Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

New York City Taxi Trip Duration

Authors:
Massimo Trippetta - 869286- m.trippetta@campus.unimib.it
Lorenzo Megna - 868929- l.megna1@campus.unimib.it

January 24, 2025


UNIVERSITA' DEGLI STUDI DI MILANO
BICOCCA

Abstract

Questo report descrive un'analisi predittiva basata su dati relativi alla durata dei viaggi in taxi a New York City. L'obiettivo principale è sviluppare modelli di machine learning e deep learning per stimare con precisione la durata dei viaggi.
Viene anche effettuata un'analisi dettagliata delle variabili più significative per la predizione.

1 Introduction

L'analisi prevede l'implementazione e il confronto di due modelli principali: una serie di reti neurali costruita tramite TensorFlow e un regressore basato su XGBoost.

Lo scopo principale è determinare quale tra i due approcci offra le migliori prestazioni in termini di accuratezza predittiva dei dati analizzati.

2 Datasets

Il dataset è composto dalle seguenti variabili:

- **id**: un identificatore univoco per ogni viaggio.
- **vendor_id**: un codice che indica il provider associato al record del viaggio.
- **pickup_datetime**: data e ora in cui il tassametro è stato attivato.
- **passenger_count**: il numero di passeggeri nel veicolo (valore inserito dal conducente).
- **pickup_longitude**: la longitudine in cui il tassametro è stato attivato.
- **pickup_latitude**: la latitudine in cui il tassametro è stato attivato.
- **dropoff_longitude**: la longitudine in cui il tassametro è stato disattivato.
- **dropoff_latitude**: la latitudine in cui il tassametro è stato disattivato.

- **store_and_fwd_flag**: questa variabile indica se il record del viaggio è stato memorizzato nel veicolo prima di essere inviato al provider.
- **trip_duration**: durata del viaggio in secondi (variabile target).

2.1 Data Exploration

In questa fase andiamo a controllare lo schema delle colonne e generare statistiche descrittive. Cercare valori mancanti o outlier. Analizzare la distribuzione della variabile target, in questo caso, "trip_duration".

2.1.1 Heatmap

Osserviamo l'Heatmap per le variabili quantitative per vedere se ci sono correlazioni significative con la variabile target.

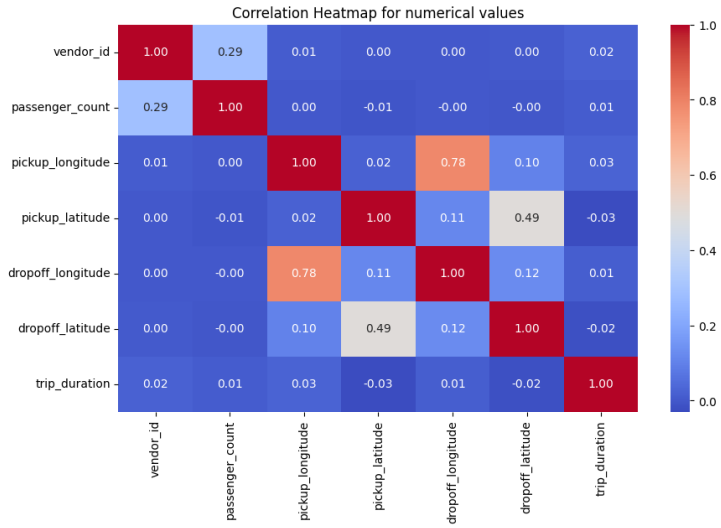

Figure 1: Heatmap

Come si può notare, sono tutte molto basse e rasentano lo zero.

2.1.2 Valori mancanti e outliers

Il dataset si presenta senza valori mancanti, ma con numerosi outliers che verranno gestiti come segue.

