Università degli Studi di Milano Bicocca
Corso di Laurea Magistrale in Informatica
2023-2024

Architetture Dati

Use Case MLOps

Feature Store

Quaggio Stefano 866504
Varisco Alberto 866109

Indice

1 Introduzione

Questo progetto è stato svolto per l'esame Architetture Dati ed ha l'obiettivo principale di esplorare e valutare l'efficacia di un concetto innovativo, i Feature Store. All'interno di questa relazione viene, inizialmente, descritto e spiegato in maniera teorica l'argomento in esame e successivamente siamo andati ad implementare un possibile caso d'uso. I capitoli riguardano nel dettaglio:

- **Feature Store**: si inizia con una spiegazione di cosa sono i Feature Store, il loro scopo all'interno di un'architettura software e i vantaggi che possono apportare. Questo include una discussione sui principi fondamentali dei Feature Store, come la gestione centralizzata delle feature engineering e la riutilizzabilità delle feature.
- **Descrizione obiettivi**: viene dettagliato uno specifico caso d'uso, includendo la descrizione dei principali task affrontati e l'identificazione del Feature Store utilizzato.
- **Implementazione**: Descrizione delle tecnologie utilizzate e gli step che ci hanno portato a rendere il Feature Store parte integrante del progetto.
- **Conclusioni**: riflessione sull'utilità dei Feature Store nel caso d'uso specifico esaminato e commenti riguardo ad espansioni future.

1.1 Features

Nel contesto del Machine Learning, le "features" costituiscono le variabili indipendenti che formano la base su cui i modelli fanno affidamento per generare previsioni. Queste informazioni rappresentano gli attributi distintivi di ciascun esempio all'interno dei dati utilizzati. In pratica, i dati di un modello di ML possono essere rappresentati come una tabella, dove ogni riga rappresenta un esempio e ogni colonna rappresenta una specifica caratteristica di tale esempio.

La creazione, la selezione e la gestione efficace di queste caratteristiche sono fondamentali per ottimizzare le prestazioni dei modelli: questo processo è cruciale affinché i modelli possano effettuare previsioni accurate e affidabili, sfruttando al meglio le informazioni disponibili per migliorare la qualità delle analisi e delle previsioni.

2 Feature Store

Negli ultimi anni, con la sempre maggiore complessità dei progetti di machine learning e l'esplosione dei dati disponibili, è diventato fondamentale gestire in modo efficiente e scalabile le caratteristiche utilizzate nei modelli predittivi. I team di data science hanno avuto la

necessità di affrontare tutte le fasi, dalla gestione dell'accesso ai dati grezzi alla costruzione delle features e alla combinazione di queste per il training dei modelli.

Questo ha portato alla diffusione e all'adozione dei Feature Store, repository dedicate alle caratteristiche nel contesto del Machine Learning. Un Feature Store è una piattaforma centralizzata che consente ai data scientist di gestire, condividere e riusare le features in modo efficiente. Garantisce inoltre che il codice utilizzato per calcolare queste caratteristiche sia coerente tra l'addestramento e l'inferenza del modello.

L'introduzione del concetto di Feature Store da parte di Uber nel 2017 ha segnato l'inizio di un'ampia adozione in diversi settori. Negli anni successivi, il numero di utilizzatori è cresciuto rapidamente, trainato anche dall'interesse e dall'entrata sul mercato di grandi aziende come Google, Amazon e Databricks. Questo ha contribuito a un vero e proprio "boom" nell'uso dei Feature Store, evidenziando il loro ruolo cruciale nel supportare l'evoluzione e la scalabilità delle soluzioni di ML.

Figura 1: Feature Store Milestones

I Feature Store sono fondamentali per tre motivi principali::

- **Riuso delle funzionalità:** spesso i team sviluppano modelli che utilizzano le stesse funzionalità, ma le definiscono in modo diverso e le calcolano separatamente. Questo comporta duplicazione degli sforzi e aumento dei costi di calcolo e archiviazione. I Feature Store agiscono come hub per standardizzare le funzioni di machine learning. Le funzionalità sono catalogate, condivise e pre-calcolate all'interno dell'organizzazione,

consentendo di risparmiare sui costi del cloud ed evitando la ricompilazione delle stesse funzioni.

- **Definizioni standardizzate delle funzionalità:** in linea con il punto precedente, quando un team definisce un set di caratteristiche per il proprio modello, potrebbe non documentare sempre come queste caratteristiche vengono estratte e calcolate. Con un Feature Store, l'origine dei dati e le trasformazioni delle caratteristiche seguono uno schema coerente e facilmente comprensibile, promuovendo il riutilizzo del lavoro tra i team.

- **Coerenza tra training e servizio:** molte delle caratteristiche di machine learning sono utilizzate nei modelli in tempo reale. È essenziale garantire che i dati siano gestiti in modo uniforme tra l'addestramento e il servizio per ottenere previsioni coerenti ed impedire che il modello si comporti in modo imprevisto a causa di discrepanze nei dati. Questa coerenza è nota come "training-serving skew". Problemi come ritardi nell'aggiornamento dei dati o differenze nelle trasformazioni dei dati possono verificarsi se non viene mantenuta questa coerenza.

2.1 Struttura

Il Feature Store è composto da **5** componenti principali:

1. **Feature Engineering (Transformations)**: facilitano l'automazione e la standardizzazione delle pipeline di dati, offrendo allo stesso tempo la flessibilità di selezionare, filtrare, aggregare e manipolare i dati grezzi in features riutilizzabili per i modelli di ML. Ad esempio vi sono strumenti di preprocessing per normalizzazione, calcolo delle variabili e calcolo di features derivate.

2. **Feature Storage**: Il concetto di Feature Storage è cruciale per supportare efficacemente il recupero delle features attraverso i livelli di feature serving. Tipicamente, questi sistemi di archiviazione includono sia un livello online che offline per soddisfare i diversi requisiti dei sistemi di feature serving. Lo storage offline è più statico e gestisce dati basati sul tempo, aggiungendoli piuttosto che riscriverli. Questo approccio garantisce che lo storage offline contenga tutti i dati elaborati e sia efficiente in termini di costi per l'archiviazione di grandi volumi di informazioni. D'altra parte, lo storage online memorizza solo gli ultimi vettori di features per una specifica entità del set di caratteristiche. Ad esempio, per un utente identificato da un ID su una piattaforma online, lo storage online conserverà solo la sua ultima visita. Questo approccio è particolarmente utile per applicazioni che richiedono un accesso rapido e diretto alle features più recenti, come nel caso di modelli che necessitano di autenticazione a due fattori (2FA).

3. **Feature Registry**: rappresenta una repository centralizzata fondamentale per gestire e tenere traccia delle caratteristiche disponibili. Questa repository non solo cataloga le caratteristiche, ma ne gestisce anche le descrizioni dettagliate, le versioni, i metadati e le dipendenze associate. Queste informazioni includono i dettagli sull'origine dei dati grezzi utilizzati per generare le feature, le trasformazioni applicate ai dati, il loro formato e il modo in cui queste sono utilizzate dai modelli di machine learning. Il Feature Registry è cruciale per garantire la tracciabilità, la riproducibilità e la gestione efficace delle caratteristiche all'interno dell'organizzazione. Facilita la collaborazione tra i team di data science e di sviluppo, assicurando che le caratteristiche siano documentate in modo chiaro e che sia possibile gestirne l'evoluzione nel tempo con un approccio controllato e strutturato.

4. **Feature Serving**: interfacce API progettate per consentire l'accesso alle features da parte dei data scientist. Queste API recuperano le features più recenti per una specifica entità necessarie per l'addestramento dei modelli, utilizzando dati storici, o per l'inferenza. Assicurano inoltre che i data scientist possano lavorare con dati aggiornati e rilevanti, contribuendo così a migliorare l'accuratezza e l'affidabilità delle previsioni.

5. **Feature Monitoring**: strumenti per monitorare le prestazioni e la qualità delle features, rilevando anomalie o degradi nel tempo e, nel caso di funzionalità in tempo reale, servono a garantire che il servizio rispetti le soglie di latenza desiderate.

