UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Progetto per Ingegneria della Conoscenza:

Trasporto ferroviario

Christian Corvino, 760415, c.corvino3@studenti.uniba.it

Link repository GitHub:

https://github.com/ccorvino3/Progetto-ICON-TrasportoPubblico.git

A.A. 2023-24

Indice

Introduzione

Obiettivo del progetto

Il dataset utilizzato per questo progetto raccoglie informazioni sui ritardi dei treni in Francia, includendo dati relativi ai tempi di percorrenza, cancellazioni, ritardi all'orario di partenza e di arrivo, nonché indicatori quantitativi e percentuali delle cause dei ritardi. L'obiettivo è sviluppare un sistema di apprendimento automatico in grado di prevedere il ritardo medio dei treni, integrando moduli di apprendimento supervisionato, una base di conoscenza logica realizzata in Prolog e modelli probabilistici bayesiani. In questo modo, il sistema non solo stima i ritardi sulla base delle caratteristiche rilevate, ma supporta anche il processo decisionale, ad esempio individuando il treno con il ritardo massimo o verificando la convenienza di determinati ritardi.

Strumenti utilizzati

Il caso di studio è stato sviluppato utilizzando **Python**, un linguaggio di programmazione versatile e ampiamente utilizzato per l'analisi dei dati e l'implementazione di algoritmi di apprendimento automatico. L'IDE scelto per il progetto è stato **Visual Studio Code**, che offre un ambiente di sviluppo altamente personalizzabile e supporta facilmente Python e altre tecnologie.

Per la gestione e l'analisi dei dati, sono state utilizzate diverse librerie, tra cui:

- **Pandas**: impiegato per il trattamento e l'analisi dei dati, particolarmente utile per il preprocessing dei dati del dataset dei ritardi dei treni.
- **Numpy**: utilizzato per il calcolo scientifico, offre supporto per array a più dimensioni e funzioni matematiche avanzate, fondamentali per la gestione e l'elaborazione dei dati numerici.
- **Scikit-learn**: sfruttato per implementare i modelli di apprendimento automatico. Fornisce una vasta gamma di algoritmi di apprendimento supervisionato, come gli alberi decisionali, e strumenti per la selezione e la valutazione dei modelli.
- **Matplotlib**: utilizzato per la visualizzazione dei dati e dei risultati delle previsioni, utile per creare grafici che supportano l'analisi dei ritardi.
- **NetworkX**: impiegato per la gestione e l'analisi di grafi, utile per rappresentare e manipolare le relazioni tra i diversi treni e le loro caratteristiche.
- **pgmpy**: utilizzato per implementare modelli probabilistici bayesiani, che sono fondamentali per il trattamento dell'incertezza e per il calcolo delle probabilità legate ai ritardi.

- **pyswip**: libreria che permette di interagire con il motore di **SWI-Prolog** da Python, utilizzata per l'integrazione della base di conoscenza logica nel sistema e per il ragionamento su regole e fatti in Prolog.

L'uso di queste librerie ha consentito di sviluppare un sistema completo e integrato per la previsione e la gestione dei ritardi dei treni.

Dataset

Il dataset considerato per l'analisi contiene le seguenti colonne:

- **Year**: Anno di riferimento per i dati.
- **Month**: Mese in cui i dati sono stati raccolti.
- **Departure station**: Stazione di partenza del treno.
- **Arrival station**: Stazione di arrivo del treno.
- **Average travel time (min)**: Tempo medio di percorrenza dei treni, espresso in minuti.
- **Number of expected circulations**: Numero di circolazioni previste dei treni in un dato periodo.
- **Number of cancelled trains**: Numero di treni cancellati.
- **Number of late trains at departure**: Numero di treni in ritardo alla partenza.
- **Average delay of late departing trains (min)**: Ritardo medio dei treni in ritardo alla partenza, espresso in minuti.
- **Average delay of all departing trains (min)**: Ritardo medio di tutti i treni in partenza, espresso in minuti.
- **Comment (optional) delays at departure**: Commenti opzionali riguardanti i ritardi alla partenza.
- **Number of trains late on arrival**: Numero di treni in ritardo all'arrivo.
- **Average delay of late arriving trains (min)**: Ritardo medio dei treni in ritardo all'arrivo, espresso in minuti.
- **Average delay of all arriving trains (min)**: Ritardo medio di tutti i treni all'arrivo, espresso in minuti.
- **Comment (optional) delays on arrival**: Commenti opzionali riguardanti i ritardi all'arrivo.
- **% trains late due to external causes (weather, obstacles, suspicious packages, malevolence, social movements, etc.)**: Percentuale di treni in ritardo a causa di fattori esterni come condizioni meteo, ostacoli, pacchi sospetti, ecc.
- **% trains late due to railway infrastructure (maintenance, works)**: Percentuale di treni in ritardo a causa di problemi legati all'infrastruttura ferroviaria, come manutenzione o lavori.
- **% trains late due to traffic management (rail line traffic, network interactions)**: Percentuale di treni in ritardo a causa di problemi nella gestione del traffico ferroviario.

- **% trains late due to rolling stock**: Percentuale di treni in ritardo a causa di problemi con il materiale rotabile.
- **% trains late due to station management and reuse of material**: Percentuale di treni in ritardo a causa di problemi nella gestione delle stazioni o riutilizzo di materiale.
- **% trains late due to passenger traffic (affluence, PSH management, connections)**: Percentuale di treni in ritardo a causa di problemi legati al traffico passeggeri, come affluenza o gestione delle connessioni.
- **Number of late trains > 15min**: Numero di treni in ritardo maggiore di 15 minuti.
- **Average train delay > 15min**: Ritardo medio dei treni in ritardo maggiore di 15 minuti.
- **Number of late trains > 30min**: Numero di treni in ritardo maggiore di 30 minuti.
- **Number of late trains > 60min**: Numero di treni in ritardo maggiore di 60 minuti.
- **Period**: Periodo di riferimento per i dati raccolti.
- **Delay due to external causes**: Ritardo causato da fattori esterni.
- **Delay due to railway infrastructure**: Ritardo causato da problemi legati all'infrastruttura ferroviaria.
- **Delay due to traffic management**: Ritardo causato da problemi nella gestione del traffico ferroviario.
- **Delay due to rolling stock**: Ritardo causato da problemi con il materiale rotabile.
- **Delay due to station management and reuse of material**: Ritardo causato da problemi nella gestione delle stazioni o riutilizzo di materiale.
- **Delay due to travellers taken into account**: Ritardo causato da fattori legati ai passeggeri.

Pre-processing del dataset

Nella fase di pre-processing viene eseguita la ridenominazione di alcune colonne per renderle più chiare e comprensibili, la conversione della colonna *'Month'* in un valore intero, l'eliminazione dei valori nulli sostituiti con la media della rispettiva colonna, ed infine la normalizzazione delle colonne numeriche utilizzando lo StandardScaler per migliorare le performance dei modelli.

Sono state eliminate colonne di minore rilevanza come '*Comment (optional) delays on arrival*' e '*Comment (optional) delays at departure*'. Inoltre, sono state rimosse anche '*Arrival station*' e '*Departure station*' perché sono informazioni categoriche che, se codificate tramite one-hot encoding, avrebbero generato un numero eccessivo di variabili, aumentando inutilmente la complessità del modello. La colonna '*Period*' è stata rimossa in quanto rappresenta una combinazione delle colonne '*Year*' e '*Month*', espressa nel formato Year-Month. Le seguenti

colonne, invece, sono state rimosse dal dataset per evitare il data leakage: *'Delay due to rolling stock'*, '*Delay due to station management and reuse of material*', *'Delay due to travellers taken into account*', '*% trains late due to rolling stock*', '*% trains late due to station management and reuse of material*', '*Average delay of all arriving trains (min)*', '*Number of trains late on arrival*', '*Average train delay > 15min*', '*Number of late trains > 15min*', '*Number of late trains > 30min*', '*Number of late trains > 60min*'. Il data leakage si verifica quando il modello ha accesso a informazioni che non sarebbero disponibili al momento della previsione, rischiando così di influenzare in maniera ingannevole le performance del modello. Rimuovendo queste colonne si garantisce che il modello venga addestrato e validato solo sui dati realmente disponibili, migliorando l'affidabilità e la generalizzazione delle previsioni.

Quando il dataset è destinato all'apprendimento bayesiano, è fondamentale trasformare le variabili continue in variabili discrete, poiché i modelli bayesiani operano in modo più efficace con dati categoriali. Per questo motivo, le variabili numeriche sono state discretizzate utilizzando quattro quantili, in modo che ciascun intervallo contenga approssimativamente lo stesso numero di osservazioni. Questo metodo garantisce una suddivisione equilibrata dei dati, riducendo l'impatto di valori estremi e migliorando la stima delle probabilità condizionali. La discretizzazione basata sui quantili consente di ottenere una rappresentazione più adatta all'inferenza bayesiana, migliorando la stabilità del modello e la sua capacità di generalizzare su nuovi dati.

Infine, i dataset pre-processati sono stati **salvati in due file CSV**:

- **df_preprocessed_continuous.csv**, contenente i dati con variabili continue normalizzate per i modelli di apprendimento automatico. Posizionato nel percorso "**Trasporto pubblico\progettazione\dataset\df_preprocessed_continuous.csv**"
- **df_preprocessed_discrete.csv**, contenente i dati discretizzati per l'apprendimento bayesiano. Posizionato nel percorso "**Trasporto pubblico\progettazione\dataset\df_preprocessed_discrete.csv**"

Questa suddivisione permette di ottimizzare l'uso dei dati in base alle esigenze dei diversi algoritmi di modellazione.