Per avere una visualizzazione più intuitiva, la variabile target è stata sottoposta ad un binning. Ecco come si presenta:

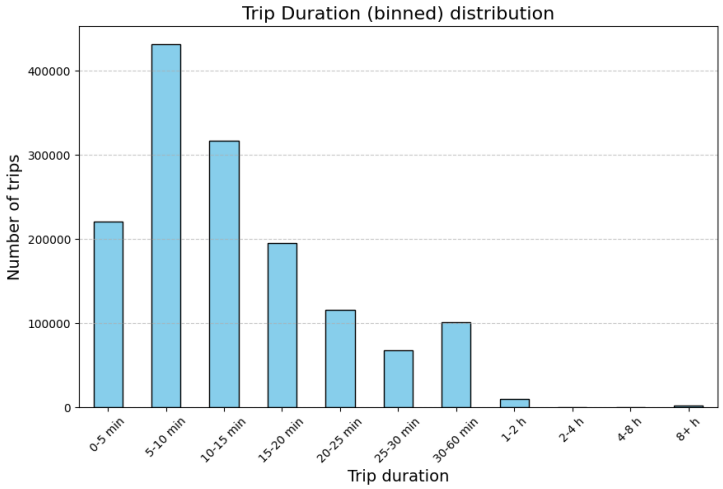

Figure 2: trip_duration dopo il binning.

Come si può notare, ci sono molti valori assurdamente alti. Si è quindi deciso di eliminare le istanze che uscissero dal seguente range:

$$[Q1 - 2 \cdot IQR\,;\, Q3 + 2 \cdot IQR] \tag{1}$$

ottenendo:

- **Lower bound**: -959.0
- **Upper bound**: 2431.0

Eliminando circa 47.548 istanze.

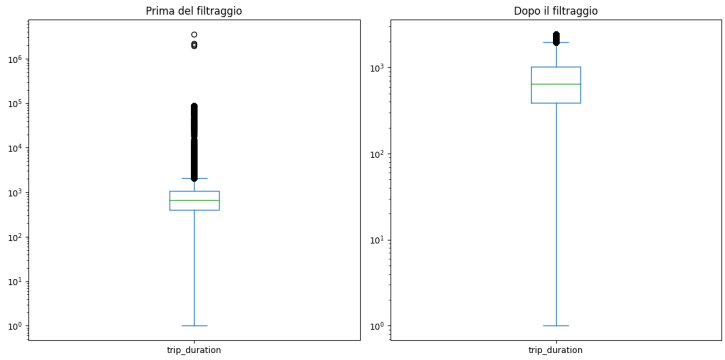

Figure 3: Box-plot dei valori di trip_duration.

Pare ovvio, a questo punto, che il lower bound non possa essere raggiunto nè tanto meno superato in quanto il tempo non può assumere valori negativi. E' però possibile notare che ci sono molte istanze che appartengono alle corse durate al massimo 5 minuti.
Andando a concentrarci su quelle e si scopre che ci sono corse che sono durate 1 o 2 minuti.

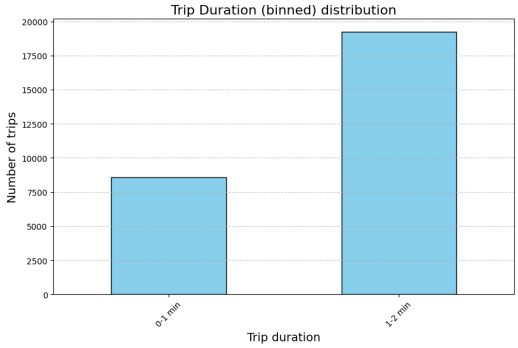

Figure 4: Binning per i primi 2 minuti della variabile trip_duration.

E' logico pensare che una corsa durata solo 2 minuti non sia molto realistica. E' stato quindi deciso di eliminare le istanze comprese in questo lasso di tempo.

Ecco come appare la distribuzione della variabile target alla fine di tutta la procedura:

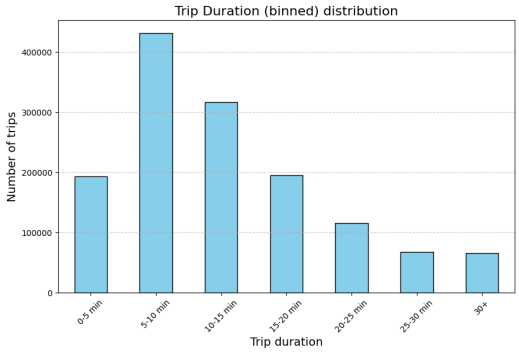

Figure 5: trip_duration dopo il filtraggio completo.