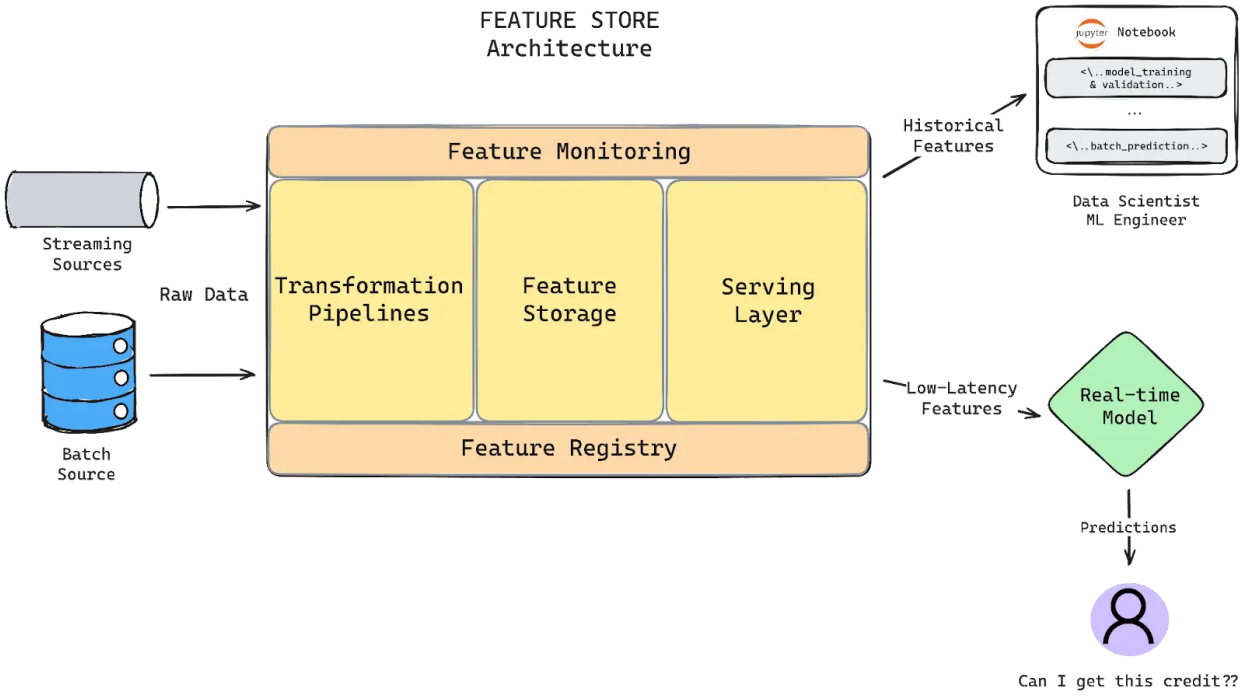


Figura 2: Struttura del Feature Store

2.2 Vantaggi e svantaggi

Oltre alle motivazioni principali descritte in precedenza, il Feature Store porta altri vantaggi quando viene utilizzanto nell'ambito ML:

- **Gestione centralizzata:** Fornisce un'unica fonte di verità per le funzionalità, migliorando la collaborazione e l'efficienza. Elimina la dispersione dei dati e garantisce che tutti i membri del team abbiano accesso alle stesse informazioni aggiornate.
- **Real-time/Offline features:** Supporta il recupero a bassa latenza delle funzioni per le applicazioni in tempo reale. Allo stesso tempo, permette l'elaborazione in batch, essenziale per l'addestramento dei modelli su grandi volumi di dati storici.
- **Scalabilità**: è progettato per gestire grandi volumi di dati e operazioni simultanee, garantendo che possa scalare con le esigenze dell'organizzazione, ottimizzando anche le prestazioni delle query.

Ovviamente hanno anche diversi svantaggi, valutabili in base alla tipologia e la struttura del proprio progetto:

- **Complessità di implementazione:** l'impostazione e la manutenzione di un Feature Store possono essere complesse e richiedere molte risorse. È necessario configurare l'infrastruttura, integrare varie fonti di dati e sviluppare pipeline di trasformazione.
- **Costo:** elevato investimento iniziale in termini di tempo e risorse, includendo l'acquisto di hardware, software e servizi cloud, nonché l'investimento in competenze tecniche per configurare e gestire il sistema. In seguito vi saranno anche i costi operativi, come l'archiviazione dei dati, la potenza computazionale e il personale dedicato alla manutenzione, che avranno un peso non indifferente.
- **Integrazione:** l'integrazione con le infrastrutture e pipelines esistenti può essere impegnativa. Potrebbero, infatti, sorgere problemi di compatibilità con i database e gli strumenti di Machine Learning già in uso. Anche la migrazione delle features esistenti in un nuovo feature store può essere un processo lungo e delicato.

3 Descrizione obiettivi

Il seguente progetto si basa sulla simulazione di un possibile use case reale in cui è possibile utilizzare il Feature Store. Dopo svariate analisi abbiamo individuato una valida applicazione per il **Controllo e Gestione del Traffico Aereo**, appoggiandoci al dataset Kaggle relativo ai ritardi aerei con dettagli su aeroporti e meteo.
Il dataset è composto dalle seguenti colonne:

Nome	**Tipo**	**Descrizione**
MONTH	Stringa	Mese
DAY_OF_WEEK	Stringa	Giorno della settimana
DEP_DEL15	Binario	Ritardo di partenza superiore a 15 minuti (1 è sì, 0 no)
DISTANCE_GROUP	Intero	Gruppo di distanza da percorrere dall'aeromobile in partenza
DEP_BLOCK	Stringa	Blocco di partenza (orario)
SEGMENT_NUMBER	Intero	Segmento che questo numero di coda sta percorrendo per il giorno
CONCURRENT_FLIGHTS	Intero	Voli concorrenti in partenza dall'aeroporto nello stesso blocco di partenza
NUMBER_OF_SEATS	Intero	Numero di posti sull'aeromobile
CARRIER_NAME	Stringa	Compagnia aerea
AIRPORT_FLIGHTS_MONTH	Intero	Media mensile dei voli dell'aeroporto
AIRLINE_FLIGHTS_MONTH	Intero	Media mensile dei voli della compagnia aerea
AIRLINE_AIRPORT_FLIGHTS_MONTH	Intero	Media mensile dei voli della compagnia aerea per l'aeroporto
AVG_MONTHLY_PASS_AIRPORT	Intero	Media mensile dei passeggeri per l'aeroporto di partenza
AVG_MONTHLY_PASS_ AIRLINE	Intero	Media mensile dei passeggeri per la compagnia aerea
FLT_ATTENDANTS_PER_PASS	Float	Assistenti di volo per passeggero per la compagnia aerea
GROUND_SERV_PER_PASS	Float	Addetti ai servizi a terra (sportello servizi) per passeggero per la compagnia aerea
PLANE_AGE	Intero	Età dell'aeromobile in partenza
DEPARTING_AIRPORT	Stringa	Aeroporto di partenza
LATITUDE	Float	Latitudine dell'aeroporto di partenza
LONGITUDE	Float	Longitudine dell'aeroporto di partenza
PREVIOUS_AIRPORT	Stringa	Aeroporto precedente da cui l'aeromobile è decollato

Nome	Tipo	Descrizione
PRCP	Float	Precipitazioni per il giorno (in pollici)
SNOW	Float	Neve per il giorno (in pollici)
SNWD	Float	Neve al suolo per il giorno (in pollici)
TMAX	Float	Temperatura massima per il giorno
AWND	Float	Velocità massima del vento per il giorno

Tabella 1: Descrizione dei Dati

Nel contesto specifico, l'aeroporto internazionale Hartsfield-Jackson di Atlanta, tra i più trafficati al mondo, sta attuando un sistema di modellazione predittiva dei ritardi di partenza. Questo sistema mira a ottimizzare il controllo del traffico aereo e migliorare l'efficienza operativa. I 2 team di sviluppo hanno un task ciascuno da svolgere:

- **Task 1**: il primo compito è incentrato sulla previsione della probabilità che un volo parta con un ritardo superiore a 15 minuti. Questo processo inizia con una fase iniziale di preprocessing dei dati, essenziale per preparare e ottimizzare le informazioni estratte dai vari dataset utilizzati. Questa fase include la pulizia dei dati per rimuovere eventuali valori nulli o outlier, la standardizzazione delle caratteristiche per uniformare le scale di misurazione e la creazione di nuove feature derivando informazioni più dettagliate dai dati grezzi disponibili.

- **Task 2**: il secondo task è incentrato sulla previsione delle condizioni atmosferiche una volta in possesso dei dati del volo, più nello specifico cerchiamo fare inferenza sulla presenza, o assenza, di pioggia in fase di partenza del volo tramite le informazioni dello stesso.

L'obiettivo principale che sta alla base del progetto, quindi, è quello di sfruttare la potenza del Feature Store per cercare di effettuare la laboriosa e complicata fase di pre-processing una sola volta durante il primo task e utilizzare le features ottenute per completare agilmente il task 2. Ma quale Feature Store utilizzare? Dopo svariate ricerche abbiamo deciso di utilizzare Hopsworks.

3.1 Hopsworks

Hopsworks è una piattaforma progettata per fornire una suite completa di strumenti per la gestione e l'analisi dei dati, il machine learning e l'intelligenza artificiale. Caratteristica non di poco conto è il fatto di essere un software completamente open-source e viene definito, secondo un articolo di ricerca [1], come la piattaforma più performante tra gli attuali Feature Store in cloud per quanto riguarda il training e le query di inferenza online.