Apprendimento supervisionato

L'**apprendimento supervisionato** è una tecnica di machine learning in cui il modello viene addestrato utilizzando un set di dati etichettati, ovvero dati per i quali sono già noti i risultati o le etichette. L'obiettivo primario è quello di insegnare al modello a mappare gli input alle etichette corrette, in modo che possa fare previsioni accurate su nuovi dati. Durante l'addestramento, il modello confronta le sue previsioni con le etichette reali e utilizza la differenza per aggiornare i suoi parametri attraverso un processo di ottimizzazione. Gli algoritmi di apprendimento supervisionato possono essere utilizzati per vari compiti, tra cui classificazione (quando le etichette sono discrete) e regressione (quando le etichette sono continue).

L'apprendimento supervisionato consiste nell'addestrare un modello utilizzando un dataset in cui ogni esempio è associato a un valore target. Nel nostro progetto, il modello è stato addestrato sui dati relativi ai ritardi dei treni, in cui le caratteristiche osservate (come tempi di percorrenza, cancellazioni e altri indicatori) sono state correlate al ritardo medio. Durante il processo di training, il modello apprende le relazioni tra le variabili in ingresso e il target, in modo da poter effettuare previsioni su dati nuovi. L'approccio adottato ha incluso tecniche come la validazione incrociata e l'ottimizzazione degli iperparametri, garantendo così una maggiore accuratezza e robustezza nelle previsioni.

Scelta dei modelli

Nel task di predizione della colonna '*Number of trains late on arrival*', l'approccio di apprendimento supervisionato è stato adottato per stabilire una relazione tra le caratteristiche del dataset e il numero di treni in ritardo all'arrivo. Poiché il target è una variabile numerica, sono stati scelti modelli di regressione in grado di catturare sia relazioni lineari sia non lineari tra le feature. In particolare, è stata valutata una serie di algoritmi:

- La **Regressione Lineare** è un modello di apprendimento supervisionato utilizzato per stimare una variabile continua sulla base di un insieme di variabili indipendenti. L'obiettivo è trovare una relazione lineare tra le feature di input e il valore target, in modo che il modello possa effettuare previsioni accurate. Il modello assume che esista una combinazione lineare dei predittori che meglio approssima il valore da predire. Strutturalmente, la regressione lineare assegna un peso a ciascuna variabile indipendente, rappresentando il suo contributo alla previsione del target. L'addestramento del modello consiste nell'ottimizzare questi pesi in modo da minimizzare l'errore tra i valori previsti e quelli reali. Questo avviene attraverso tecniche come la discesa del gradiente o il metodo dei minimi quadrati. È un modello semplice e interpretabile, adatto a dati con relazioni lineari, ma può risultare limitato quando il rapporto tra le variabili è più complesso o influenzato da outlier e multicollinearità.
- Il **Support Vector Regression (SVR)** è un modello di apprendimento supervisionato basato sui principi delle **Support Vector Machines (SVM)**, progettato per risolvere problemi di regressione. A differenza dei metodi classici, l'SVR cerca di trovare una funzione che predica i valori target con un margine di tolleranza, ignorando piccoli errori e concentrandosi sui punti più influenti, detti **support vectors**. Questa caratteristica lo rende

particolarmente efficace per dati rumorosi o con relazioni non lineari. L'SVR può utilizzare diversi kernel (lineare, polinomiale, RBF) per adattarsi a diverse distribuzioni dei dati, permettendo così una maggiore flessibilità nel modellare relazioni complesse.

- Il **Random Forest Regressor** è un algoritmo di apprendimento supervisionato basato su un insieme di alberi decisionali, progettato per migliorare la precisione e ridurre il rischio di overfitting rispetto a un singolo albero. Funziona costruendo molti alberi decisionali indipendenti durante la fase di addestramento e aggregando le loro previsioni tramite una media, rendendolo particolarmente robusto alle variazioni nei dati. Ogni albero è addestrato su un **sottoinsieme casuale del dataset** (bootstrap sampling), e a ogni nodo viene selezionato casualmente un sottoinsieme delle feature per effettuare le suddivisioni, riducendo la correlazione tra gli alberi. Questo approccio permette al modello di catturare pattern complessi e mitigare il rischio di overfitting.
- Il **Gradient Boosting Regressor** è un algoritmo di apprendimento supervisionato basato su un insieme di alberi decisionali costruiti in modo sequenziale. A differenza di metodi come il Random Forest, che combinano alberi indipendenti, il Gradient Boosting crea ogni nuovo albero correggendo gli errori del precedente, riducendo progressivamente la differenza tra le predizioni e i valori reali. Questo approccio consente di ottenere un modello altamente accurato e capace di catturare anche relazioni complesse nei dati. L'algoritmo è sensibile agli iperparametri, come il numero di alberi, la profondità massima e il tasso di apprendimento, che devono essere ottimizzati per evitare il sovradattamento. Il Gradient Boosting è ampiamente utilizzato per problemi di regressione grazie alla sua capacità di migliorare iterativamente le prestazioni, rendendolo ideale per compiti di predizione con dati complessi.

Scelta degli iperparametri

Nel processo di selezione degli iperparametri è stata adottata una strategia che combina la Grid Search con la tecnica di Repeated K-Fold Cross Validation. In questo approccio il dataset viene suddiviso in cinque parti, ripetute tre volte, utilizzando RepeatedKFold con n_splits=5 e n_repeats=3. Ogni ripetizione prevede che, a turno, ogni fold venga utilizzato come set di test, mentre gli altri fungono da training set, ottenendo così una stima più stabile e affidabile delle prestazioni del modello. La scelta di 5 fold e 3 ripetizioni rappresenta un compromesso equilibrato: da un lato consente una valutazione robusta, e dall'altro mantiene il tempo computazionale a livelli ragionevoli.

La Grid Search esplora in maniera sistematica una griglia di combinazioni di iperparametri definite per ciascun modello. Ad ogni configurazione viene applicata la validazione incrociata tramite il metodo di Repeated K-Fold, e il modello viene valutato sulla base del criterio del negativo dell'errore quadratico medio (neg_mean_squared_error). L'obiettivo è trovare la configurazione che minimizzi tale errore, migliorando così la capacità del modello di generalizzare su dati non visti. Inoltre, l'impiego del parametro n_jobs=-1 permette di sfruttare tutte le risorse computazionali disponibili, accelerando il processo di ricerca. Questo approccio integrato garantisce una selezione ottimale degli iperparametri, contribuendo in modo significativo al miglioramento delle prestazioni complessive del modello.

Per il modello **LinearRegression** gli iperparametri sono assenti, poiché questo algoritmo si basa su una semplice relazione lineare tra le variabili indipendenti e la variabile target. Non essendoci alcun parametro da ottimizzare, il modello viene addestrato direttamente sui dati senza necessità di configurazioni aggiuntive. La scelta di LinearRegression è spesso motivata dalla sua interpretabilità e dalla sua efficienza computazionale, rendendolo un punto di riferimento ideale per valutare le

prestazioni dei modelli più complessi. Essendo un modello di base, LinearRegression funge da baseline utile per confrontare i risultati ottenuti con approcci più sofisticati, permettendo di comprendere se l'aggiunta di complessità porta a miglioramenti significativi nelle previsioni.

Per il modello **SVR** sono stati scelti i seguenti iperparametri:

1. **C**: Questo parametro regola il compromesso tra l'errore di addestramento e la complessità del modello. Un valore alto di C cerca di ridurre al minimo l'errore di addestramento, ma potrebbe portare a overfitting, mentre un valore basso di C promuove la generalizzazione riducendo la complessità del modello a scapito dell'errore di addestramento. In questo caso, si sono scelti i valori 0.1, 1 e 10 per testare diverse intensità nella penalizzazione degli errori.

2. **Kernel**: Indica la funzione da utilizzare per trasformare i dati in uno spazio ad alta dimensione in modo da soddisfare una separazione lineare. I due kernel scelti per l'esplorazione sono: linear: per problemi che sono separabili linearmente; rbf (Radial Basis Function): un kernel non lineare che è molto potente, particolarmente utile quando i dati non sono separabili linearmente. L'esclusione del kernel polinomiale favorisce una ricerca più rapida in quanto il kernel linear e il kernel RBF sono i più comunemente usati, evitando la complessità aggiuntiva che potrebbe derivare da un kernel polinomiale.
3. **Gamma**: Questo parametro si applica al kernel RBF e controlla l'influenza che un singolo punto di dati ha su ogni altra osservazione. Un gamma basso implica una influenza "lontana" per ogni punto, mentre un gamma elevato aumenta l'influenza diretta di ogni punto sul modello. Con valori scelti come "scale" (che è il valore predefinito e si adatta automaticamente in base ai dati) e 0.1, si esplorano sia un'interpretazione automatica che una versione più semplificata.
4. **Epsilon**: Definisce l'ampiezza della zona di errore tollerato intorno alla linea di regressione; punti di training che finiscono in questa zona non contribuiscono alla funzione obiettivo. Ciò permette di ridurre l'impatto dei dati rumorosi. Scegliendo epsilon con valori 0.1 e 0.2, si cerca di ottimizzare la propensione a limitare l'errore senza sacrificare troppo l'adattamento al dataset.