2.2 Feature engineering

Dato che le variabili a disposizione sono veramente poche e conferiscono poca informazione, è stato deciso di fare del **feature engineering** andando ad aggiungere e a modificare le seguenti variabili:

- **distance_km**: tramite la formula di **haversine** è stata calcolata la distanza in "km" utilizzando le coordinate.
- **bearing**: questo è l'angolo di *movimento*, ovvero, indica la direzione in cui si sono spostati i taxi. Anche questo dato è stato ottenuto tramite le coordinate.
- **._boroughs**: sia il punto di partenza che quello di arrivo sono stati trasformati, tramite le coordinate, nei vari distretti conosciuti come: *Manhattan, Brooklyn, Queens, Bronx e Staten Island*. Per quelli al di fuori di New York è stata asseganta l'etichetta *not_know*.
- **Data e ora**: per quanto riguarda questa varibile, sono state create delle sottovariabili che inidicano *anno, mese, giorno e giorno della settimana* (lunedì, martedì, ecc.).

2.3 Data preparation

Alla fine del Feature engineering, il dataset è finalmente pronto per essere usato come base di allenamento, ma non prima di essere stato sottoposto ad un encoding; infatti abbiamo effettuato un encoding (**one-hot**) su tutti i

valori stringa così da poter usare librerire come *keras* e *sci-kit*.

In fine l'intero dataset è stato suddiviso in 80% per il training, 10% per il validation e 10% per il test.

3 The Methodological Approach

Per affrontare il problema della predizione della durata del viaggio (*trip duration*), abbiamo inizialmente adottato metodologie basate su reti neurali. Questa scelta è stata motivata dalla capacità delle reti neurali di catturare pattern complessi e relazioni non lineari all'interno dei dati, risultando particolarmente adatte per problemi di regressione su dataset con molte feature. Per valutare le prestazioni di ogni modello sviluppato, sono state adottate due metriche principali:

- **Mean Absolute Error (MAE)**: rappresenta l'errore assoluto medio tra le predizioni del modello e i valori reali, fornendo una misura intuitiva e facilmente interpretabile della precisione del modello.
- **Root Mean Squared Logarithmic Error (RMSLE)**: utilizzata come funzione di perdita (*loss function*) per l'ottimizzazione, permette di penalizzare maggiormente le predizioni con errori proporzionalmente più grandi rispetto ai target reali. Questa scelta è particolarmente utile in problemi dove gli errori relativi sulle piccole durate devono essere controllati.

L'utilizzo congiunto di MAE e RMSLE garantisce una valutazione bilanciata, in grado di tenere conto sia dell'accuratezza generale che della proporzionalità degli errori predittivi.

3.1 Reti Neurali:

3.1.1 Primo modello

Il primo modello di rete neurale sviluppato è stato progettato per la regressione, utilizzando 5 strati densi progressivamente più piccoli (partendo da 256 andando a dimezzare ogni layer) per catturare i pattern complessi riducendo al contempo i costi computazionali. L'attivazione ReLU è stata utilizzata per

introdurre non-linearità, mentre l'attivazione lineare nello strato di output è risultata adeguata per il target continuo.
Per ottimizzare il processo di apprendimento, è stato utilizzato l'ottimizzatore Adam con un learning rate di 0.001, garantendo aggiornamenti dei pesi efficienti. Inoltre, per migliorare la stabilità dell'addestramento, sono stati adottati i seguenti approcci:

- **ReduceLROnPlateau**: diminuisce dinamicamente il learning rate se la perdita di validazione rimane stabile, favorendo una migliore convergenza.
- **EarlyStopping**: evita l'overfitting interrompendo l'addestramento quando non si osservano miglioramenti significativi nella perdita di validazione, ripristinando i pesi migliori del modello per la valutazione finale.
- **Epoche**: 30
- **Batch size**: 128
- **Learning rate**: 0,001 per una convergenza stabile.

Avremo alla fine **51,713 parametri**.