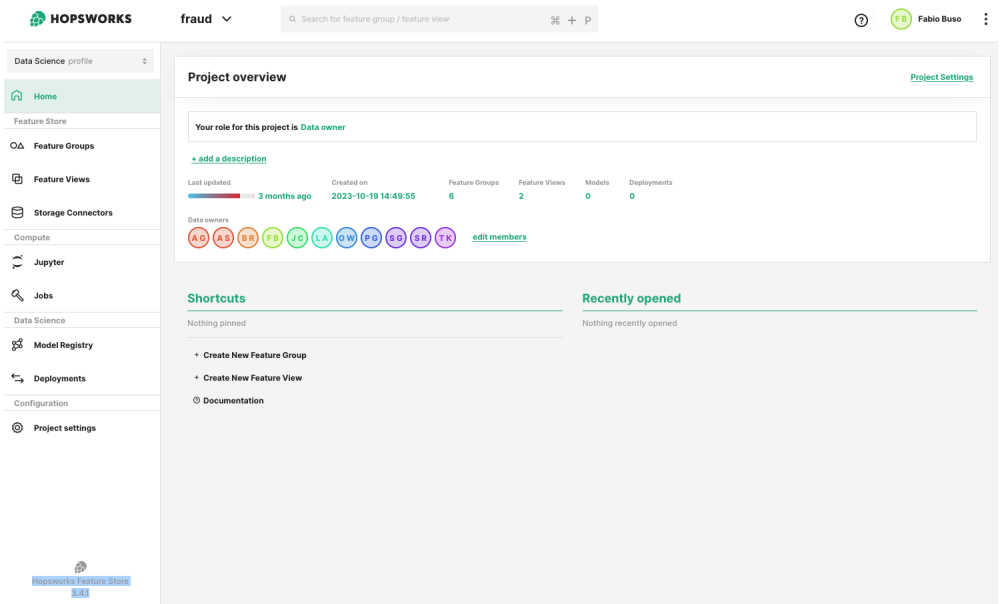


Figura 3: Hopsworks

La scelta all'interno di questo progetto è stata dettata da alcune caratteristiche chiave di Hopsworks, che lo hanno reso ottimo per effettuare tutti i nostri test:

- In primis, Hopsworks contiene all'interno della sua vasta suite, un potente Feature Store che permette sia di utilizzare la modalità offline in batch che l'opzione online in stream. In più la piattaforma è progettata per scalare orizzontalmente, consentendo di gestire in modo efficiente grandi insiemi di dati e carichi di lavoro computazionali, perfetto per il nostro dataset relativo ai voli.
- In secondo luogo, dà la possibilità agli sviluppatori di "democratizzare" la struttura ML, fornendo un servizio completamente gratuito di "Serverless Feature Store", il quale offre tutti i vantaggi di un feature store pronto all'uso su larga scala senza la necessità di un'infrastruttura complessa.
- Volendo utilizzare Python per lo sviluppo del progetto, Hopsworks è l'unico Feature Store Python-centrico, dando la possibilità di leggere e scrivere DataFrames (in Python o Spark) in pochi secondi. Fornisce un'implementazione e un servizio nuovo di API, che permette ai developer di specificare quali features selezionare da quale gruppo di caratteristiche, come unirle e quali utilizzare nelle condizioni di unione.

Nella seguenti sezioni andiamo ad indicare i vari concetti utilizzati durante lo sviluppo del progetto

3.1.1 Feature Group

Un Feature Group è un insieme logico di features che vengono raggruppate insieme perché condividono un contesto comune. Sono memorizzate in un doppio sistema di database, con

un database online che memorizza i valori più recenti in formato tabellare orientato alle righe (ad esempio, database di valori-chiave) per un rapido recupero delle features durante l'inferenza online, mentre il secondo database offline memorizza i valori storici in un archivio orientato alle colonne. È costituito da uno schema, metadati e tabelle:

- Lo schema può essere fornito esplicitamente o implicitamente tramite un DataFrame.
- I metadati contengono un nome fornito dall'utente, una versione, una chiave primaria e un flag che specifica se deve essere abilitato online o meno.
- La chiave primaria è necessaria per recuperare le righe dei dati delle features online ed evitare i dati duplicati, mentre il numero di versione consente di supportare i test A/B delle features da parte di diversi modelli e gli aggiornamenti del sistema ML.

Nell'esempio in Figura 4, abbiamo un Feature Group che contiene tutte le caratteristiche del dataset relative al meteo, come precipitazioni, pollici giornalieri di neve, ecc.

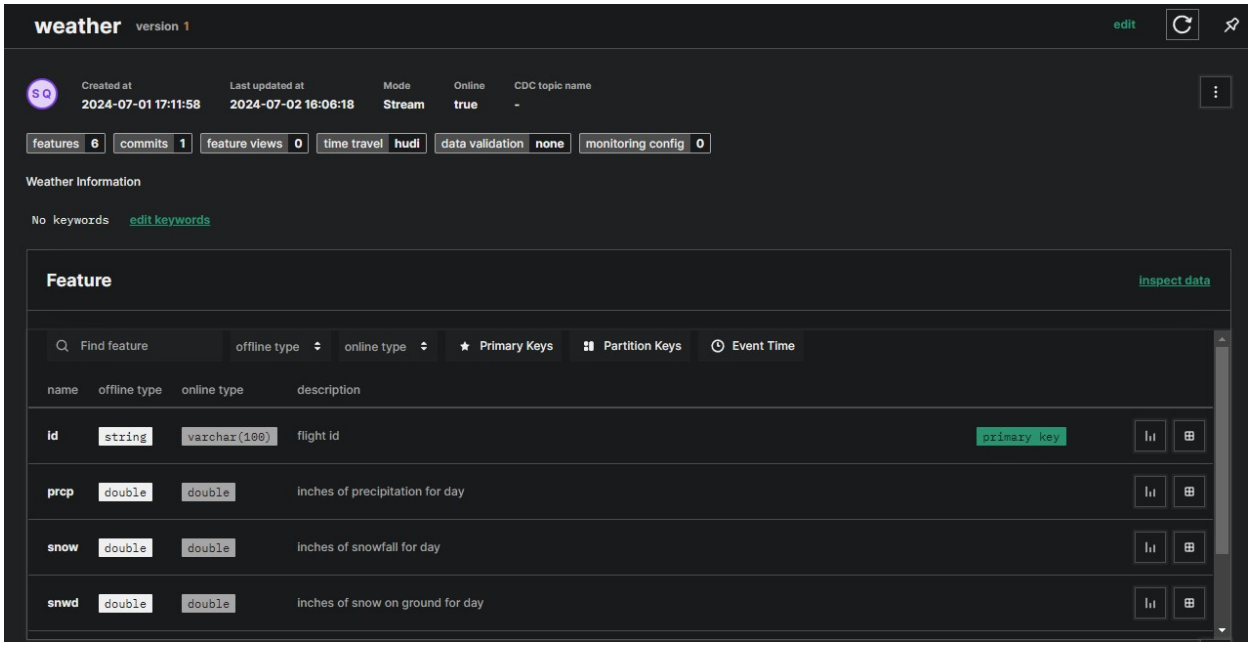


Figura 4: Esempio di Feature Group

3.1.2 Feature View

La Feature View è una selezione di feature (ed etichette) da uno o più Feature Groups. Si crea una vista di feature unendo le caratteristiche di Feature Groups esistenti che vengono presentati come un unico dataset.

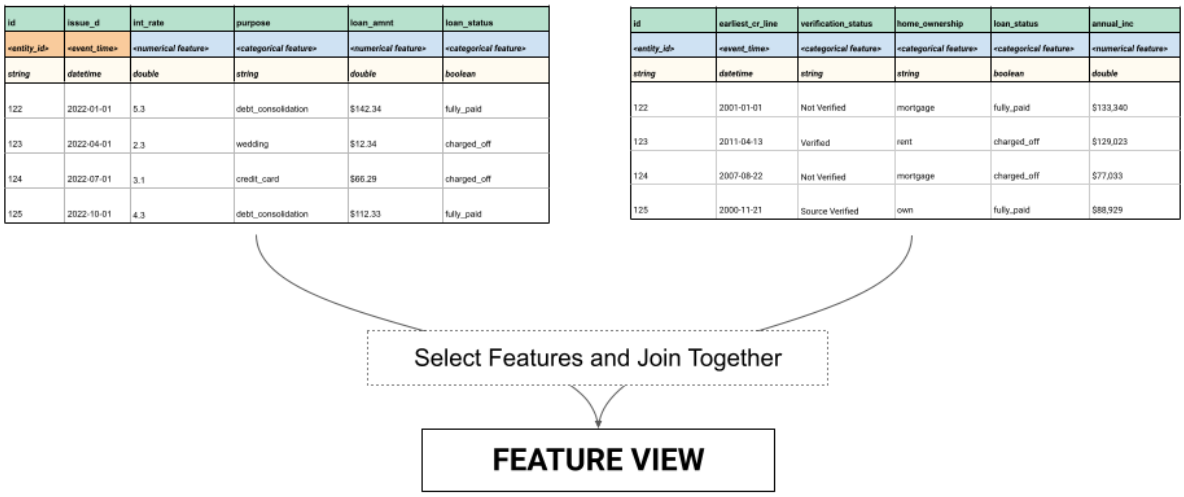


Figura 5: Creazione della Feature View

Vengono utilizzate le Feature View come un'astrazione per risolvere i problemi legati alla lettura dei dati dal Feature Store:

1. Forniscono una semplice API Python per la lettura point-in-time (le feature utilizzate riflettono accuratamente lo stato dei dati a un momento specifico nel passato) di dati corretti di feature/label per l'addestramento e l'inferenza