Per il modello **RandomForestRegressor** sono stati scelti i seguenti iperparametri:

1. **n_estimators**: Questo parametro determina il numero di alberi decisionali che compongono la foresta. Un numero maggiore di alberi (ad esempio, 200 contro 100) tende a migliorare la precisione del modello, riducendo il rischio di overfitting e migliorando la generalizzazione, ma aumenta anche il tempo di calcolo. In questo caso, sono stati scelti 100 e 200 per esplorare l'effetto di vari livelli di complessità del modello.
2. **max_depth**: Controlla la profondità massima di ciascun albero, ossia quanto lontano un albero può crescere prima di fermarsi. Limitarne la profondità evita alberi troppo complessi che potrebbero adattarsi troppo ai dati di addestramento, causando overfitting. I valori scelti sono None (nessun limite), 5 e 10, permettendo di testare alberi più complessi rispetto a modelli meno profondi.
3. **min_samples_split**: Specifica il numero minimo di campioni necessari per dividere un nodo. Se il numero di campioni in un nodo è inferiore a questo valore, il nodo non verrà diviso ulteriormente. Valori più alti riducono la profondità e la complessità degli alberi, contribuendo a evitare overfitting. I valori scelti sono 2, 5 e 10, per esplorare l'influenza di diversi numeri minimi di campioni per la divisione.
4. **min_samples_leaf**: Indica il numero minimo di campioni che devono trovarsi in una foglia dell'albero. Un valore maggiore di 1 riduce l'overfitting impedendo che piccoli dettagli,

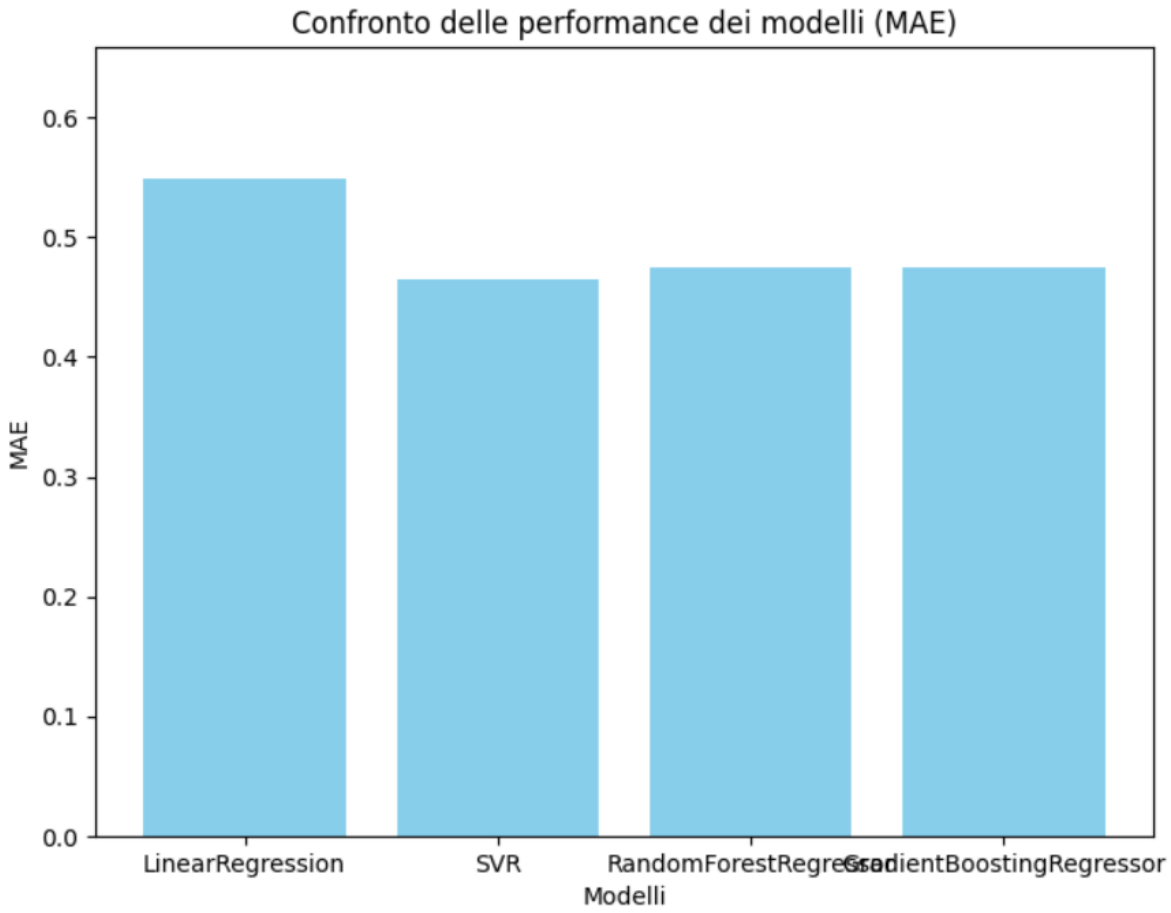
rappresentanti delle fluttuazioni nei dati, vengano adattati dalle foglie. Sono stati scelti i valori 1 e 2 per testare alberi che siano sufficientemente generali da evitare di adattarsi eccessivamente ai dati di addestramento.

5. **bootstrap**: Imposta se il campionamento dei dati dei singoli alberi venga effettuato con o senza ripetizione. Se True, il campionamento avviene con ripetizione, portando alla creazione di differenti set di dati di addestramento per ogni albero; se False, ogni albero è addestrato su un sottoinsieme completo del dataset senza ripetizione. Questo parametro contribuisce alla diversità degli alberi nel modello ensemble, influenzando l'accuratezza e l'efficienza del modello.

Per il modello **GradientBoostingRegressor** sono stati scelti i seguenti iperparametri:

1. **n_estimators**: Questo parametro determina il numero di alberi che verranno costruiti nel modello. Un numero maggiore di alberi tende a migliorare l'accuratezza del modello, riducendo l'errore complessivo, ma aumenta anche il tempo di calcolo. In questo caso, i valori scelti sono 100 e 150 per esplorare l'effetto di diversi livelli di complessità del modello.
2. **learning_rate**: Il tasso di apprendimento controlla quanto ciascun albero contribuisce all'aggiornamento del modello. Un valore più basso (come 0.01) rallenta l'apprendimento, evitando che il modello si adatti troppo rapidamente ai dati e riducendo il rischio di overfitting. Al contrario, un valore più alto (come 0.1) consente di apprendere più velocemente, ma potrebbe portare a modelli meno generali.
3. **max_depth**: Questo parametro limita la profondità massima degli alberi. Un valore più elevato, come None (senza limiti), permette che gli alberi crescano finché non raggiungono la purezza dei nodi o il numero minimo di campioni. I valori scelti come 5 e 10 permettono di testare alberi meno profondi, riducendo la complessità e prevenendo il rischio di overfitting.
4. **subsample**: Questo parametro stabilisce la percentuale di campioni da utilizzare per ogni albero. Usare una percentuale più bassa di campioni per alberi diversi aiuta a ridurre la varianza e contribuisce a una maggiore diversificazione degli alberi nel modello. In questo caso, sono stati scelti 0.1 e 0.5 per esplorare l'effetto di diversi livelli di campionamento.
5. **min_samples_split**: Questo parametro definisce il numero minimo di campioni necessari per fare uno split in un nodo. Valori più elevati contribuiscono a rendere l'albero più semplice e riducono il rischio di overfitting, impedendo una divisione troppo fine. I valori scelti (2, 5) permettono di esplorare diverse granularità di suddivisione.
6. **min_samples_leaf**: Definisce il numero minimo di campioni per creare un nodo foglia. Scegliendo valori come 1 e 2, si testano alberi che non si adattano troppo ai dati di addestramento, migliorando la generalizzazione e riducendo l'overfitting.
7. **loss**: Questo parametro definisce la funzione di perdita da minimizzare durante il processo di addestramento. Il valore scelto è squared_error, che minimizza l'errore quadratico medio tra le predizioni del modello e i valori reali, un criterio ampiamente utilizzato per problemi di regressione.

Valutazione dei modelli



Il grafico mostra il confronto delle performance dei modelli di regressione in termini di **Mean Absolute Error** (MAE). Il MAE misura la differenza media assoluta tra i valori previsti dal modello e quelli reali, fornendo un'indicazione diretta dell'accuratezza delle previsioni. Un valore più basso indica una maggiore precisione del modello.

Dal grafico si osserva che la regressione lineare (**LinearRegression**) ha il MAE più alto, pari a **0.548833**, indicando una maggiore distanza tra le previsioni e i valori reali. Al contrario, l'**SVR** presenta il MAE più basso, **0.46428**, suggerendo una migliore capacità predittiva rispetto agli altri modelli. **RandomForestRegressor** e **GradientBoostingRegressor** mostrano valori molto simili, rispettivamente **0.475384** e **0.475231**, entrambi inferiori alla regressione lineare ma leggermente superiori rispetto a SVR. Questo suggerisce che i modelli basati su alberi offrono una buona capacità di generalizzazione, sebbene non superino SVR in termini di accuratezza media delle previsioni.

2. Modelli valutati.

	MAE	MSE	RMSE
LinearRegression	0.548833	0.690459	0.829792
SVR	0.46428	0.580292	0.760183
RandomForestRegressor	0.475384	0.570538	0.754046
GradientBoostingRegressor	0.475231	0.577451	0.758684

Dopo aver analizzato il MAE, è utile esaminare anche il **Mean Squared Error** (MSE) e il **Root Mean Squared Error** (RMSE), che penalizzano maggiormente gli errori più grandi rispetto al MAE, offrendo un'ulteriore prospettiva sulla precisione dei modelli. Il Mean Squared Error (MSE) misura la media dei quadrati degli errori tra le predizioni e i valori reali, enfatizzando gli errori maggiori e fornendo un'indicazione complessiva della precisione del modello. Il Root Mean Squared Error (RMSE), essendo la radice quadrata del MSE, riporta questo errore alle stesse unità della variabile target, facilitandone l'interpretazione.