3.1.2 Secondo Modello

Il secondo modello presenta una modifica significativa rispetto al precedente: l'utilizzo della funzione di attivazione *tanh* nel primo strato denso, sostituendo la *ReLU*. La *tanh* scala i valori di output tra -1 e 1, consentendo al modello di apprendere contributi sia negativi che positivi, una caratteristica utile per determinate distribuzioni dei dati.
La restante architettura del modello, inclusi l'ottimizzatore, il learning rate, e i parametri per *ReduceLROnPlateau* ed *EarlyStopping*, rimangono invariati rispetto al primo modello. Il modello presenta un totale di **51,713 parametri** (202.00 KB), mantenendo una complessità computazionale moderata.

3.1.3 Terzo Modello

Il terzo modello introduce una complessità significativamente maggiore rispetto ai precedenti. Lo strato iniziale presenta 1024 neuroni, seguito da strati progressivamente più piccoli (512, 256, ecc.), aumentando la capacità del modello

di apprendere pattern complessi. Questa configurazione conta un totale di **7 strati**, rispetto ai 5 dei modelli precedenti, consentendo al modello di catturare relazioni più complesse nei dati.
La funzione di attivazione *tanh* è mantenuta nel primo strato per apprendere contributi negativi e positivi, mentre i restanti strati utilizzano la *ReLU*, preservando la non linearità e prevenendo il problema del *vanishing gradient*. Il modello ha un totale di **796,929 parametri** (3.04 MB), aumentando significativamente la capacità del modello rispetto al secondo.

3.1.4 Quarto Modello

Il quarto modello introduce una complessità significativamente maggiore rispetto a tutti i modelli precedenti. Lo strato iniziale presenta 2048 neuroni, seguito da strati di 1024, 512, 256 neuroni, con alcune ripetizioni di strati di dimensioni simili (ad esempio, due strati da 1024 e due da 512 neuroni). Questo aumenta notevolmente il numero totale di parametri, migliorando la capacità del modello di apprendere pattern estremamente complessi.
Con un totale di **9 strati**, questo è il modello più profondo, in grado di modellare relazioni intricate nei dati, ma con un costo computazionale significativamente maggiore. Inoltre, il learning rate è stato ridotto da 0.001 a 0.0001 per garantire una convergenza più graduale e stabile, evitando grandi aggiornamenti dei pesi che potrebbero compromettere l'ottimizzazione.
Il quarto modello conta un totale di **4,241,153 parametri** (16.18 MB), rappresentando la configurazione più complessa e potente sviluppata in questa analisi.

3.2 Explainable AI con LIME per le Reti Neurali

Per migliorare la comprensione del comportamento delle reti neurali, abbiamo utilizzato LIME (*Local Interpretable Model-agnostic Explanations*) per spiegare le predizioni del modello su un insieme di istanze selezionate. In particolare, abbiamo analizzato **1000 righe** del dataset, scelte casualmente, e per ciascuna di esse abbiamo generato una spiegazione locale utilizzando LIME.
Per ogni istanza, LIME identifica le **5 feature più influenti** che hanno contribuito alla predizione del modello e salva queste informazioni in un dizionario. Le importanze delle feature sono successivamente aggregate su tutte le istanze analizzate, ponderandole in base al loro contributo assoluto.

Infine, le feature sono ordinate in base alla loro importanza complessiva, fornendo una visione globale di quali variabili abbiano il maggior impatto sulle predizioni del modello.
Questo approccio ci ha permesso di identificare in modo trasparente le variabili più rilevanti per il modello, garantendo una maggiore interpretabilità delle sue decisioni.

3.3 XGBoost

Abbiamo deciso di passare dalle reti neurali a XGBoost poiché spesso si dimostra più efficace su dati strutturati e tabulari. XGBoost offre maggiore interpretabilità, gestisce nativamente i valori mancanti e richiede meno ottimizzazioni rispetto alle reti neurali, che possono sovradattarsi o incontrare difficoltà con dataset di piccole o medie dimensioni.

3.3.1 Primo Modello XGBoost

Il primo modello XGBoost, *model5*, è stato configurato per bilanciare capacità predittiva e rischio di overfitting. Sono stati utilizzati **500 alberi** con una **profondità massima di 6**, ottimizzando la capacità del modello di catturare pattern complessi senza esagerare in complessità. Il **learning rate**, fissato a 0.05, ha garantito aggiornamenti stabili e graduali durante l'addestramento. Inoltre, il modello utilizza l'**80% dei campioni** (*subsample*) e l'**80% delle feature** (*colsample_bytree*) per ciascun albero, introducendo casualità per migliorare la generalizzazione.
Per ridurre ulteriormente il rischio di overfitting, è stato impostato un **peso minimo dei figli** (*min_child_weight*) pari a 5, che impedisce la creazione di foglie con dati scarsamente rappresentativi. Infine, il parametro **gamma** è stato impostato a 0.1 per penalizzare gli split meno significativi, e il modello è stato reso ripetibile fissando il *random_state* a 42.

