Università degli studi di Bari facoltà di scienze MM.FF.NN

Progetto ingegneria della conoscenza

TrainDelay-project

by

Vito Proscia mat. 735975

Email: v.proscia3@studenti.uniba.it

Repository github: TrainDelay-project

Anno accadenico 2022-2023

Contents

1 Introduzione

1.1 Definizione obiettivo principale

L'obiettivo principale del progetto è la creazione di un motore di ricerca che trova i migliori itinerari di viaggio in treno sulla base della stazione di partenza e di arrivo e che, per ogni viaggio, mostra una predizione del probabile ritardo.
Questo sistema non solo potrà far risparmiare del tempo a chi organizza dei viaggi valutando ogni singola tratta, ma garantirà un risparmio econimico ai viaggiatori garantendo che la tratta scelta dal sistema sia la minima e necessaria per arrivare alla destinazione, inoltre la predizione del ritardo andrà a ridurre in maniera significativa il disagio da parte dei viaggiatori.

1.2 Tool utilizzati

Per la sperimentazione sono stati usati diversi stumenti/librerie, i principali sono:

- **PySWIP**, libreria Python che fornisce un'interfaccia per utilizzare SWI-Prolog, usato per la rappreesentazione formale della schedul dei treni;
- **NetworkX**, libreria Python utilizzata per la creazione, l'analisi e la manipolazione di reti complesse. Questa libreria fornisce un insieme di strumenti per la rappresentazione di reti e grafi, oltre a un'ampia gamma di algoritmi e funzioni per eseguire diverse operazioni su di essi;
- **Scikit-learn**, libreria open-source che fornisce un vasto insieme di strumenti nell'ambito del machine learning e dell'apprendimento automatico.

2 Rappresentazione formale della conoscenza

La rappresentazione formale della conoscenza è importante per consentire l'espressione della conoscenza in modo preciso, organizzato e interpretabile da parte di sistemi informatici.
Per gestire formalmente la conoscenza, facilitando la ricerca e l'accesso a informazioni specifiche, si costruisce una **knowledge base** (base di conoscenza), una raccolta strutturata di informazioni o dati che rappresenta la conoscenza su un determinato dominio o argomento, utilizzate per immagazzinare e organizzare la conoscenza in modo che sia accessibile e utilizzabile da sistemi informatici.

2.1 Origine dei dati

Tutte le informazioni relative allo schedul dei treni sono state reperite per mezzo dell'interrogazione alle API messe a disposizione dal sito www.viaggiatreno.it, mentre le informazioni relative alle stazioni sono state recuperate dal repository "trenitalia: scraping di viaggiatreno".[1]

2.2 Descrizione dei dati

La parte iniziale del progetto si è concentarta sulla rappresentazione formale attraverso fatti e regole Prolog (linguaggio di programmazione logica utilizzato per definire relazioni tra fatti e regole attraverso la logica dei predicati) dello schedule dei treni e delle stazioni, in particolare ogni treno si è ritenuto opportuno caratterizzarlo da:

1. *ID treno* identificatore univoco del treno;
2. *Tipo di treno* regionale o nazionale;
3. *ID stazione di partenza*;
4. *ID stazione di arrivo*;
5. *Orario di partenza* (HH:MM);
6. *Orario di arrivo* (HH:MM);
7. *Lista delle fermate*.

Di seguito è riportato un esempio:

```
train(320, nazionale, s01700, s01301, '15:10', '15:58', [s01700, s01307, s01301]).
train(321, nazionale, s01301, s01700, '18:02', '18:50', [s01301, s01307, s01700]).
train(322, nazionale, s01700, s01301, '17:10', '17:58', [s01700, s01307, s01301]).
train(323, nazionale, s01301, s01700, '20:02', '20:50', [s01301, s01307, s01700]).
train(324, nazionale, s01700, s01301, '19:10', '19:58', [s01700, s01307, s01301]).
train(325, nazionale, s01301, s01700, '22:02', '22:50', [s01301, s01307, s01700]).
```

Figure 1: Esempio di rappresentazione formale dello shedule dei treni

Mentre ogni stazione si è pensato caratterizzarla da:

1. *ID stazione*, identificatore univoco delle stazioni;
2. *Nome stazione*;
3. *Regione stazione*.

Di seguito è riportato un esempio:

```
station(s06950, "BINARIO S.MARCO VECCHIO", "Toscana").
station(s11113, "BISCEGLIE", "Puglia").
station(s00870, "BISTAGNO", "Piemonte").
station(s01202, "BISUSCHIO VIGGIU", "Lombardia").
station(s11501, "BITETTO PALO DEL COLLE", "Puglia").
station(s03313, "BIVIO D`AURISINA", "Friuli Venezia Giulia").
```

Figure 2: Esempio di rappresentazione formale delle stazioni

Questa rappresentazione dei treni e delle stazioni nel linguaggio di programmazione logica ha fornito una base solida per la gestione e l'analisi dei dati ferroviari.