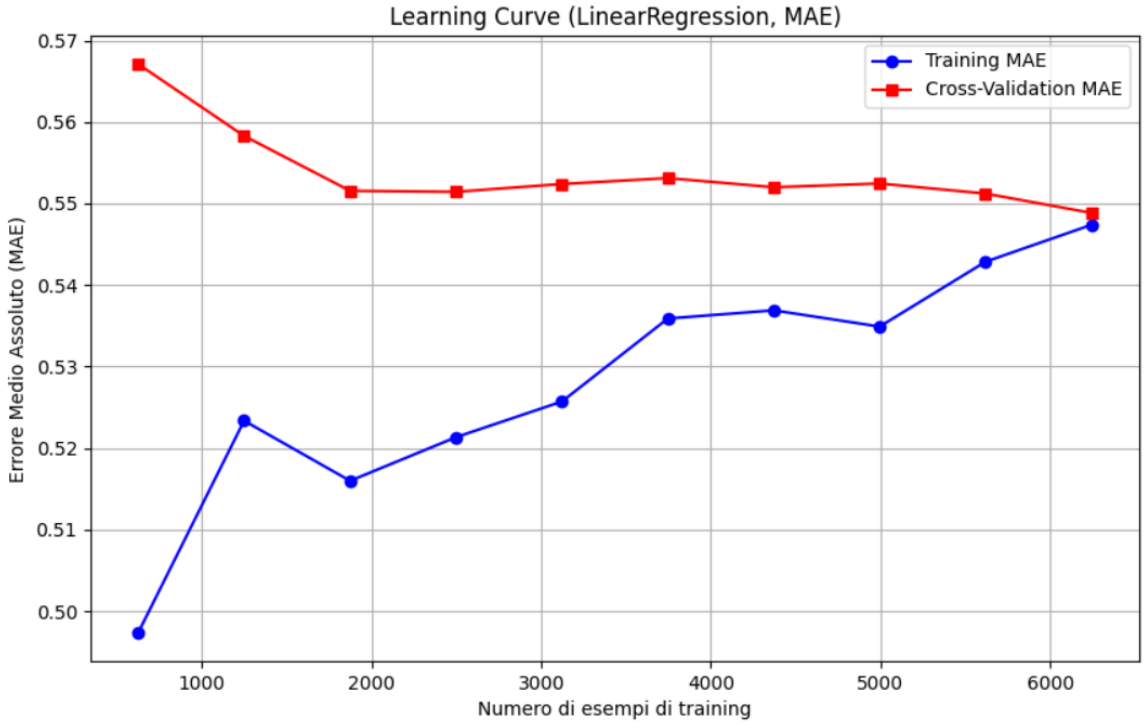
Dal confronto dei modelli emerge che **LinearRegression** presenta il valore di MSE più alto (**0.690459**), indicando una maggiore varianza negli errori e, di conseguenza, una minore accuratezza nelle previsioni rispetto agli altri modelli. Il suo RMSE (**0.829792**) conferma questa tendenza, suggerendo che gli errori medi sono relativamente elevati rispetto agli altri algoritmi.

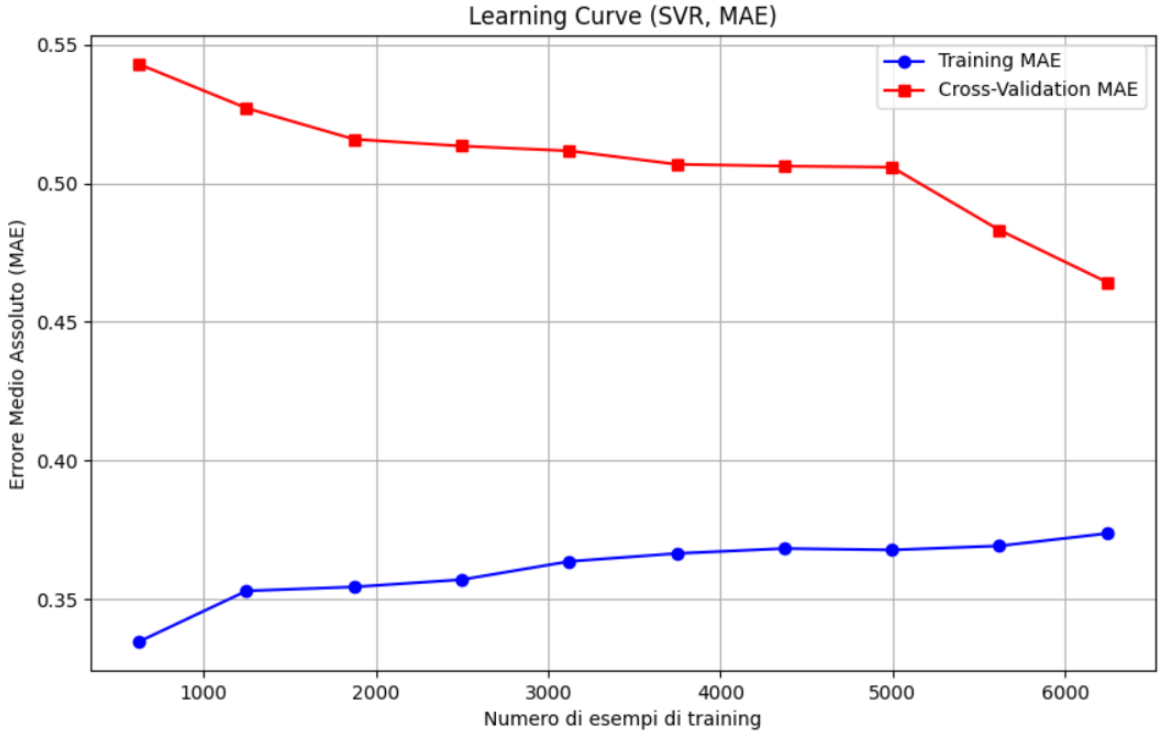
L'**SVR**, invece, ottiene un MSE inferiore (**0.580292**) rispetto alla regressione lineare, con un RMSE pari a **0.760183**, suggerendo una maggiore capacità di adattarsi ai dati e di ridurre gli errori più significativi. Tuttavia, non è il modello con il miglior MSE.

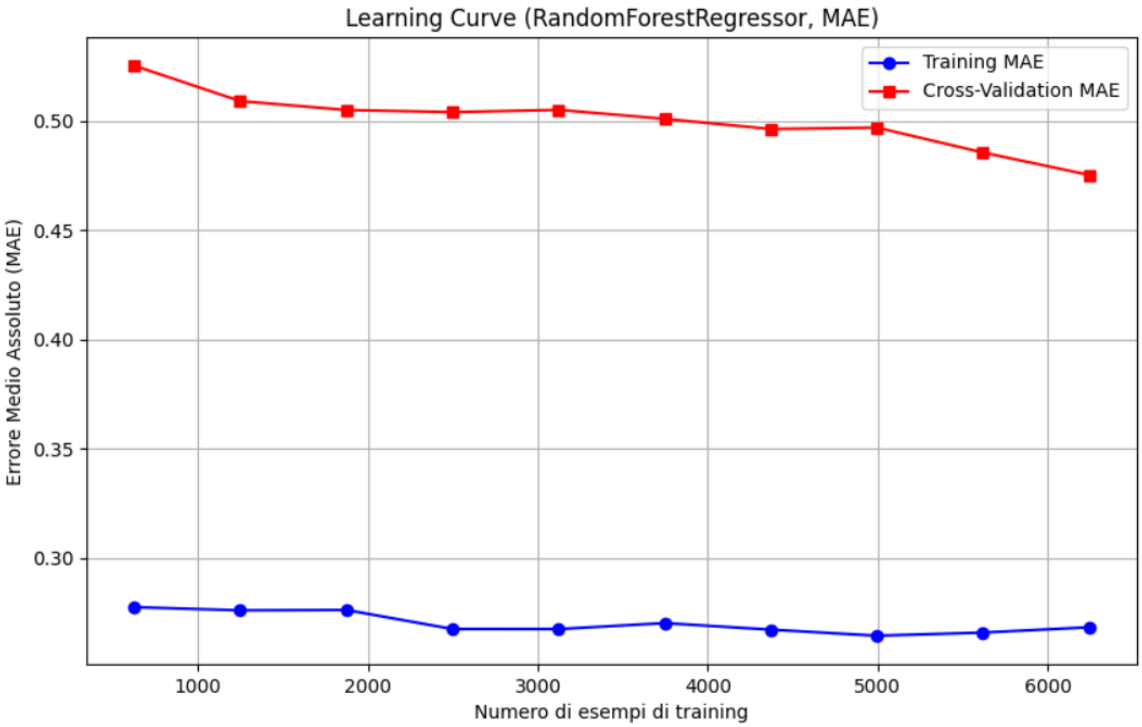
Il **RandomForestRegressor** si distingue per avere il MSE più basso tra tutti i modelli analizzati (**0.570538**), il che implica che le sue previsioni hanno errori complessivamente minori rispetto agli altri modelli. Il suo RMSE (**0.754046**) supporta questa osservazione, mostrando una buona accuratezza complessiva.

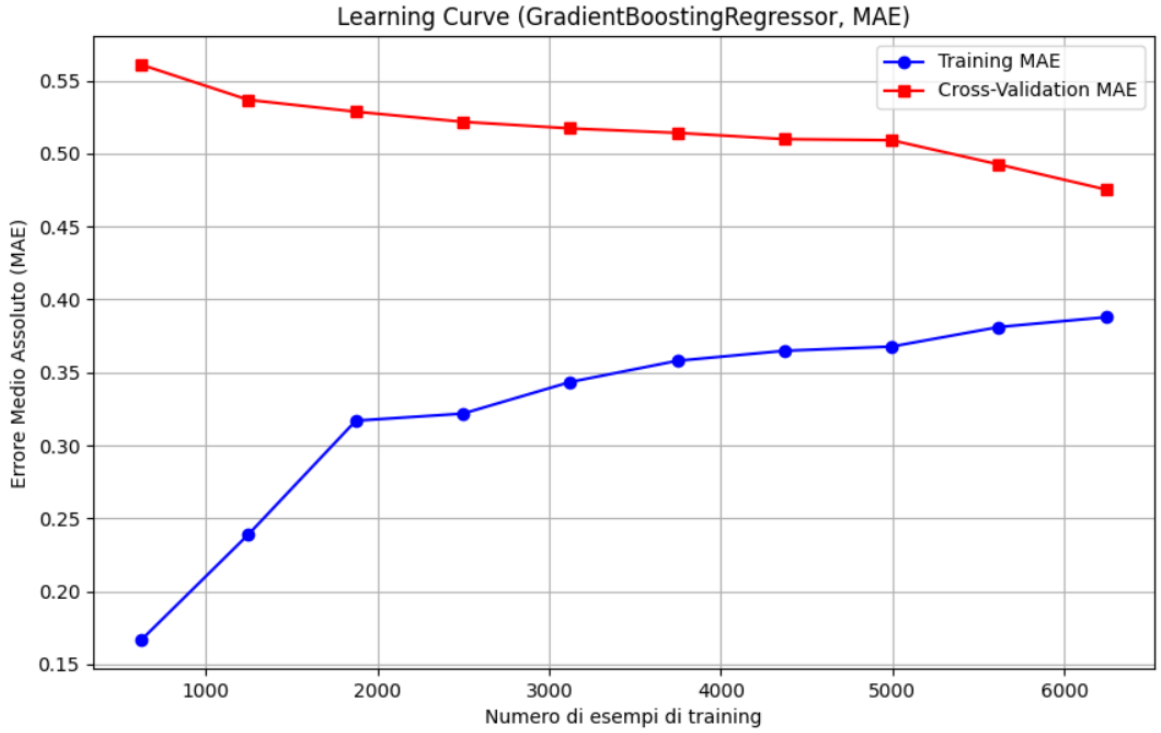
Infine, il **GradientBoostingRegressor** presenta un MSE leggermente superiore (**0.577451**) rispetto al RandomForestRegressor, con un RMSE di **0.758684**, confermando prestazioni simili tra i due modelli basati su alberi decisionali.