3.3.2 Secondo Modello XGBoost

Il secondo modello, *model6*, introduce modifiche significative rispetto al primo per aumentare la capacità del modello di catturare pattern complessi e integrare una regolarizzazione più forte per evitare il sovradattamento.
Rispetto al primo modello:

- Il numero di alberi è stato aumentato a **2000**, combinato con un **learning rate** ridotto a 0.01 per consentire una convergenza più graduale e precisa.
- La profondità degli alberi è stata portata a **8**, permettendo di modellare relazioni più complesse nei dati.
- Sono stati ridotti sia il **subsample** che il **colsample_bytree** al 70%, aumentando la casualità per migliorare la generalizzazione.
- Il **peso minimo dei figli** (*min_child_weight*) è stato aumentato a 10, e sono state introdotte regolarizzazioni aggiuntive: **reg_alpha** (L1) pari a 0.1 e **reg_lambda** (L2) pari a 1.0, per penalizzare i modelli troppo complessi.

Questi aggiustamenti hanno permesso al modello di mantenere un equilibrio tra capacità di apprendimento e generalizzazione.

3.3.3 Terzo Modello XGBoost

Il terzo modello, *model7*, si differenzia dagli altri modelli XGBoost e dalle reti neurali principalmente per l'approccio adottato nella gestione delle variabili categoriali. Invece di utilizzare il **one-hot encoding** per le variabili relative ai quartieri di *pickup* e *dropoff*, così come per il giorno della settimana, è stato implementato il **label encoding**. Questo metodo riduce la dimensionalità del dataset, consentendo al modello di gestire le feature categoriali in modo più efficiente.
A livello di configurazione dei parametri, sono stati mantenuti gli stessi valori del miglior modello precedente. Questa scelta ha permesso di combinare la potenza predittiva di XGBoost con una rappresentazione delle feature più compatta e adatta a dati strutturati. Questo approccio ha mostrato migliori risultati in termini di performance, riducendo il rischio di sovradattamento grazie alla minore complessità introdotta dal label encoding.
Inoltre per ogni modello di XGBoost sono state estratte le variabili più importanti tramite il metodo get_booster()

4 Results and Evaluation

4.1 Risultati Neural Network

Table 1: Confronto delle performance delle reti neurali in termini di Validation Loss e Validation MAE.

Modello	Validation Loss	Validation MAE
Modello 1	0.3338	178.7190
Modello 2	0.3325	178.1342
Modello 3	0.3309	176.4509
Modello 4	0.3299	175.6956

Figure 6: Confronto del MAE durante il training tra i modelli di reti neurali.

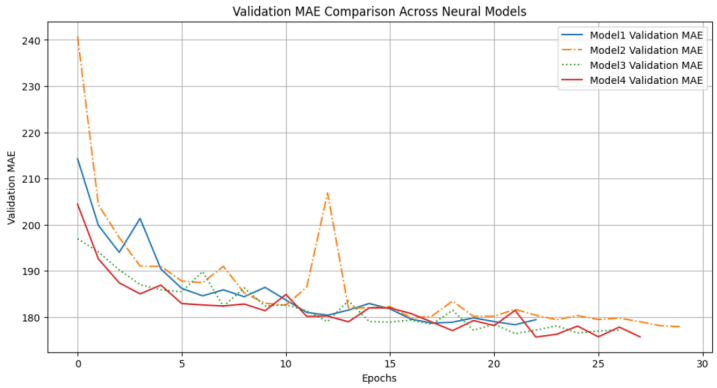

Figure 7: Confronto del MAE sulla validazione tra i modelli di reti neurali.