2. Forniscono un unico schema di modello per evitare l'incompatibilità tra le pipeline di addestramento e di inferenza

3. Consentono la riproduzione dei dati di addestramento utilizzando solo i metadati

3.1.3 Model Registry

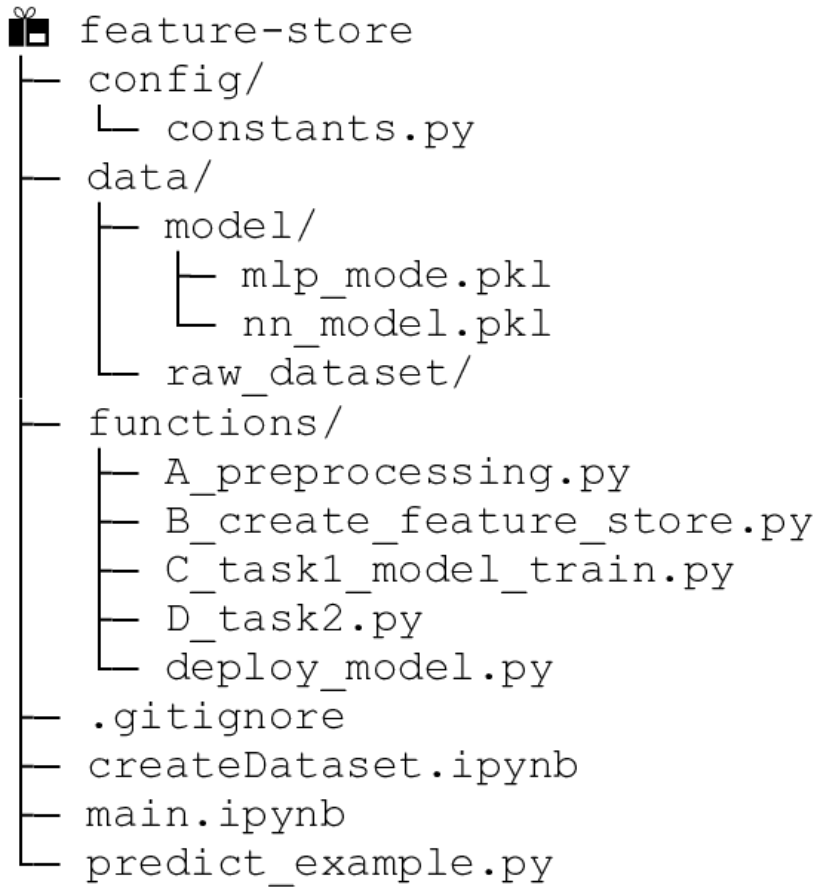
Il Model Registry è un sistema centralizzato per il tracciamento, la gestione e la distribuzione dei modelli di ML. Funziona come una repository per i modelli addestrati, fornendo funzionalità per la registrazione, la versione, la valutazione, e il monitoraggio di essi, come accade all'interno del nostro progetto (per entrambi i task che verranno specificati in seguito). È il luogo designato dove gli sviluppatori possono pubblicare i loro modelli durante la fase di sperimentazione. Inoltre, Hopsworks consente di condividere facilmente i modelli con il team interno e con gli stakeholders, facilitando la collaborazione e assicurando che tutti i membri del team possano accedere e contribuire alla valutazione e all'evoluzione dei modelli implementati.

4 Implementazione

La repository Github del progetto è disponibile a questo link.
L'implementazione è stata effettuata con l'utilizzo di Python e i Jupyter Notebook, utilissimi per tutta la parte di pre-processing e visualizzazione dei dati con l'utilizzo di grafici. Librerie in uso:

- **Hopsworks**: libreria principale fornita da Hopsworks, che permette tutta l'interazione con le API messe a disposizione dalla piattaforma.
- **Scikit-learn e Keras**: utilizzate per la creazione dei modelli di ML e il pre-processing di alcune feature (come ad esempio il label encoding o la standardizzazione dei valori).
- **Numpy e Pandas**: utilizzate per la gestione del dataset e per la gestione di operazioni matematiche come media, deviazione standard, ecc.
- **Pickle**: utilizzata per salvare su disco il modello allenato con i dati di training.
- **Seaborn e Matplotlib**: usati per visualizzare i vari grafici.
- **uuid**: usata per la creazione di ID univoci utilizzati per identificare ogni record dei dataset.

4.1 Struttura della repository



Abbiamo deciso di suddividere la repository nel seguente modo:

- **Config**: cartella che contiene i file di configurazione
  - **constants.py**: costanti utilizzate per inizializzare Hopsworks (come per esempio API KEY). Viene ignorata durante il push su Github.
- **Data**: cartella contenente i dati utilizzati nel progetto
  - **Model**: cartella contenente i modelli ottenuti dal training
  - **raw-dataset**: cartella contenente tutti i dati grezzi del dataset in version csv
- **Functions**: cartella contenente le funzioni utilizzate nel progetto
  - **A_preprocessing.py**: File Python contenente tutte le funzioni utili al preprocessing del dataset (normalizzazione, label encoding, creazione grafici, ecc).
  - **B_create_feature_store.py**: File Python contenente tutte le funzioni utili alla gestione di Hopsworks, come connessione con il server, creazione Feature Groups, ecc.

  - **C_task1_model_train.py**: File Python contenente la parte di creazione dei 2 modelli per il task 1, compreso di training, validazione e salvataggio
  - **D_task2.py**: File Python contenente la parte di creazione del modello per il task 2, compreso di training, validazione e salvataggio.
  - **deploy_model.py**: File Python per il deploy del modello su Hopworks.

- **.gitignore**: file utilizzato per specificare quali file e directory devono essere ignorati da Git durante commit e push.

- **createDataset.ipynb**: Notebook utilizzato per effettuare le operazioni iniziali per creare il dataset da utilizzare all'interno del progetto. Da molteplici file csv relativi al meteo, informazioni degli aeroporti e degli aerei di linea, tramite una pesante fase di merge e pulizia si ottiene il file csv pronto all'uso.

- **main.ipynb**: Notebook principale del progetto.

- **predict_example.py**: File per effettuare le predizioni per il modello durante il deploy.

4.2 Descrizione del dataset

Il dataset utilizzato per il nostro progetto consisteva in diversi file, ciascuno focalizzato su una specifica componente del dominio di interesse. Di seguito ne forniamo una descrizione dettagliata.

- **ONTIME_REPORTING_XX.csv**: file separati per ciascun mese, ognuno contenente dettagli riguardanti gli orari di partenza e di arrivo dei voli.

- **airport_weather_2019.csv**: file al cui interno sono inclusi i dati sulle condizioni atmosferiche rilevate tramite stazioni posizionate all'interno degli aeroporti.

- **airports_list.csv**: contiene la lista degli aeroporti con i relativi id da collegare agli altri file.

- **B43_AIRCRAFT_INVENTORY.csv**: file contenente le varie informazioni degli aerei, come anno di costruzione e numero massimo di passeggeri, presenti nel dominio di interesse.

- **CARRIER_DECODE.csv**: questo file contiene la lista di carrier con i relativi nomi e id da collegare agli altri file.

- **AIRPORT_COORDINATES.csv**: fornisce le coordinate, latitudine e longitudine, di ogni aeroporto.

4.3 Visualizzazione dei dati

In questa sezione cerchiamo di riassumere la parte di visualizzazione dei dati eseguita per cercare di avere una migliore idea della distribuzione del dataset.

Come prima cosa vogliamo mostrare la distribuzione dei voli in orario e dei voli che presentano un ritardo superiore a 15 minuti:

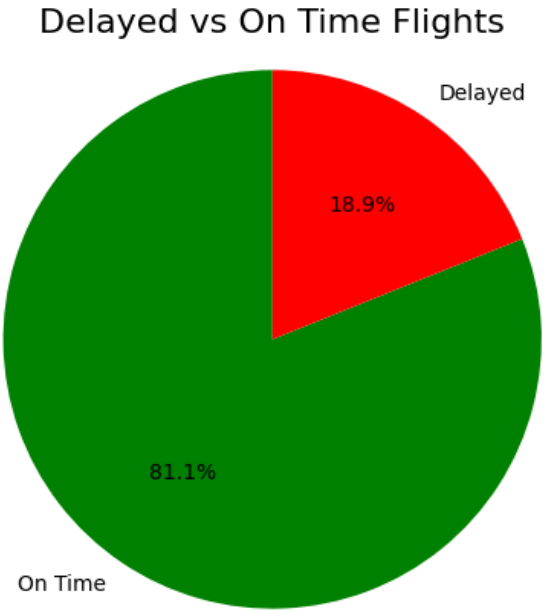


Figura 6: Distribuzione ritardi voli

Possiamo notare che il dataset contiene molti voli che riteniamo in orario $\approx 81\%$, specifichiamo che voli che hanno un ritardo sulla partenza inferiore a 15 minuti rientrano anch'essi in questa categoria. Successivamente, abbiamo voluto analizzare la correlazione tra le varie colonne del dataset e la colonna target del task 1. In particolare, mostriamo di seguito una heatmap che visualizza la correlazione con la colonna "PART_OF_DAY". Questa colonna è stata creata appositamente a partire dalla colonna "DEP_TIME_BLK" presente nel dataset, la quale indica il blocco di partenza del volo. Inoltre, la correlazione è stata calcolata anche per i vari giorni della settimana.