In sintesi, il confronto tra MSE e RMSE evidenzia che i modelli basati su alberi, in particolare **RandomForestRegressor**, offrono le previsioni più accurate, mentre la regressione lineare è il modello meno performante in termini di errori quadratici medi.




Learning Curve (SVR, MAE)
Training MAE
Cross-Validation MAE
Errore Medio Assoluto (MAE)
0.55
0.50
0.45
0.40
0.35
1000
2000
3000
4000
5000
6000
Numero di esempi di training


Learning Curve (RandomForestRegressor, MAE)
Training MAE
Cross-Validation MAE
Errore Medio Assoluto (MAE)
0.50
0.45
0.40
0.35
0.30
1000
2000
3000
4000
5000
6000
Numero di esempi di training



Nei grafici qui sopra, l'asse orizzontale rappresenta il numero di esempi di training utilizzati per addestrare i modelli, mentre l'asse verticale indica l'errore medio assoluto. In entrambe le curve, la linea blu corrisponde all'errore di training (Training MAE), mentre la linea rossa rappresenta l'errore di convalida incrociata (Cross-Validation MAE).

Nel grafico del **LinearRegression**, analizzando il comportamento del training MAE, si osserva che inizialmente l'errore è basso, poiché il modello si adatta molto bene ai pochi esempi disponibili. Tuttavia, man mano che il numero di dati di training aumenta, l'errore cresce progressivamente, suggerendo che il modello sta diventando meno specifico per quei pochi esempi iniziali e sta generalizzando maggiormente. Al contrario, la curva di validazione inizia con un valore elevato, segnalando un'elevata difficoltà del modello nel generalizzare quando ha pochi dati. Con l'aumentare del numero di esempi, il MAE di validazione diminuisce inizialmente e poi si stabilizza, indicando un miglioramento nella generalizzazione, ma con un limite strutturale dovuto alla semplicità della regressione lineare. L'andamento delle due curve suggerisce che il modello soffre di un bias elevato, ovvero una tendenza sistematica a produrre errori anche con un numero maggiore di dati. Sebbene la distanza tra il MAE di training e quello di validazione si riduca progressivamente, essa non si annulla completamente, segnalando che il modello non è in grado di catturare tutta la complessità del fenomeno analizzato. Questo fenomeno è tipico dei modelli lineari applicati a problemi con relazioni più complesse tra le variabili.

Per l'**SVR**, Il grafico mostra la learning curve del modello valutato in termini di Mean Absolute Error (MAE), illustrando il suo comportamento con l'aumentare del numero di dati di training. Si osserva che il MAE di training è costantemente basso e cresce solo leggermente con l'aumentare dei dati, suggerendo che il modello riesce ad adattarsi bene ai dati di addestramento senza un'eccessiva complessità. Il MAE di validazione parte da un valore piuttosto elevato ma diminuisce gradualmente man mano che si aggiungono più dati, indicando un miglioramento nella capacità di generalizzazione del modello. A differenza della regressione lineare, SVR mostra un divario più marcato tra training e validazione, il che suggerisce una

maggiore capacità di apprendere strutture più complesse nei dati. Tuttavia, il fatto che la curva di validazione non converga completamente verso quella di training potrebbe indicare un leggero overfitting, sebbene il modello sembri comunque migliorare con più dati. Nel complesso, il comportamento di SVR suggerisce che il modello è più efficace nel catturare le relazioni tra le variabili rispetto alla regressione lineare, ottenendo un errore di previsione inferiore e una migliore capacità di generalizzazione.

Per il **RandomForestRegressor**, un modello di tipo ensemble che addestra più alberi di decisione e ne media le predizioni, l'errore sul training si mantiene costantemente basso, segno che il modello si adatta molto bene ai dati che vede durante l'addestramento. Tuttavia, l'errore di validazione, pur riducendosi leggermente con l'aumentare dei dati, rimane nettamente più alto rispetto a quello di training. Questo divario indica un certo grado di overfitting, poiché il RandomForestRegressor tende a specializzarsi troppo sui dati di addestramento, a discapito della generalizzazione. Per mitigare questo problema, si possono applicare tecniche di regolarizzazione, come limitare la profondità massima degli alberi o aumentare il numero di alberi, in modo da ridurre la varianza e migliorare ulteriormente la capacità predittiva su dati non visti.

Nel grafico del **GradientBoostingRegressor**, un metodo di ensemble che costruisce progressivamente nuovi "weak learners" per correggere gli errori dei precedenti, si osserva un andamento iniziale in cui l'errore sul training risulta piuttosto basso, perché il modello riesce a adattarsi bene a un numero ridotto di esempi. Con l'aumentare dei dati, l'errore di training tende a crescere, indicando che il modello fatica a mantenere lo stesso livello di specializzazione. Nel frattempo, l'errore di validazione diminuisce gradualmente, segnalando un miglioramento nella capacità di generalizzare. Tuttavia, il divario tra l'errore di training e quello di validazione resta evidente, suggerendo che il modello potrebbe essere soggetto a un certo grado di overfitting. Per attenuare questo fenomeno, si possono regolare parametri come la profondità massima degli alberi, il numero di stadi di boosting o il learning rate. La diminuzione dell'errore di validazione, comunque, indica che con un numero sufficiente di dati e una calibrazione accurata degli iperparametri, il GradientBoostingRegressor è in grado di fornire prestazioni predittive solide.

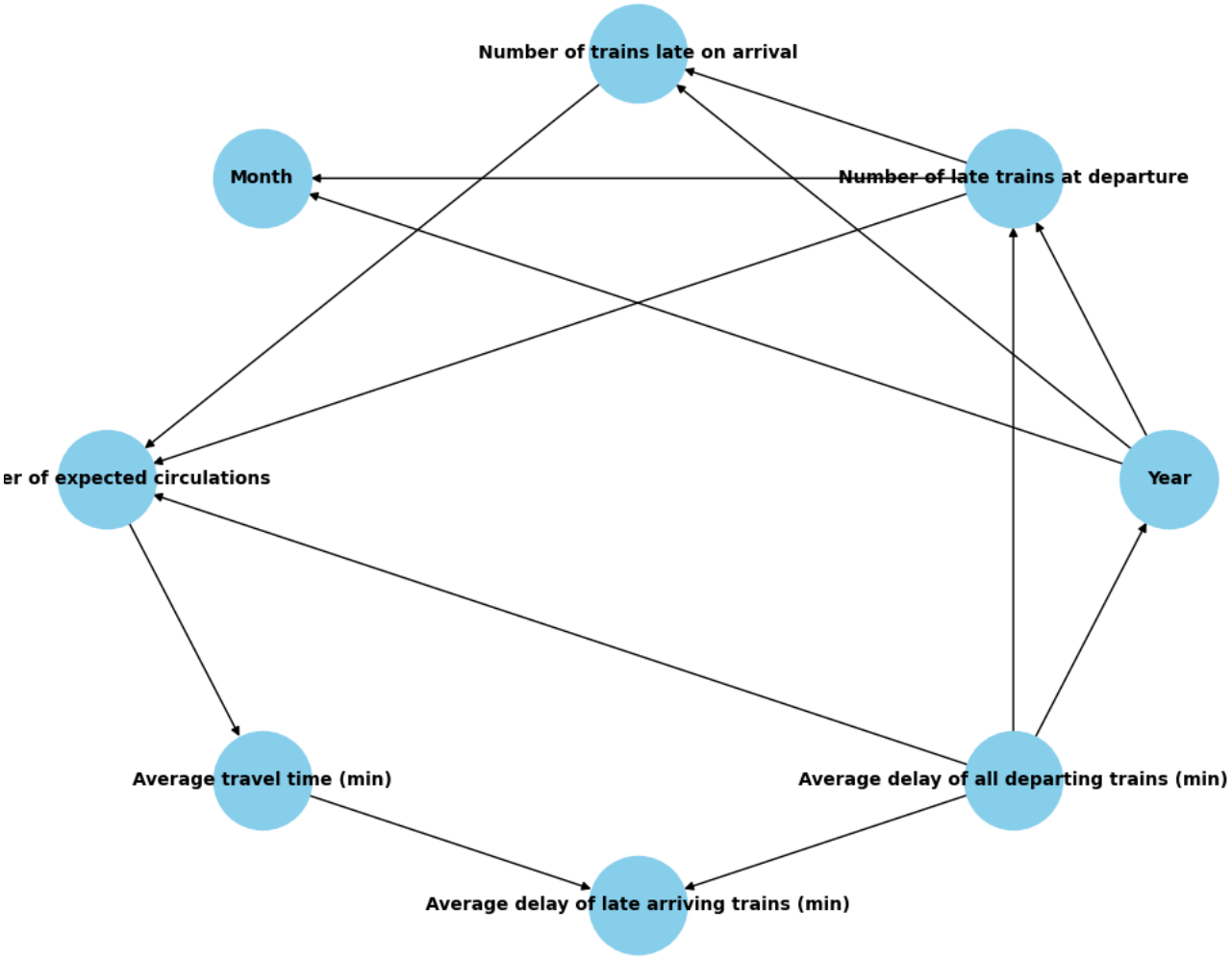
Infine, tutti i grafici generati verranno salvati nel percorso:
"Trasporto pubblico\documentazione\res\drawable\img_supervised" al momento dell'esecuzione del codice sorgente.