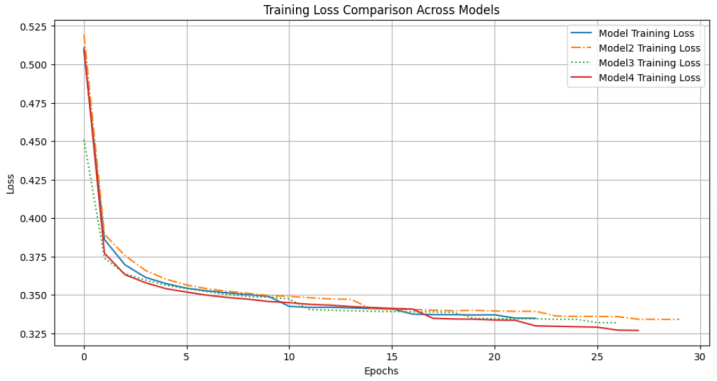

Figure 8: Confronto della Loss durante il training tra i modelli di reti neurali.

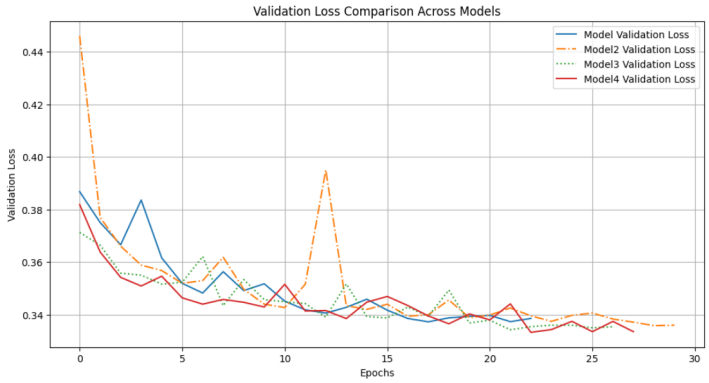

Figure 9: Confronto della Loss sulla validazione tra i modelli di reti neurali.

4.2 Risultati Lime

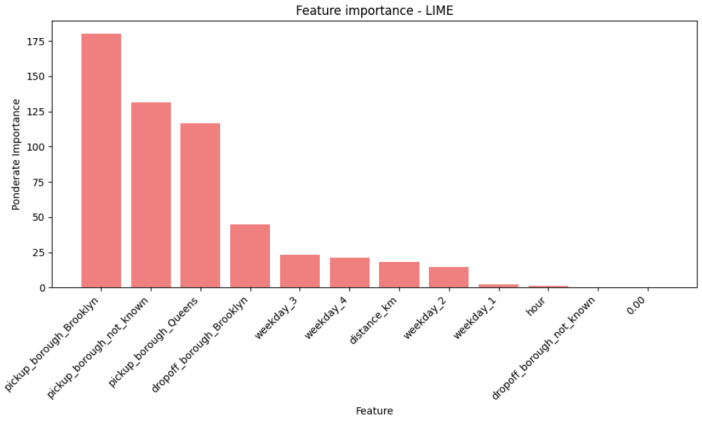

Figure 10: Feature Importance NN

4.3 Risultati XGBoost

Table 2: Confronto delle performance dei modelli XGBoost in termini di Validation MAE e Validation RMSLE.

Modello	Validation MAE	Validation RMSLE
XGBoost 1	159.9072	0.3105
XGBoost 2	153.3156	0.2989
XGBoost 3	152.3948	0.2965

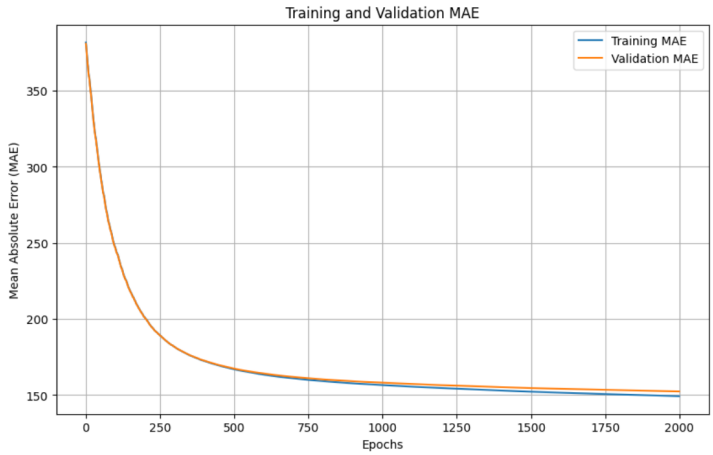

Figure 11: Andamento del MAE durante il training e la validazione per il modello XGBoost 3.