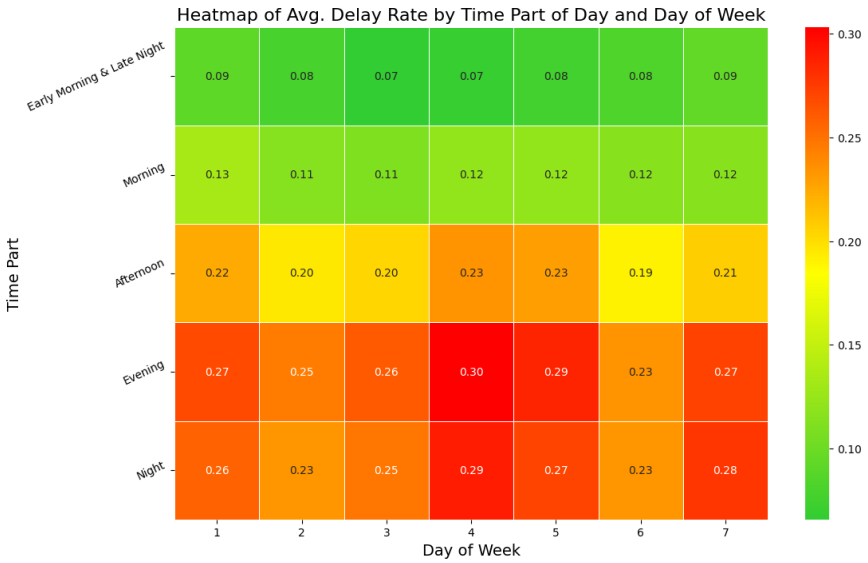


Figura 7: Correlazione PART_OF_DAY

Possiamo notare che le correlazioni non sono molto alte ma notiamo un piccolo incremento generale rispetto alla colonna che indica la parte della giornata in cui il volo parte, mentre risulta in questo caso il giorno della settimana risulta poco significativo.

4.4 Pre-processing

La fase di pre-processing riveste un ruolo cruciale poiché consente di ottenere un dataset consistente, fondamentale per lo sviluppo di modelli predittivi accurati. Inizialmente, abbiamo caricato i vari dataset e condotto un'analisi preliminare dei dati. In particolare, il dataset relativo alle condizioni atmosferiche presentava diversi valori nulli che abbiamo dovuto gestire attentamente, considerando la loro importanza per il raggiungimento degli obiettivi del progetto.
I valori nulli sono stati gestiti in modi differenziati in base al tipo di dato. In particolare, per le colonne "PRCP", che indica la quantità di pioggia in inches, e "TMAX", che rappresenta la temperatura massima registrata durante la giornata, abbiamo optato per una strategia specifica. Dato che il numero di valori nulli è stato insignificante, abbiamo deciso di sostituire questi valori con la media delle rilevazioni effettuate nella stessa stazione:

```
1 weather['TMAX'].fillna(round(weather.groupby('ORIGIN_AIRPORT_ID')['TMAX'].transform('mean')
      , 1), inplace=True)
2 weather['AWND'].fillna(round(weather.groupby('ORIGIN_AIRPORT_ID')['AWND'].transform('mean')
      , 1), inplace=True)
```

Invece, per quanto riguarda i valori delle colonne "SNOW" e "SNWD", che indicano rispettivamente la quantità di neve caduta e la quantità di neve sul terreno durante la giornata (espressi entrambi in inches), abbiamo riscontrato una percentuale elevata di valori nulli, circa il 30%. Per gestire questa situazione, abbiamo deciso di trattare le colonne mancanti come se non ci fosse stata neve quel giorno, quindi abbiamo riempito i valori mancanti con 0.

Successivamente, abbiamo sviluppato una funzione che, partendo dal file contenente i dati mensili dei voli, costruisce un dataset completo con tutte le feature necessarie. Inizialmente, abbiamo rimosso tutti i voli cancellati o quei voli che non avevano il valore corrispondente per l'orario di partenza o il numero univoco dell'aereo.

```
monthly_data.drop(monthly_data.loc[monthly_data['DEP_TIME'].isna()].index, axis=0, inplace=
    True)
monthly_data.drop(monthly_data.loc[monthly_data['TAIL_NUM'].isna()].index, axis=0, inplace=
    True)
monthly_data.drop(monthly_data.loc[monthly_data['CANCELLED']==1].index, axis=0, inplace=
    True)
```

In seguito, abbiamo eseguito un merge con il file "*airports_list*" per ottenere il numero di posti disponibili sull'aereo relativo al volo. In aggiunta, abbiamo gestito i valori nulli della colonna riempiendoli con la media dei valori presenti nella stessa colonna.

```
# Merge aircraft info with main frame on tail number - get NUMBER_OF_SEATS
monthly_data = pd.merge(monthly_data, aircraft, how="left", on='TAIL_NUM')
# Fill missing aircraft info with means
monthly_data['NUMBER_OF_SEATS'].fillna((monthly_data['NUMBER_OF_SEATS'].mean()), inplace=
    True)
```

Abbiamo inoltre eseguito un merge con il file "*CARRIER_DECODE*" per ottenere il nome della compagnia aerea corrispondente a ciascun volo.

```
monthly_data = pd.merge(monthly_data, names, how='left', on=['OP_UNIQUE_CARRIER'])
```

Infine, abbiamo combinato i valori per ottenere i dati riguardanti le condizioni atmosferiche nell'aeroporto di partenza del volo.

```
monthly_data = pd.merge(monthly_data, weather, how='inner', on=['ORIGIN_AIRPORT_ID', 'MONTH
    ', 'DAY_OF_MONTH'])
```

4.5 Feature Engineering

In questa sezione descriviamo la serie di features che abbiamo costruito appositamente a partire dai dati pre-processati. Come prima feature, abbiamo deciso di calcolare il numero di voli che dovevano partire nello stesso blocco di tempo in ciascun aeroporto. Questo è stato ottenuto raggruppando per l'ID dell'aeroporto, il giorno del mese e il blocco di tempo designato per la partenza, seguito da un conteggio degli ID degli aerei.

```
monthly_data['CONCURRENT_FLIGHTS'] = monthly_data.groupby(['ORIGIN_AIRPORT_ID','
    DAY_OF_MONTH', 'DEP_TIME_BLK'])['OP_UNIQUE_CARRIER'].transform("count")
```

Due altre metriche che abbiamo calcolato includono il numero totale di voli eseguiti nel mese corrente in ciascun aeroporto e da ciascuna compagnia aerea. Per il primo calcolo, abbiamo raggruppato i dati per l'ID dell'aeroporto e contato i valori corrispondenti al numero di voli pianificati in ogni città in cui si trova l'aeroporto nel dato mese. Nel secondo calcolo, abbiamo raggruppato per l'ID della compagnia aerea e contato il numero di voli che partivano per ogni città.

```
monthly_data['AIRPORT_FLIGHTS_MONTH'] = monthly_data.groupby(['ORIGIN_AIRPORT_ID'])['
    ORIGIN_CITY_NAME'].transform('count')
monthly_data['AIRLINE_FLIGHTS_MONTH'] = monthly_data.groupby(['OP_UNIQUE_CARRIER'])['
    ORIGIN_CITY_NAME'].transform('count')
```

Una metrica molto semplice ma di interesse è l'età dell'aereo utilizzato per il volo. Questo valore è calcolato utilizzando i dati di costruzione di ciascun aeromobile contenuti nel file "*B43_AIRCRAFT_INVENTORY*".

```
monthly_data['MANUFACTURE_YEAR'].fillna((monthly_data['MANUFACTURE_YEAR'].mean()), inplace=
    True)
monthly_data['PLANE_AGE'] = 2019 - monthly_data['MANUFACTURE_YEAR']
```

4.6 Connessione Hopsworks

La fase successiva all'elaborazione dei dati, riguarda la configurazione e l'implementazione di Hopsworks all'interno del progetto. Inizialmente, volendo testare tutte le possibili funzionalità di Hopsworks, come l'integrazione delle pipeline CI/CD direttamente da Git, oppure l'uso di Apache Kafka come piattaforma di elaborazione dei flussi, si è effettuata la configurazione con l'utilizzo di AWS. Gli step seguiti sono stati:

- Creazione di un cross-account role, ovvero un ruolo di AWS Identity and Access Management (IAM) che consente agli utenti di accedere in modo sicuro alle risorse, come per esempio la potenza in cloud delle macchine virtuali.
- Creazione di un Instance profile, che ci ha permesso di associare il nostro profilo con un'istanza di Elastic Compute Cloud offerta da AWS e ha permesso ai nodi cluster di Hopsworks di accedere ad altre risorse come gli S3 Bucket e CloudWatch.
- Creazione di storage nell'S3 Bucket di AWS
- Creazione di una chiave SSH
- Deploy di un cluster Hopsworks con la configurazione creata in precedenza

Questo ci ha portato, però, ad ottenere sì un cluster Hopsworks funzionante, ma durante i vari test abbiamo notato che la versione gratuita delle risorse AWS si era già esaurita velocemente (soprattutto la potenza di calcolo dell'Elastic Compute Cloud). Essendo un progetto universitario, abbiamo deciso di effettuare altre ricerche per trovare una soluzione alternativa e abbiamo scoperto che Hopsworks permette l'inizializzazione di un Feature Store serverless. Infatti, il servizio permette a studenti e piccoli sviluppatori di ottenere facilmente e rapidamente una propria infrastruttura (con funzionalità limitate), semplicemente registrando un account. Una volta creata l'istanza serverless, è stato possibile connettere agilmente il nostro codice Python con l'account Hopsworks grazie all'uso di API keys, generate all'interno della GUI del Feature Store.