Apprendimento bayesiano

L'**apprendimento bayesiano** è un approccio all'inferenza che si basa sul teorema di Bayes per aggiornare le probabilità di ipotesi in presenza di nuove evidenze. Esso consente di modellare la relazione tra variabili tramite reti bayesiane. Il sistema calcola la probabilità a posteriori di ogni ipotesi, combinando la probabilità a priori e la probabilità condizionata osservata nei dati. Questo metodo è particolarmente utile quando si deve gestire l'incertezza e si ha a che fare con dati rumorosi o incompleti. Viene impiegato in diversi ambiti, come la diagnostica medica, la previsione del rischio e l'analisi di sistemi complessi, dove è importante valutare le probabilità in presenza di molteplici fattori interconnessi.

Quando si applica l'apprendimento bayesiano a dati continui, è comune procedere alla discretizzazione, già effettuata nel pre-processing, ovvero la trasformazione delle variabili continue in categorie o intervalli. Questo processo, spesso basato sui quantili, permette di ottenere una rappresentazione bilanciata dei dati, semplificando il calcolo delle probabilità condizionali e rendendo il modello più gestibile dal punto di vista computazionale.

Struttura della rete bayesiana



Per creare il modello bayesiano, è stato utilizzato l'algoritmo **HillClimbSearch**, un metodo di ricerca locale che esplora diverse configurazioni per individuare la struttura ottimale delle relazioni tra le variabili del dataset. Il metodo di scoring scelto è il **BicScore**, che valuta la bontà del modello tenendo conto della complessità e dell'adattamento ai dati.

Una volta identificata la struttura migliore (contenuta in *best_model*), viene estratto l'insieme degli archi che rappresentano le connessioni tra le variabili. Questi archi vengono poi utilizzati per definire la rete bayesiana, mediante la funzione *BayesianNetwork(best_model.edges())*. In questo modo, il modello risultante riflette una configurazione ottimizzata che descrive le dipendenze condizionali presenti nel dataset.

La struttura di una rete bayesiana è un grafo orientato aciclico, dove ogni nodo rappresenta una variabile casuale e le frecce indicano le dipendenze condizionali. Ogni nodo è associato a una tabella di probabilità condizionata, che descrive come la variabile dipende dai suoi genitori. Questa struttura consente di modellare e inferire probabilità in sistemi complessi, aggiornando le credenze con nuove evidenze.

Addestramento della rete bayesiana

Il processo di addestramento si basa sulla struttura della rete bayesiana definita in precedenza, in cui vengono costruite le **CPD (Conditional Probability Distributions)**. Queste distribuzioni descrivono la probabilità di ogni variabile in relazione alle sue variabili genitore, consentendo di modellare le dipendenze condizionali tra le variabili nel dataset.

Per creare le **CPD** nella rete bayesiana, il processo dipende dalla presenza di genitori per ogni variabile. Se una variabile non ha genitori, viene creata una **CPD marginale**, che rappresenta la probabilità della variabile basata sulla sua distribuzione nel dataset. Le probabilità vengono normalizzate e, se necessario, viene utilizzata una distribuzione uniforme per evitare che la somma delle probabilità sia zero. Nel caso in cui la variabile abbia dei genitori, viene costruita una **CPD condizionata**. In questo caso, si calcola la distribuzione della variabile dato un particolare insieme di stati dei genitori, creando una tabella che rappresenta queste probabilità condizionali. Anche in questo caso, la normalizzazione è fondamentale per garantire che la somma delle probabilità sia pari a uno, e se la somma è zero, si utilizza una distribuzione uniforme come fallback.

Il risultato finale consiste in un insieme di **CPD** che rappresentano le distribuzioni di probabilità per ciascun nodo, prendendo in considerazione le dipendenze condizionali tra le variabili. Queste CPD vengono quindi aggiunte al modello, completando la definizione della rete bayesiana.

Utilizzo della rete bayesiana

Una volta creata e definita la rete bayesiana, è possibile utilizzarla per effettuare inferenze, ossia per calcolare non solo lo stato più probabile, ma l’intera distribuzione di probabilità di una variabile target, dato un insieme di evidenze. In altre parole, si può determinare come cambia la probabilità di ciascuno stato discretizzato della variabile target in base ai valori osservati di altre variabili nel modello. Questo processo non si limita a prevedere un singolo evento futuro, ma quantifica l’incertezza associata a tutti i possibili stati del target, basandosi sulle informazioni disponibili.

L’inferenza viene eseguita utilizzando il metodo **Variable Elimination**, che calcola la distribuzione a posteriori esatta per la variabile target (ad esempio, il numero di treni in ritardo), considerando un

esempio specifico (una riga del dataset). Le evidenze per l'inferenza vengono estratte dal dataset, escludendo la variabile target, e vengono utilizzate per generare le probabilità di tutti gli stati discretizzati del target, non solo dello stato massimo. A differenza di approcci euristici, il Variable Elimination garantisce il calcolo preciso di *P(target = bin_i | evidenze)* per ogni intervallo definito.

Poiché nel pre-processing è stato impostato un numero di bins pari a 4, ogni variabile numerica è stata suddivisa in 4 intervalli durante la discretizzazione. Questo vale anche per la variabile target, come nel caso del '*Number of trains late on arrival*', che assume 4 possibili stati (0, 1, 2, 3). Il modello non assegna semplicemente un bin alla riga analizzata, ma calcola una probabilità specifica per ciascuno dei 4 stati, riflettendo la complessità delle relazioni tra le variabili.

Come mostrato nel grafico qui sopra, la distribuzione a posteriori per la riga selezionata (row_index=0) evidenzia la probabilità associata a ciascun valore discretizzato della variabile target, in questo caso '*Number of trains late on arrival*'. Sull'asse orizzontale sono rappresentati gli stati 0-3, che corrispondono ai quattro intervalli (bins) in cui è stata suddivisa la variabile target durante la fase di discretizzazione. Sull'asse verticale, invece, è riportata la probabilità esatta calcolata dal Variable Elimination per ogni stato. Le altezze delle barre non derivano da una stima approssimata, ma dall'output diretto del metodo di inferenza, che combina le probabilità condizionate delle variabili genitore. Ciò significa che, data la configurazione dei dati di input per la riga 0, la rete calcola in che misura è plausibile che il numero di treni in ritardo all'arrivo appartenga a uno dei quattro intervalli definiti. Questo approccio è particolarmente utile in contesti decisionali: anziché fornire una stima puntuale (es. "il bin 3 è il più probabile"), la rete bayesiana mostra l'intera distribuzione di probabilità, permettendo di valutare:

- La plausibilità relativa di bin adiacenti (es. 48% per il bin 2 vs 49% per il bin 3),
- Il rischio cumulativo (es. probabilità che il target superi una soglia critica),
- L’incertezza residua, fondamentale per scenari in cui anche probabilità non dominanti richiedono attenzione.

Infine, tutti i grafici generati verranno salvati nel percorso:
"Trasporto pubblico\documentazione\res\drawable\img_bayesian" al momento dell'esecuzione del codice sorgente.

Prolog e KB

Come ultimo task si è deciso di predire la stessa variabile target dell'apprendimento supervisionato utilizzando una **Base di Conoscenza** (KB) in **Prolog**. L'obiettivo è trasporre la struttura dell'albero di decisione ottenuto dalla **Random Forest** in un insieme di fatti e regole logiche, in modo da consentire la classificazione dei dati tramite inferenza logica.

L'albero di decisione è costituito da **nodi intermedi** e **nodi foglia**. I nodi intermedi contengono quattro informazioni essenziali, che vengono memorizzate in array:

- **feature**: contiene gli **indici delle feature** utilizzate per le decisioni nei nodi.
- **threshold**: memorizza le **soglie di confronto** utilizzate nei nodi decisionali.
- **left_child** e **right_child**: contengono gli **ID dei nodi figli** per ciascun nodo intermedio.

Quando si valuta un'istanza, il valore della feature viene confrontato con la soglia:

- Se il valore è **minore o uguale** alla soglia, si prosegue verso il **figlio sinistro**.
- Se il valore è **maggiore**, si segue il **figlio destro**.

Questo processo continua fino a raggiungere un **nodo foglia**, che rappresenta la previsione finale del modello. Le foglie sono caratterizzate da un **array value**, che contiene la distribuzione dei valori di output per ciascuna classe, permettendo così di determinare la previsione finale.

In **Prolog**, la struttura dell'albero viene rappresentata tramite i fatti:

- **node(NodeID, FeatureIndex, Threshold, LeftChild, RightChild)** per i nodi decisionali, dove **NodeID** è l'identificatore univoco del nodo all'interno dell'albero.
- **leaf(NodeID, Prediction)** per i nodi foglia, in cui **NodeID** identifica il nodo terminale che contiene la previsione finale.

Per facilitare questo processo, è stata implementata in Python una funzione che converte l'albero di decisione ottenuto da Scikit-learn in fatti Prolog. La funzione scorre l'albero, estraendo dai nodi intermedi le informazioni relative alla feature, alla soglia e agli ID dei figli, e per le foglie il valore predetto. In questo modo, l'intera struttura viene esportata in un file, presente nel percorso "**Trasporto pubblico\random_forest_facts.pl**", rendendo possibile l'inferenza logica in Prolog.

