebbe comportarsi allo stesso modo su diversi sottogruppi di dati, quando invece dovrebbe agire in modo diverso
- Alcuni sottoinsiemi di dati potrebbero essere più critici
- Per esempio, quando si costruisce un modello per la previsione del churn degli utenti (prevedere quando un utente cancellerà un abbonamento o un servizio), gli utenti a pagamento sono più critici di quelli non a pagamento
- Concentrarsi sulle performance complessive di un modello potrebbe danneggiare le sue performance su questi sottogruppi critici

Valutazione Slice-based

- Un motivo affascinante e apparentemente controintuitivo per cui la valutazione slice-based è fondamentale è il paradosso di Simpson, un fenomeno in cui una tendenza appare in diversi gruppi di dati ma scompare o si inverte quando i gruppi vengono combinati
- Ciò significa che il modello B può avere performance migliori del modello A su tutti i dati insieme, ma il modello A ha performance migliori del modello B su ciascun sottogruppo separatamente
- Consideriamo le performance del modello A e del modello B sul gruppo A e sul gruppo B
- Il modello A supera il modello B sia per il gruppo A che per il gruppo B, ma quando vengono combinati, il modello B supera il modello A

	Group A	Group B	Overall
Model A	**93% (81/87)**	**73% (192/263)**	78% (273/350)
Model B	87% (234/270)	69% (55/80)	**83% (289/350)**

Valutazione Slice-based

- Per monitorare le performance del modello sui sottogruppi critici, occorre innanzitutto sapere quali sono i sottogruppi critici nei dati
- Ecco i tre approcci principali:
  - Basato sull'euristica. Suddividere i dati in base alla conoscenza che avete dei dati e del task da svolgere
  - Analisi degli errori. Esaminare manualmente gli esempi classificati erroneamente e trovare pattern significativi
  - Slice finder. Il processo inizia generalmente con la generazione di sottogruppi candidati tramite algoritmi come il beam search o il clustering, e poi procede eliminando i candidati sbagliati nei sottogruppi e riassegnandoli

In conclusione

- In questa lezione abbiamo trattato diversi aspetti legati alla selezione, al tracciamento e alla valutazione dei modelli di ML
  - Esistono diversi aspetti da considerare per prendere una decisione informata su quale modello sia il migliore rispetto agli obiettivi, vincoli e requisiti
  - L'utilizzo di ensemble di modelli può incrementare le performance dei modelli
  - Il tracciamento e il versioning degli esperimenti sono generalmente ritenuti importanti, ma molti ricercatori/ingegneri di ML li ignorano perché potrebbe sembrare un lavoro faticoso
  - L'addestramento distribuito sta diventando una competenza essenziale per gli sviluppatori di modelli di ML e abbiamo discusso le tecniche di parallelismo, tra cui quello dei dati, del modello e della pipeline
  - Esistono diversi metodi di valutazione per scegliere il modello migliore da implementare. Tuttavia, essi non sono molto significativi se non si dispone di baseline con cui confrontare i modelli