Il codice per la connessione è il seguente (le chiavi di accesso vengono richieste la prima volta direttamente all'interno di VSCode quando la funzione viene eseguita):

```
def connect_to_hopsworks():
    project = hopsworks.login()

    fs = project.get_feature_store()

    return project, fs
```

4.7 Creazione Feature Groups

Una volta effettuata la connessione con Hopsworks, si può procedere a caricare tutti i dati ottenuti precedentemente nella fase di preparazione e pulizia del dataset. Poiché non era presente una chiave univoca per i record, abbiamo utilizzato la libreria uuid per generare delle stringhe alfanumeriche da utilizzare come *primary key*.

```
df['ID'] = [uuid.uuid4() for _ in range(len(df))]
df['ID'] = df['ID'].astype(str)
```

Successivamente abbiamo suddiviso il dataset rispetto alle informazioni che ogni colonna poteva fornirci, in particolare:

- **Voli e aeroporti**: blocco di feature relativo ai voli e agli aeroporti, come mese e giorno di registrazione del volo, distanza percorsa e nome aeroporto.
- **Condizioni climatiche**: tutte le features riguardanti precipitazioni, neve, temperatura, ecc.
- **Target**: il target effettivo del nostro task, ovvero se il volo è in ritardo o meno (**DEP_DEL15**).

Per fornire informazioni dettagliate sui dati, facilitando la loro gestione, comprensione e utilizzo in diverse applicazioni, abbiamo deciso di creare metadata per ogni suddivisione, concentrandoci in particolare sulla loro descrizione. L'ultimo step è stato la creazione dei Feature Groups:

```
def create_feature_group(name, version, description, primary_key, online_enabled, fs, df,
    feature_descriptions):
    fg = fs.get_or_create_feature_group(
        name=name,
        version=version,
        description=description,
        primary_key=primary_key,
        online_enabled=online_enabled,
    )

    # Insert data into feature group
    fg.insert(df)

```

```
    for desc in feature_descriptions:
        fg.update_feature_description(desc["name"], desc["description"])

    return fg
```

Vengono specificati:

- Nome del Feature Group
- Versione (così da utilizzare sempre quella più aggiornata)
- Descrizione del Feature Group
- Chiave primaria
- Se abilitare l'online e rendere disponibili i dati non solo in batch

Dopo la sua creazione, vengono inseriti i vari blocchi di feature dal dataset e i loro metadata.

4.8 Task 1: previsione ritardi dei voli

Il primo task che viene svolto dal team di sviluppo riguarda la creazione di un modello di ML in grado di predire quando un volo effettua più di 15 minuti di ritardo. Volendo simulare l'esecuzione del progetto da parte di vari team nella stessa azienda, il primo task non ha ancora la possibilità di accedere al Feature Store, non avendo ancora nulla al suo interno. Il team, quindim deve utilizzare tutti i dati precedentemente processati in locale. Come modello viene creato un Multi-layer Perceptron classifier, che è un tipo di rete neurale feedforward composta da più strati nascosti (hidden layers) tra l'input e l'output. Viene utilizzato per classificare se un certo tipo di volo è in ritardo rispetto alla sua ora di partenza oppure no, indicato come 1 o 0 nel campo DEP_DEL15.

```
df.drop(['ID'], axis=1, inplace=True)

X_train, X_test, y_train, y_test = preprocessing.prepare_data_for_ML_model(df)
mlp_model = create_model.define_MLP_classifier()

train_losses, test_losses, train_accuracies, test_accuracies = create_model.
    train_MLP_classifier(mlp_model, X_train, X_test, y_train, y_test)
create_model.show_MLP_performance(train_losses, test_losses, train_accuracies,
    test_accuracies)
```

Come si può notare, vengono eliminate inizialmente le colonne che non sono utili per la previsione, come il campo *ID*. Separatamente, nel file ***C_task1_model_train.py*** vengono definite le seguenti funzioni per la creazione del modello:

```
def define_MLP_classifier():
    mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam',
    max_iter=300, random_state=42)
    return mlp

```

```
def train_MLP_classifier(mlp, X_train, X_test, Y_train, Y_test):
    train_losses = []
    test_losses = []
    train_accuracies = []
    test_accuracies = []

    for i in range(10):
        mlp.partial_fit(X_train, Y_train, classes=np.unique(Y_train))

        # Calculate metrics on training set
        train_loss = mlp.loss_
        train_losses.append(train_loss)
        train_accuracy = accuracy_score(Y_train, mlp.predict(X_train))
        train_accuracies.append(train_accuracy)

        # Calculate metrics on test set
        test_loss = np.mean(mlp.loss_curve_)
        test_losses.append(test_loss)
        test_accuracy = accuracy_score(Y_test, mlp.predict(X_test))
        test_accuracies.append(test_accuracy)

        print(f"Epoch {i+1}/{10} - Train Loss: {train_loss:.4f} - Train Acc: {
    train_accuracy:.4f} - Test Loss: {test_loss:.4f} - Test Acc: {test_accuracy:.4f}")

    return train_losses, test_losses, train_accuracies, test_accuracies
```

Particolarità del codice:

- Vengono utilizzati 4 array in cui salvare l'accuratezza e la loss relativa ai dati di training e ai dati di test, i quali verranno utilizzati successivamente come andamento del modello durante le varie epoche.
- Dopo vari test abbiamo notato che 20 epoche erano abbastanza per avere un accuratezza accettabile, quindi abbiamo settato quel valore in modo statico.

Volendo comparare vari modelli e siccome Hopsworks permette, successivamente, di selezionare il modello con più accuratezza in automatico, abbiamo deciso di creare anche una Neural Network. In questo caso le epoche sono state ridotte a 10 e come risultato otteniamo già un array al cui interno vi sono i dati relativi a loss e accuracy di training e test set.

```
def define_NN_model(X_train):
    nn_model=Sequential()
    nn_model.add(Dense(32, input_shape=(1, X_train.shape[2]), activation='relu'))
    nn_model.add(Dense(32, activation='relu'))
    nn_model.add(Dropout(0.2))
    nn_model.add(Dense(1, activation='sigmoid'))
    nn_model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['accuracy'])

    return nn_model

def train_NN_model(neural_model, X_train, X_test, Y_train, Y_test):

    history = neural_model.fit(X_train, Y_train, epochs=10, batch_size=200, verbose=1,
    validation_data=(X_test, Y_test))
```

```
14
15      _, train_acc = neural_model.evaluate(X_train, Y_train, verbose=0)
16      _, test_acc = neural_model.evaluate(X_test, Y_test, verbose=0)
17      print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
18      return history
```

4.8.1 Risultati

I risultati ottenuti da entrambi i modelli possono essere considerati soddisfacenti: il valore di accuracy si assesta circa a 0.816, evitando il fenomeno dell'overfitting, fenomeno che si verifica quando un modello di ML si adatta eccessivamente ai dati di addestramento, acquisendo una rappresentazione troppo specifica e dettagliata dei dati stessi. Il modello di Neural Network presenta una buona generalizzazione e ha un grande potenziale per ulteriori miglioramenti, come ad esempio la variazione delle epoche utilizzate oppure l'introduzione di nuovi strati e livelli interni.

La nota negativa è relativa alla complessità del modello: l'addestramento della rete neurale è stata computazionalmente costosa, richiedendo molte risorse di calcolo, con un tempo medio di esecuzione per l'addestramento e la visualizzazione dei risultati di circa 8 minuti. Di contro il modello Multi-Layer Perceptron mostra un comportamento più "classico" con la loss e l'accuratezza di training leggermente migliori di quelle di test, ma sembra convergere più lentamente rispetto al precedente, con miglioramenti graduali fino all'epoca 17. Rispetto invece alle performance abbiamo un netto miglioramento delle tempistiche, con solamente 4 minuti tra training e grafici. Di seguito sono riportate le curve di accuracy e loss durante il training dei modelli:

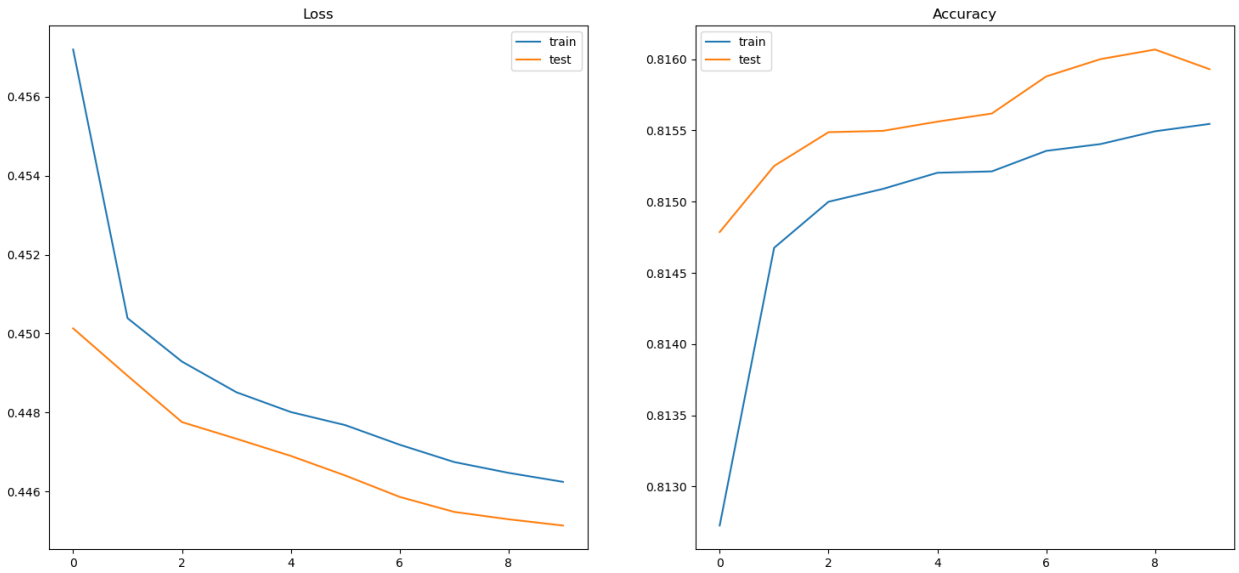


Figura 8: Neural Network

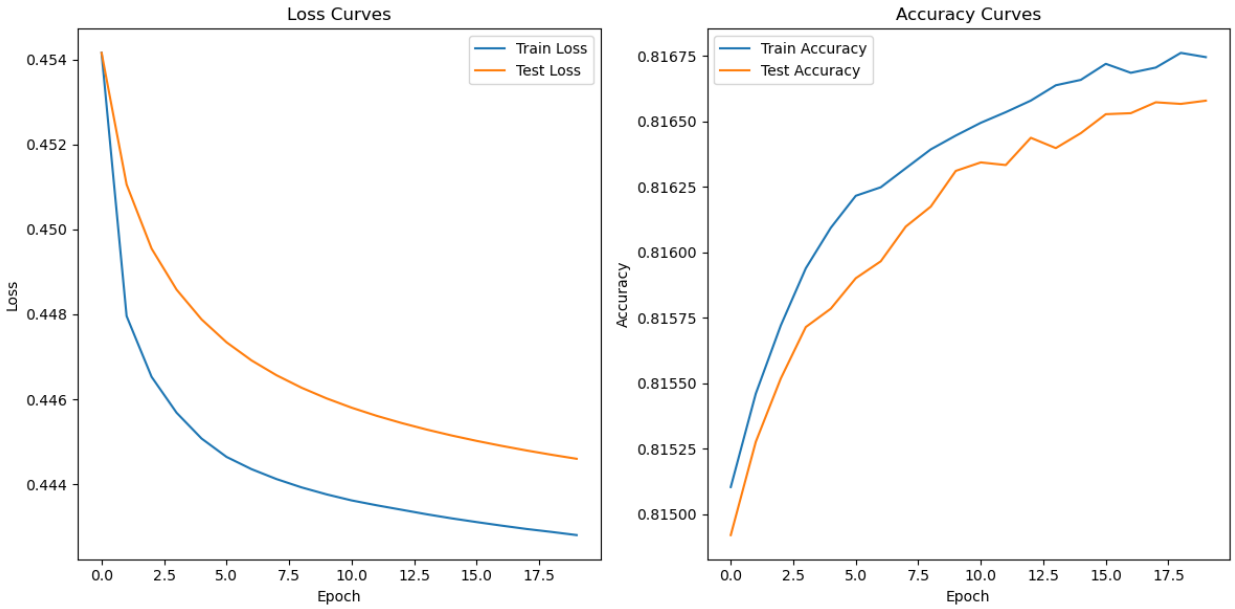


Figura 9: MLP Classifier

4.8.2 Deploy dei modelli

Il processo di deployment permette ai modelli che abbiamo addestrato di essere inseriti in un ambiente dove può ricevere richieste, elaborare dati in tempo reale o batch, e restituire predizioni. I modelli precedentemente ottenuti vengono salvati in locale tramite la libreria *Pickle* e poi vengono seguiti i seguenti step:

1. **Registrazione del modello**: prima del deployment vero e proprio, il modello viene registrato nel Model Registry di Hopsworks, che traccia versioni, metadati e metriche di performance. Qui è possibile memorizzarne diverse versioni e confrontarne le prestazioni. Di seguito andiamo a creare quello che è il suo schema, il quale ci garantisce che i dati rispettino la struttura e i tipi previsti, riducendo il rischio di errori durante il funzionamento del modello.

```
# Create input schema using X_train
input_schema = Schema(X_train)

# Create output schema using y_train
output_schema = Schema(y_train)

# Create a ModelSchema object specifying the input and output schemas
model_schema = ModelSchema(
    input_schema=input_schema,
    output_schema=output_schema,
)

# Convert the model schema to a dictionary
model_schema.to_dict()
```

   Utilizziamo poi lo schema per effettuare il salvataggio all'interno del Model Registry, indicando anche la versione del modello, come metrica l'accuratezza e infine descrizione

di cosa il modello predice. L'accuratezza ci permette di andare via via a selezionare con il tempo il modello più preciso.

```
    tf_model  = mr.python.create_model(
        name=f"flight_delay_{model_name}",
        version = version,
        metrics={"accuracy": accuracy},
        description=description,
        model_schema=model_schema
    )

    tf_model.save(f'./data/model/{model_name}.pkl')
```

La chiamata alla libreria ha la seguente forma:

```
    nn_model = deploy_model(project, "Neural Network model for predicting flight
    delays",  X_train_NN, y_train_NN, 1, "nn_model", history.history['accuracy'][-1])

    mlp_model = deploy.deploy_model(project, "MLPClassifier model for predicting
    flight delays", X_train, y_train, 1, "mlp_model", train_accuracies[-1])
```

2. **Deploy**: una volta salvato il modello in Hopsworks, abbiamo due possibilità: effettuare il deploy direttamente dalla GUI di Hopsworks, oppure crearla tramite codice. Noi abbiamo optato per crearla tramite Python, per mantenere consistenza nelle operazioni. Si sceglie il modello e la versione e poi si chiama la funzione *deploy*. Durante il deploy è possibile specificare anche lo script Python che verrà utilizzato per fare le predizioni insieme al modello.

```
    my_model = mr.get_model("flight_delay_nn_model", version=1)
    my_deployment = my_model.deploy()
```

Purtroppo questa fase, a causa di limitate funzionalità del piano gratuito e da errori della dashboard per visualizzare i log di debug, non siamo stati in grado di effettuare con successo il deploy dell'applicazione, la quale sarà sicuramente un upgrade per il futuro.

4.9 Task 2: previsione delle condizioni atmosferiche

Per il secondo task, abbiamo sviluppato un modello per predire le condizioni atmosferiche utilizzando le feature estratte dalle informazioni dei voli. Le features utilizzate sono prelevate direttamente dal Feature Store, il che ci consente di evitare il ricalcolo di tutte le features aggiuntive non presenti nel dataset originale, in quanto sono state precalcolate e salvate direttamente nel Feature Store. Per eseguire questo passaggio, abbiamo utilizzato le Feature View. Come nei database relazionali, questa non è una tabella effettiva o un Feature Group, ma una vista logica, poiché le feature sono contenute nei vari Feature Group. La differenza sta nel fatto che le Views possono fornire feature provenienti da diversi Feature Groups e vengono utilizzate per leggere dati sia in fase di training che di serving. Sono state create

diverse Feature Views; in questo caso, ci limitiamo a mostrare quella riferita esclusivamente al task 2:

```
weatherFatures = flight_airport_fg.select(["id", "part_of_day", "plane_age", "
    departing_airport", "previous_airport", "plane_age"]).join(weather_fg.select_all()).
    join(target_fg.select_all())

# Get or create the 'weather_prcp_predi' feature view
weather_view = fs.get_or_create_feature_view(
    name='weather_prcp_predi',
    version=1,
    query=weatherFatures,
    labels=["prcp"]
)
```

Come prima cosa, viene effettuata una query selezionando dai vari Feature Groups le feature che ci interessa includere nella View. Come mostrato nel codice, viene eseguita un'operazione di join sui vari Feature Groups. Successivamente, viene creata la View, assegnandole un nome, la query precedentemente calcolata e le varie label che corrispondono alle colonne target. Il seguente codice esegue il passaggio di estrazione delle features dalla Feature View:

```
X_train, X_test, y_train, y_test = weather_view.train_test_split(
    description='flight_delay_online_fv',
    test_size=0.2,
)
```

Lo scopo del modello è prevedere le condizioni atmosferiche al momento della partenza del volo, in particolare vogliamo predire la presenza o l'assenza di pioggia. Per fare questo, abbiamo trasformato la colonna "PRCP", descritta nelle sezioni precedenti, in una variabile booleana:

```
# Trasformiamo il task in binario
y_train = y_train > 0.0
y_test = y_test > 0.0
```

Successivamente i dati vengono usati per allenare una Support Vector Machine con kernel lineare, il parametro di regolarizzazione C è stato impostato inizialmente ad 1, risulta dopo vari tentativi essere il miglior compromesso tra performance e tempo richiesto per il training:

```
supportVec = LinearSVC(tol=1e-5, C=1)
supportVec.fit(X_train, y_train)
y_pred = supportVec.predict(X_test)
```

La svm performa discretamente con un'accuracy del 71% sul secondo task, come accennato precedentemente il parametro C non risulta essere molto influente sul risultato soprattutto pesando il tempo di addestramento rispetto al miglioramento ottenuto. Riportiamo la matrice:

	precision	recall	f1-score	support
False	0.72	0.98	0.83	910575
True	0.67	0.11	0.19	387238
accuracy			0.72	1297813
macro avg	0.70	0.54	0.51	1297813
weighted avg	0.71	0.72	0.64	1297813

Figura 10: Classification report

Possiamo vedere che la recall sul "False" risulta essere molto precisa mentre per il dato "True" ci suggerisce che fatica ad identificare consistentemente la presenza di pioggia.