Per predire il ritardo di un treno è stata definita la regola Prolog predire_ritardo, che funge da punto di ingresso per l'inferenza nel modello.

```
% Predizione del ritardo
predire_ritardo(Features, Ritardo) :-
    percorri_albero(0, Features, Ritardo).
```

Questa regola avvia il processo decisionale chiamando la regola ricorsiva percorri_albero, il quale parte dalla radice (nodo 0) dell'albero e guida la navigazione attraverso i nodi dell'albero tradotto in Prolog fino a raggiungere una foglia.

```
% Passo base (nodo foglia)
percorri_albero(NodeID, _, Predizione) :-
    leaf(NodeID, Predizione).

% Passo ricorsivo (nodo intermedio)
percorri_albero(NodeID, Features, Predizione) :-
    node(NodeID, FeatureIndex, Threshold, LeftChild, RightChild),
    nth0(FeatureIndex, Features, FeatureValue),
    (FeatureValue =< Threshold ->
        percorri_albero(LeftChild, Features, Predizione)
    ;
        percorri_albero(RightChild, Features, Predizione)
    ).
```

Nei nodi intermedi, viene valutata la condizione relativa a una specifica feature: il valore dell'attributo in input viene confrontato con una soglia. Se il valore risulta minore o uguale alla soglia, il percorso procede verso il figlio sinistro; altrimenti, verso il figlio destro. Questo iterativo confronto continua finché non si giunge a un nodo foglia, che contiene la predizione finale.

Oltre a predire il ritardo per un singolo treno, sono state definite ulteriori regole per estendere l'inferenza.

Ad esempio, la regola ricorsiva treno_piu_in_ritardo consente di identificare, tra una lista di treni, quello con il ritardo maggiore, confrontando ricorsivamente la predizione di ciascun treno.

```
% Passo ricorsivo: Trova il treno con il ritardo massimo in una lista.
treno_piu_in_ritardo([Features|AltriTreni], Treno, Ritardo) :-
    predire_ritardo(Features, RitardoCorrente),
    treno_piu_in_ritardo(AltriTreni, TempTrain, TempDelay),
    (RitardoCorrente > TempDelay ->
        Treno = Features, Ritardo = RitardoCorrente
    ;
        Treno = TempTrain, Ritardo = TempDelay
    ).

% Passo base: se c e' un solo treno, restituisci le sue caratteristiche e il ritardo predetto.
treno_piu_in_ritardo([Features], Features, Ritardo) :-
    predire_ritardo(Features, Ritardo).
```

La regola ritardo_conveniente verifica se un ritardo proposto è inferiore a quello predetto dal modello, offrendo un criterio per valutare la convenienza di un ritardo stimato.

```
% Predizione del ritardo piu' conveniente
ritardo_conveniente(Features, RitardoProposto, RitardoPredetto, Conveniente) :-
    predire_ritardo(Features, RitardoPredetto),
    (RitardoProposto < RitardoPredetto ->
        Conveniente = true
    ;
        Conveniente = false).
```

Le regole citate sono presenti nel percorso "**Trasporto pubblico\model_delay_rules.pl**".

Per esempio, il sistema basato su Prolog ha permesso di ottenere una predizione del ritardo in linea con quella del modello Random Forest in Python considerando due treni di test presi dal dataset.

```
Date le features dei treni di esempio:
[2019.0, 7.0, 247.0, 0.0, 191.0, 131.914979757, 3.5763525305400012, 2.67827260459, 0.25, 0.15, 0.275, 0.175, 25.0, 15.0, 27.500000000000004]
[2019.0, 7.0, 242.0, 0.0, 178.0, 175.611570248, 9.78080524345, 7.03360881543, 0.2, 0.244444444444, 0.266666666667, 0.0444444444443999, 20.0, 24.4444444444, 26.6666666667]
```

In pratica, utilizzando la regola predire_ritardo, il KB ha determinato che per il primo treno (df[0]) il ritardo previsto è di 21 minuti e 59 secondi, mentre per il secondo treno (df[1]) il ritardo risulta essere di 28 minuti e 48 secondi. Questi valori corrispondono esattamente alle previsioni ottenute con Scikit-learn, come evidenziato dalle rappresentazioni dei vettori delle feature associati a ciascun treno.

```
Prolog prediction:
df[0] -> 21m 59s
df[1] -> 28m 48s

Scikit prediction:
df[0] -> 21m 59s
df[1] -> 28m 48s
```

Questo esempio dimostra la coerenza tra l'approccio supervisionato implementato in Python e il sistema basato su una Base di Conoscenza in Prolog, evidenziando come entrambi gli approcci possano integrarsi per fornire predizioni.

Inoltre, delle due regole supplementari, la regola treno_piu_in_ritardo ha consentito di identificare il treno con il ritardo massimo, confermando che il secondo treno, con un ritardo di 28 minuti e 48 secondi, è quello che presenta la predizione più elevata.

La regola ritardo_conveniente, invece, è stata utilizzata per valutare se un ritardo proposto sia inferiore a quello stimato dal modello. Per esempio, per il primo treno (con una predizione di 21 minuti e 59 secondi), il sistema ha restituito il messaggio "Il ritardo proposto 20 è più conveniente di quello predetto", poiché il valore proposto (20) risulta inferiore al ritardo stimato.

```
Treno con il ritardo massimo 28m 48s

Il ritardo proposto 20 è più conveniente di quello predetto (21m 59s ).
```

Inoltre, è stata sviluppata una base di conoscenza in Prolog per stimare il ritardo di un treno a partire dalle sue caratteristiche. Il sistema analizza una serie di attributi che influenzano il ritardo, assegnando a ciascuno un contributo specifico che viene combinato per ottenere una previsione complessiva. Questo approccio permette di effettuare valutazioni dettagliate sui ritardi, confrontare più treni e determinare la convenienza di un ritardo proposto rispetto a quello stimato.

```
% Passo ricorsivo: Prende in input una lista di features e salva man
% mano il ritardo piu' alto
treno_piu_in_ritardo([Features|AltriTreni], Treno, Ritardo) :-
    predire_ritardo(Features, RitardoCorrente),
    treno_piu_in_ritardo(AltriTreni, MaxTreno, MaxRitardo),
    (RitardoCorrente > MaxRitardo ->
     Treno = Features, Ritardo = RitardoCorrente
     ;
     Treno = MaxTreno, Ritardo = MaxRitardo).
```

La regola treno_piu_in_ritardo/3 prende in input una lista di treni (rappresentata come una lista di caratteristiche Features). Per ogni treno nella lista, calcola il ritardo tramite la regola predire_ritardo/2. Confronta poi il ritardo del treno corrente (RitardoCorrente) con il ritardo massimo trovato finora (MaxRitardo). Se il ritardo corrente è maggiore, allora il treno corrente diventa il treno con il ritardo massimo; altrimenti, si mantiene il treno con il ritardo massimo precedente.

```
% Passo base: se c'e' un solo treno, restituisci le sue caratteristiche
% e il ritardo predetto.
treno_piu_in_ritardo([Features], Features, Ritardo) :-
    predire_ritardo(Features, Ritardo).
```

Se la lista di treni contiene solo un treno, la regola calcola direttamente il ritardo di quel treno tramite la regola predire_ritardo/2 e restituisce le sue caratteristiche insieme al ritardo predetto.

```
% Predizione del ritardo piu' conveniente
ritardo_conveniente(Features, RitardoProposto, RitardoPredetto, Conveniente) :-
    predire_ritardo(Features, RitardoPredetto),
    (RitardoProposto < RitardoPredetto ->
        Conveniente = true
    ;
        Conveniente = false).
```

La regola ritardo_conveniente/4 determina se un ritardo proposto (RitardoProposto) risulti più conveniente rispetto a quello predetto dal modello, calcolato dalla regola predire_ritardo/2. Se il ritardo proposto è inferiore al ritardo predetto, il valore di Conveniente sarà true; in caso contrario, sarà false.

```
% Predizione del ritardo
predire_ritardo(Features, Ritardo) :-
    reverse(Features, FeaturesInvertite),
    calcolare_ritardo(FeaturesInvertite, 0, Ritardo).
```

La regola predire_ritardo/2 prende in input una lista di caratteristiche del treno (Features) e calcola il ritardo associato. Inverte prima la lista delle caratteristiche e poi calcola l'incremento del ritardo passo passo utilizzando la regola calcolare_ritardo/3. La variabile Ritardo rappresenta il ritardo finale predetto per il treno.

```
% Passo ricorsivo: Predicato per calcolare gli incrementi basati sulle caratteristiche
calcolare_ritardo([Feature|AltreFeature], RitardoParziale, Ritardo) :-
    length(AltreFeature, Indice),
    incremento_feature(Indice, Feature, Incremento),
    NuovoRitardoParziale is RitardoParziale + Incremento,
    calcolare_ritardo(AltreFeature, NuovoRitardoParziale, Ritardo).
```

La regola calcolare_ritardo/3 calcola il ritardo parziale per ogni caratteristica del treno. Prende la lista di caratteristiche (con Feature come la caratteristica corrente) e la calcola ricorsivamente. Per ogni caratteristica, chiama incremento_feature/3 per determinare l'incremento del ritardo. Il ritardo parziale accumulato viene aggiornato e la regola continua ricorsivamente per tutte le caratteristiche rimanenti.

```
% Passo base: Se non ci sono piu' feature da analizzare, restituisci il
% ritardo parziale
calcolare_ritardo([], RitardoParziale, RitardoParziale).
```