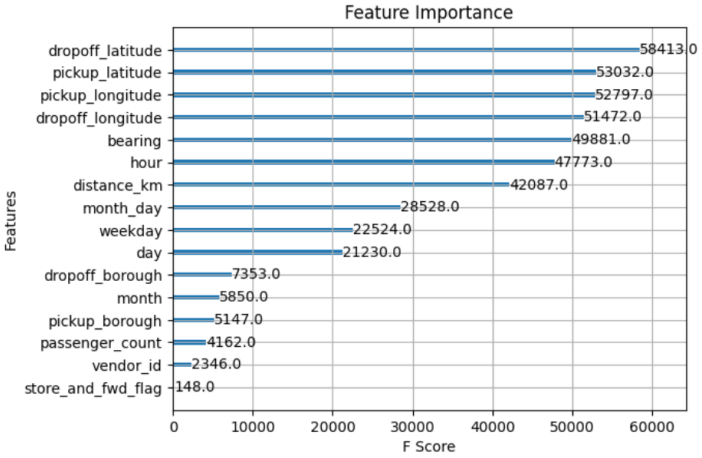

Figure 12: Importanza delle feature calcolata per il modello XGBoost 3.

4.4 Risultati sul nuovo Test set

4.4.1 NN

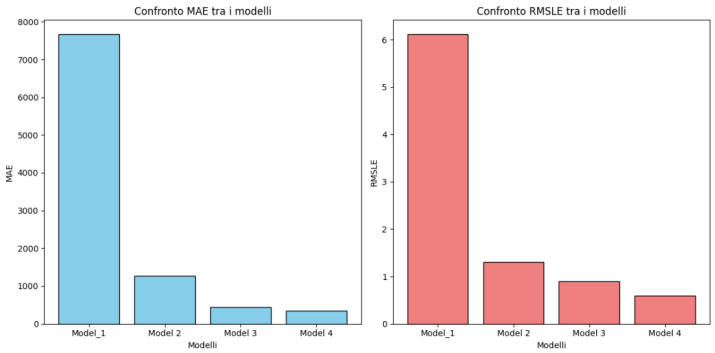

Figure 13: Risultati NN su test set.

4.4.2 XGBoost

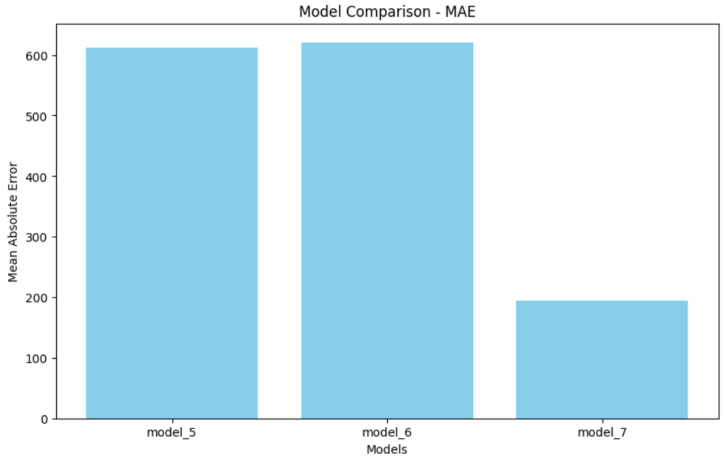

Figure 14: Risultati MAE XGBoost su test set.