Successivamente, abbiamo provato ad addestrare una rete neurale utilizzando le funzioni già descritte in precedenza per cercare un miglioramento delle performance. La rete è stata allenata su 10 epoche. Abbiamo osservato che si stabilizza già dalla prima epoca su un valore di precisione di circa 0.81, suggerendo che il limite per il modello sia già molto vicino. Inoltre, la loss rimane costante intorno a 0.45, indicando una stabilità del modello anche su questo fronte.

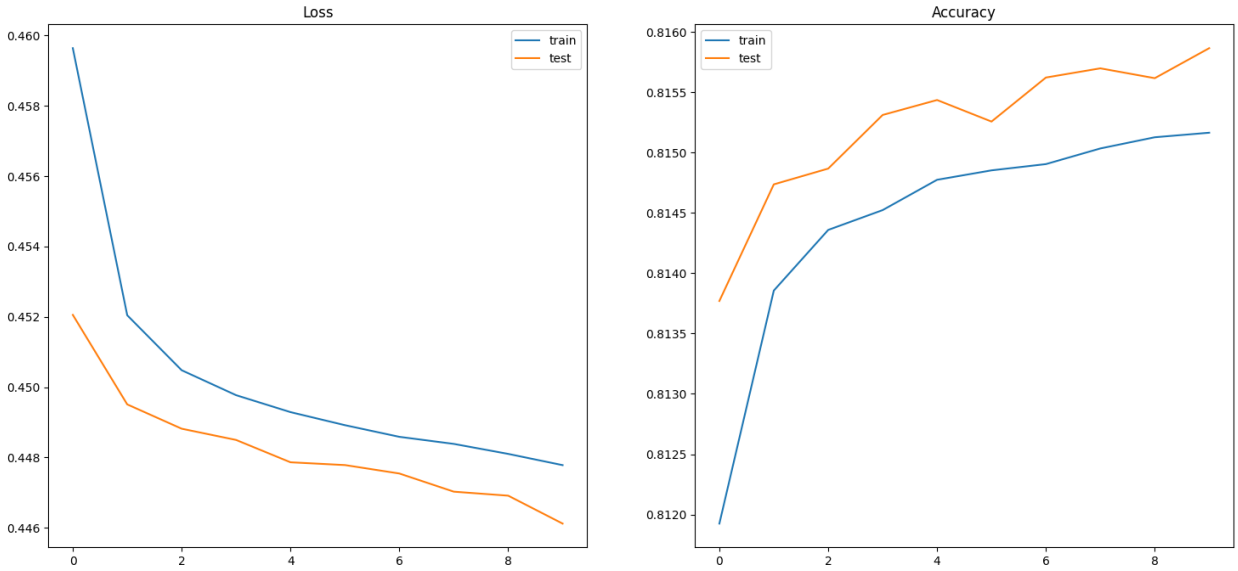


Figura 11: Report della Neural Network

Possiamo ipotizzare che il miglioramento in performance dalla SVM alla Rete Neurale è dovuta alla condizione di linearità del kernel imposto sulla SVM.

5 Conclusioni

Durante lo svolgimento del progetto abbiamo avuto la possibilità di lavorare con una nuova tecnologia che in questi anni sta prendendo sempre più piede. L'adozione di Hopsworks come feature store ha migliorato significativamente l'efficienza nella gestione delle feature. Grazie alla centralizzazione e al loro riutilizzo, è stato possibile ridurre notevolmente il tempo necessario per preparare i dati per i modelli di machine learning quando si è realizzato il task 2. Anche la capacità di estrarre, trasformare e caricare le feature direttamente dal feature store ha semplificato il processo di sviluppo dei modelli, riducendo la complessità delle pipeline e migliorando l'efficienza operativa.

5.0.1 Note relative ad Hopsworks

Nello specifico, Hopsworks si è dimostrato un buonissimo strumento da utilizzare in progetti con grandi quantità di dati, grazie alla sua facile integrazione e alla possibilità di richiedere facilmente e velocemente feature già precedentemente processate da altri team di sviluppo. La possibilità di creare Feature View con semplici query tra le varie Feature Group, ci ha permesso di adattare i dati ai nuovi obbiettivi di predizione, riuscendo ad evitare il preprocessing del dataset originale.

Il fatto che Hopsworks sia un progetto open-source e disponga di una documentazione dettagliata ha contribuito in modo significativo a risolvere rapidamente eventuali problemi e a sfruttare al meglio le funzionalità offerte. Un ruolo fondamentale è stato giocato dal comparto UI, particolarmente dalla dashboard messa a disposizione, infatti, grazie all'interfaccia web intuitiva e ben strutturata, è stato possibile monitorare e verificare che tutte le operazioni venissero eseguite correttamente. Questo controllo ha garantito che la creazione di Feature Groups, Feature Views, training data e l'upload dei modelli includessero i metadata necessari per il corretto svolgimento del progetto. L'efficienza e la trasparenza fornite dalla piattaforma hanno quindi permesso di migliorare l'intero processo di sviluppo, facilitando sia la gestione dei dati che l'integrazione dei modelli in un ambiente di produzione.

La possibilità di effettuare lo split di training, test e validation set tramite funzionalità implementate direttamente in Hopsworks, garantisce la riduzione di errori umani quando si manipolano i dati, permettendo consistenza e robustezza.

Come qualsiasi tecnologia, presenta alcune note negative e queste sono quelle che abbiamo riscontrato noi:

- **Problemi di scalabilità**: In alcuni scenari, specialmente con carichi di lavoro molto grandi o complessi, Hopsworks può presentare problemi di scalabilità. Questo può

portare a una diminuzione delle prestazioni o a interruzioni del servizio. Durante la fase di caricamento del dataframe all'interno delle varie Feature Group, è capitato più volte che il server ci restituisse un errore, in particolare veniva segnalato il superamento di una soglia di carico di lavoro e di tempo trascorso.

- **Curva di apprendimento ripida**: durante lo sviluppo del progetto abbiamo trovato che la quantità di tempo e risorse necessarie per familiarizzare con tutte le caratteristiche e le best practices di Hopsworks è significativa. Abbiamo dovuto dedicare molte ore allo studio della documentazione e seguendo tutorial forniti direttamente dalla piattaforma.

- **Difficoltà di debug**: molto spesso abbiamo riscontrato errori utilizzando le API di Hopsworks senza riuscire a comprendere cosa non funzionasse e senza trovare soluzioni online. Abbiamo incontrato limitazioni anche all'interno della dashboard, dove i log implementati erano spesso inaccessibili a causa di "Internal Server Error" o problemi con strumenti come OpenSearch Dashboards, utilizzato per visualizzare i log del deployment dei modelli.

5.0.2 Sviluppi futuri

Essendo un progetto ristretto, non vi è stata la possibilità di renderlo effettivamente attivo in produzione e renderlo il più ottimizzato possibile. I possibili sviluppi futuri riguardano infatti sia la parte di ampliamento del progetto, sia il miglioramente delle implementazioni correnti sia livello algoritmico sia a livello di integrazione e sfruttamento delle API di Hopsworks.

L'espansione delle funzionalità del modello può essere attuata tramite un pre-processing più accurato, utilizzando altri parametri per l'identificazione di correlazioni più nascoste tra le varie features e la possibilità di aggiungere nuove caratteristiche nel dataset finale. É possibile, poi, migliorare i modelli attuali anche tramite l'adozione di tecniche avanzate di tuning degli iperparametri, che consentono di trovare combinazioni ottimali di parametri per massimizzare le performance.

L'uso di pipeline CI/CD permette di implementare un sistema automatizzato che integri, testi e validi i modelli in modo continuo, dando la possibilità di individuare e correggere tempestivamente errori o regressioni nel codice e/o nei dati. Questo può garantire anche a nuove feature e modelli di essere rilasciate in maniera del tutto autonoma e sicura. L'implementazione potrà essere fatta sia tramite Github con i file YAML, sia utilizzando le funzionalità già integrate all'interno di Hopsworks. Purtroppo non siamo stati in grado di poter testare questa funzionalità perchè non presente nella versione gratuita utilizzata per questo progetto.

Infine, per rendere questo progetto applicabile concretamente, uno sviluppo futuro riguarda sicuramente l'integrazione di un software che utilizza il deployment del modello su Hopsworks per poter fare inferenza e ottenere previsioni e dati utili in base al task scelto dal team di analisi. Questa funzionalità risulta molto utile anche in caso in uno scenario di utilizzo dei modelli in tempo reale.

Riferimenti bibliografici

[1] Javier de la Rúa Martínez et al. “The Hopsworks Feature Store for Machine Learning”. In: SIGMOD/PODS ’24 (2024), pp. 135–147. DOI: 10.1145/3626246.3653389. URL: https://doi.org/10.1145/3626246.3653389.