Quando la lista delle caratteristiche è vuota, la regola termina restituendo il ritardo parziale finale calcolato finora.

```
% Year (Anni piu' recenti -> meno ritardo)
incremento_feature(0, Anno, Incremento) :-
    Incremento is (2025 - Anno) * -0.5.

% Month (Inverno -> Piu' ritardo)
incremento_feature(1, Mese, Incremento) :-
    (Mese >= 11 ; Mese =< 2 -> Incremento is 5 ; Incremento is 0).

% Indice 4: Number of expected circulations
incremento_feature(2, Circolazioni, Incremento) :-
    Incremento is Circolazioni * 0.2.

% Number of cancelled trains (Piu' cancellazioni = Maggiore congestione
% e ritardo)
incremento_feature(3, Cancellati, Incremento) :-
    Incremento is Cancellati * 1.

% Number of late trains at departure (Piu' treni in ritardo alla
% partenza = Piu' ritardi in arrivo)
incremento_feature(4, NRitardiPartenza, Incremento) :-
    Incremento is NRitardiPartenza * 0.8.

% Average travel time (Piu' viaggio = Piu' probabilita' di ritardo)
incremento_feature(5, TempoMedio, Incremento) :-
    Incremento is TempoMedio * 0.1.

% Average delay of late departing trains (Aumenta direttamente il ritardo previsto)
incremento_feature(6, RitardoPartenza, Incremento) :-
    Incremento is RitardoPartenza * 0.9.

% Default per le feature non specificate
incremento_feature(_, _, 0).
```

Queste regole determinano l’incremento di ritardo associato a ciascuna caratteristica di un treno. Ogni chiamata a incremento_feature(Indice, ValoreCaratteristica, Incremento) calcola un contributo (Incremento) in base alla posizione (Indice) e al valore effettivo della caratteristica (ValoreCaratteristica). L’idea di fondo è che alcune variabili aumentino il ritardo, mentre altre possano ridurlo. Ogni caratteristica influisce sul ritardo in modo specifico:

1. **Anno (Year):** I treni degli anni più recenti tendono ad avere un ritardo inferiore. L'incremento è calcolato come la differenza tra l'anno corrente (2025) e l'anno, moltiplicato per -0.5. Treni più recenti quindi avranno un ritardo minore.
2. **Mese (Month):** I mesi invernali (novembre, dicembre, gennaio, febbraio) sono associati a un aumento dei ritardi. Se il mese è invernale, l'incremento del ritardo è 5 minuti; altrimenti, non ci sono incrementi.
3. **Numero di circolazioni attese (Number of expected circulations):** Se Indice è 2, la caratteristica indica il numero di circolazioni previste. Un valore più elevato di circolazioni aumenta il ritardo di 0.2 minuti per ogni circolazione, ipotizzando che un traffico più intenso possa comportare maggiori rallentamenti.
4. **Numero di treni cancellati (Number of cancelled trains):** Un numero maggiore di treni cancellati indica maggiore congestione, aumentando il ritardo. Ogni treno cancellato contribuisce con un incremento di 1 minuto.
5. **Numero di treni in ritardo alla partenza (Number of late trains at departure):** Un numero maggiore di treni in ritardo alla partenza è correlato a un aumento del ritardo in arrivo. Ogni treno in ritardo alla partenza contribuisce con un incremento di 0.8 minuti.
6. **Tempo medio di viaggio (Average travel time):** Più lungo è il viaggio medio di un treno, maggiore sarà il ritardo previsto. L'incremento è dato dal tempo medio di viaggio moltiplicato per 0.1.
7. **Ritardo medio dei treni in ritardo alla partenza (Average delay of late departing trains):** Un aumento del ritardo medio dei treni in partenza porta direttamente a un incremento nel ritardo previsto. L'incremento è calcolato come il ritardo di partenza moltiplicato per 0.9.

Infine, per le caratteristiche non specificate, viene restituito un incremento di 0 minuti, indicante che non c'è alcun impatto sul ritardo per quelle caratteristiche.

Tali regole sono presenti nel percorso "**Trasporto pubblico\trains_delay_rules.pl**".

La base di conoscenza può dunque essere interrogata in Python utilizzando la libreria pyswip con una qualsiasi delle righe del dataset pre-processato.

Considerando questi esempi presi dal dataset:

```
Date le features dei treni di esempio:
[2019.0, 7.0, 435.0, 5.0, 391.0, 62.39534883719999, 3.8969735720400003, 3.52934108527, 0.1617647058819999, 0.323529411765, 0.264705882353, 0.07352941176469
99, 16.1764705882, 32.3529411765, 26.4705882353000003]
[2019.0, 7.0, 114.0, 0.0, 101.0, 172.421052632, 1.9509900990099993, 1.6856725146200002, 0.153846153846, 0.153846153846, 0.230769230769, 0.0769230769231, 15
.3846153846, 15.3846153846, 23.0769230769]
[2019.0, 7.0, 404.0, 4.0, 284.0, 67.31, 8.37910798122, 5.803125, 0.184615384615, 0.123076923077, 0.4, 0.0461538461538, 18.4615384615, 12.3076923077, 40.0]
```

I risultati ottenuti dall'inferenza Prolog evidenziano le seguenti previsioni: per la riga df[2] il sistema ha stimato un ritardo di 26 minuti e 20 secondi, mentre per df[3] è stato calcolato un anticipo di 23 minuti e 34 secondi e per df[4] un ritardo di 14 minuti e 18 secondi.

Il sistema Prolog ha identificato il treno con il ritardo massimo stimato di 26 minuti e 20 secondi, confrontando le previsioni su 3 esempi analizzati. Inoltre, è stato valutato un ritardo proposto di 20 minuti, risultando più conveniente rispetto a quello predetto. Questo approccio consente di determinare se un ritardo suggerito possa essere accettabile rispetto alla stima ottenuta, supportando così decisioni più informate nella gestione dei ritardi.

```
Prolog prediction:
df[2] -> 26m 20s
df[3] -> 23m 34s in anticipo
df[4] -> 14m 18s

Treno con il ritardo massimo 26m 20s

Il ritardo proposto 20 è più conveniente di quello predetto (26m 20s ).
```

Conclusioni

Il progetto ha avuto come obiettivo principale la previsione dei ritardi dei treni utilizzando diverse tecniche di machine learning e approcci logici. In particolare, è stato esplorato l'apprendimento supervisionato, l'apprendimento bayesiano e l'integrazione con una Base di Conoscenza (KB) in Prolog, al fine di creare un sistema completo in grado di predire i ritardi dei treni e confrontare le performance dei vari modelli utilizzati.

Inizialmente, è stato implementato un task di apprendimento supervisionato basato su tecniche di regressione come SVR, Random Forest, Gradient Boosting e Regressione Lineare. I risultati ottenuti attraverso l'analisi dei metriche – MAE, MSE e RMSE – e delle learning curve hanno evidenziato differenze rilevanti nelle capacità predittive dei vari modelli. La Regressione Lineare, caratterizzata da un bias elevato, ha prodotto errori medi più alti, segnalando una limitata capacità di catturare relazioni complesse. Al contrario, l'SVR ha mostrato il MAE più basso, dimostrando una maggiore precisione nelle previsioni. I modelli basati su ensemble, come Random Forest e Gradient Boosting, hanno offerto prestazioni solide, in particolare in termini di MSE e RMSE, sebbene presentassero un certo grado di overfitting. In sintesi, la scelta del modello ideale in un contesto di apprendimento supervisionato deve bilanciare la complessità dei dati e la capacità di generalizzazione, con SVR e gli ensemble che emergono come opzioni particolarmente performanti per catturare relazioni non lineari.

In parallelo, il progetto ha esplorato l'uso di un approccio di **apprendimento bayesiano**, che sfrutta il **teorema di Bayes** per aggiornare le probabilità in base alle evidenze osservate. Questo approccio ha permesso di modellare le relazioni tra le variabili in modo probabilistico, fornendo una visione alternativa alla previsione dei ritardi. L'uso delle **reti bayesiane** ha contribuito ad analizzare i dati in presenza di incertezze e a fare inferenze basate su dati incompleti, arricchendo ulteriormente il sistema di previsione.

Infine, è stato integrato un sistema basato su Prolog per la previsione dei ritardi, trasformando l'albero di decisione della Random Forest in una serie di fatti e regole logiche. Questo approccio ha dimostrato di essere efficace nel generare previsioni compatibili con il modello di apprendimento supervisionato, mostrando come l'inferenza logica possa essere utilizzata per eseguire classificazioni. La regola "predire_ritardo" ha permesso di determinare il ritardo previsto per ciascun treno, mentre altre regole logiche hanno consentito l'identificazione dei treni più in ritardo e la valutazione della convenienza di ritardi proposti.

Inoltre, la seconda parte del sistema Prolog calcola direttamente il ritardo di un treno partendo dalle sue caratteristiche. Viene elaborata una lista di attributi – come l'anno, il mese, il numero di circolazioni, le cancellazioni, i ritardi alla partenza, il tempo di viaggio, ecc. – e a ciascuno di questi viene assegnato un contributo specifico, che sommato agli altri fornisce una stima complessiva del ritardo. Il sistema consente anche di confrontare i ritardi di diversi treni per individuare quello più elevato e di verificare se un ritardo proposto sia inferiore a quello stimato, offrendo così un criterio di convenienza.

Possibili sviluppi

Implementare una UI per visualizzare le predizioni dei ritardi in modo più intuitivo, permettendo all'utente di inserire dati e ottenere risultati in tempo reale.

Si potrebbe arricchire la base di conoscenza Prolog includendo nuove variabili o regole aggiuntive per migliorare la precisione delle predizioni.

Attualmente sono stati utilizzati Regressione Lineare, Gradient Boosting, Random Forest e SVR per l'apprendimento supervisionato. Uno sviluppo futuro potrebbe consistere nel testare altri modelli, come reti neurali o modelli basati su serie temporali, per valutare se possano migliorare l'accuratezza delle predizioni.

Riferimenti bibliografici

Apprendimento supervisionato: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.7]

Apprendimento bayesiano: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.10]

Basi di conoscenza: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.15]