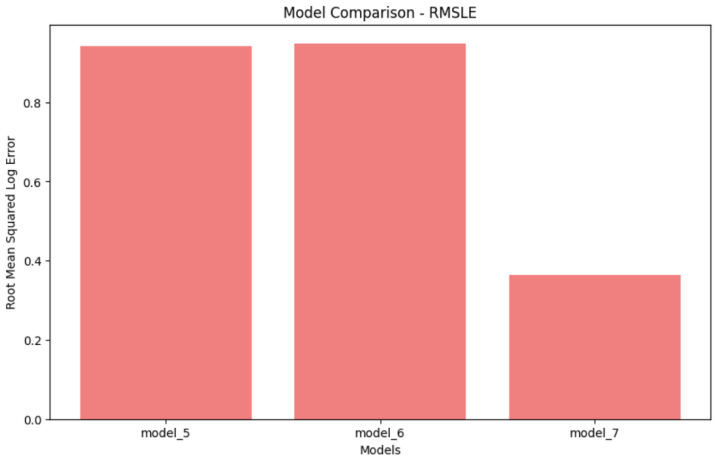

Figure 15: Risultati RMSLE XGBoost su test set.

5 Discussion

Il regressore XGBoost ha dimostrato prestazioni superiori in termini di interpretabilità ed efficienza computazionale rispetto alla rete neurale.

A differenza delle reti neurali profonde, che richiedono una quantità significativa di dati per apprendere efficacemente, XGBoost funziona bene anche con dataset di dimensioni più contenute o moderatamente grandi.
D'altro canto, pur avendo raggiunto un'accuratezza quasi comparabile, la rete neurale si è dimostrata più complessa e meno trasparente nelle decisioni prese (pur avendo mostrato un miglioramento progressivo tra i modelli, non sono riuscite a raggiungere livelli di accuratezza pari a quelli di XGBoost); infatti, il modello migliore tra le reti (Modello 4) ha ottenuto una Validation MAE di **175.6956** e una Validation Loss di **0.3299**. Tuttavia, XGBoost 3 ha superato queste prestazioni, raggiungendo una Validation MAE di **152.3948** e una Validation RMSLE di **0.2965**.

Nell'evaluation del Test set, possiamo notare come, il modello XGBoost che non contiene la normalizzazione dei dati e che presenta il label encoding al posto del one-hot per le variabili categoriche, abbia dei risultati decisamente migliori rispetto agli altri modelli; Infatti, la migliore rete ottiene una MAE pari a 337.122376, circa 140 secondi maggiore rispetto al modello XGBoost, il quale ha un MAE pari a un terzo rispetto ai due altri modelli XGBoost.

L'analisi dell'importanza delle feature tramite il grafico di Feature Importance ha evidenziato che variabili come *latitude, longitude, bearing* e *distance_km* hanno avuto un peso maggiore nelle predizioni, suggerendo che le nuove variabili sono risultate efficaci. Anche se la parte di feature engineering potrebbe essere migliorata in diversi modi. Ad esempio, si potrebbe verificare un miglioramento delle prestazioni con l'integrazione di caratteristiche aggiuntive, come le condizioni meteorologiche, i modelli di traffico o i giorni festivi.
Nonostante le buone prestazioni di XGBoost, alcuni limiti rimangono:

- La complessità computazionale del modello migliore (XGBoost 3) è significativa, specialmente con un elevato numero di alberi (2000) e una profondità di 8. Questo potrebbe essere ottimizzato riducendo leggermente i parametri senza compromettere eccessivamente le prestazioni.
- L'uso del *label encoding* potrebbe aver limitato l'efficacia del modello su variabili categoriali più complesse. In futuro, potrebbe essere interessante esplorare tecniche ibride, come *target encoding*.

Infine, le reti neurali profonde eccellono quando i dati presentano relazioni altamente non lineari e sono di natura non strutturata (ad esempio, immagini, testo o audio). Per un problema come la previsione della durata di un viaggio in taxi, i dati sono per lo più strutturati (latitudine, longitudine, orari, conteggio dei passeggeri, ecc.). In questi casi, algoritmi come XGBoost, sono spesso in grado di catturare le relazioni presenti nei dati con una complessità inferiore offrendo risultati altrettanto validi o migliori rispetto a una rete neurale con il vantaggio di una minore complessità nel processo di sviluppo e interpretazione.

6 Conclusions

In conclusione, i risultati dimostrano che XGBoost è una scelta più efficace rispetto alle reti neurali per questo problema, sia in termini di accuratezza che di interpretabilità. Tuttavia, ulteriori miglioramenti e ottimizzazioni possono ancora essere esplorati per ottenere un modello ancora più performante ed efficiente.