Con questa struttura dati, è stato possibile effettuare interrogazioni, trovare treni tra stazioni, e condurre altre operazioni analitiche necessarie per il progetto.

2.3 Rappresentazione basata su grafo

2.3.1 Costruzione grafo

Per la possibilità di ricercare l'itinerario di viaggio migliore, cioè con il numero minimo di stazioni, si è pensato di costruire un grafo delle stazioni, dove ogni nodo rappresenta una stazione diversa (`idStazione`) e la presenza di un arco tra due nodi si traduce in un collegamento ferroviario tra le due stazioni.

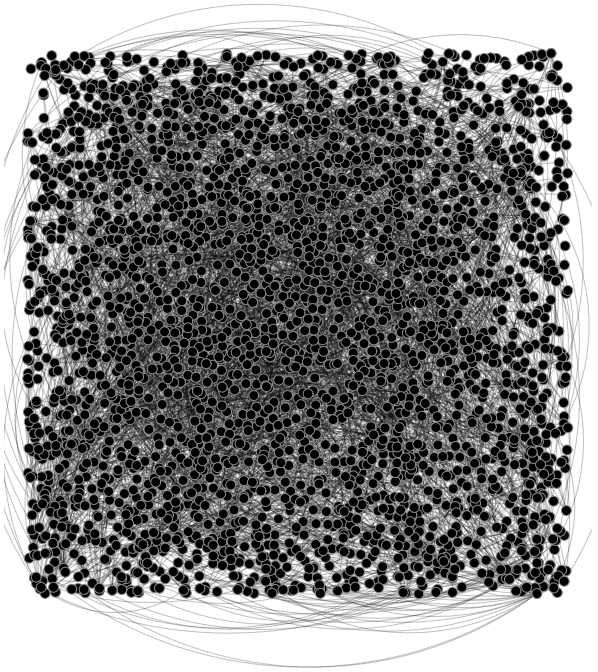

Figure 3: Grafo delle stazioni

2.3.2 Ricerca grafo

Per la ricerca del percorso più breve tra due stazioni si è utilizzato **l'algoritmo di Dijkstra**, un algoritmo di ricerca del cammino più breve in un grafo pesato con pesi non negativi, in questo caso specifico il peso di ogni arco è 1, quindi l'algortmo cercherà il percorso con il più breve numero di nodi.
Inizia da un nodo sorgente e calcola le distanze minime da esso a tutti gli altri nodi, mantenendo una coda di priorità, durante l'esecuzione, visita i nodi adiacenti al nodo corrente e aggiorna le distanze minime se trova un cammino più breve. Implementato dalla libreria **NetworkX** che restituisce il percorso più breve sotto forma di lista di nodi.

Anche se il grao prodotto non è propriamente pesato, si è pensato di usare l'algoritmo di Dijkstra, per una possibile estensione e miglioramento del sistema convertendo il grafo attuale in uno pesato, magari andando ad impostare come pesi degli archi le distanze che intercorrono tra le stazioni collegate.

Algorithm 1 Algoritmo di Dijkstra

```
1:  procedure DIJKSTRA(G, s)
2:      dist ← array di distanze inizializzato a ∞ per tutti i nodi
3:      dist[s] ← 0
4:      S ← insieme vuoto dei nodi visitati
5:      while S non contiene tutti i nodi do
6:          u ← nodo non visitato con la minima distanza in dist
7:          Aggiungi u a S
8:          for all nodi adiacenti v di u do
9:              alt ← dist[u]+ peso dell'arco tra u e v
10:             if alt < dist[v] then
11:                 dist[v] ← alt
12:             end if
13:         end for
14:     end while
15:     return dist
16: end procedure
```

Operazione	**Complessità**
Inizializzazione	$O(V)$
Inserimento di nodi nella coda	$O(V \log V)$
Estrazione del nodo con distanza minima	$O(\log V)$
Aggiornamento delle distanze	$O(E \log V)$
Totale	$O((V + E) \log V)$

Table 1: Complessità dell'algoritmo di Dijkstra con heap binario

2.4 Query Knowledge base

Per andare ad interagire un Knowledge base vengono eseguite delle **query**, interrogazioni alla KB, che permettono di estrapolare informazioni specifiche. In questo caso sono state messe a disposizione delle query predefinite per poter recuperare le informazioni relative ai treni secondo le esigenze dell'utente. In particolare sono state pensate quattro query principali, che che verranno eseguite in base alle esigenze dell'utente correlate alla ricerca specifica che andrà a fare.

2.4.1 Query predefinite

In questa sezione verranno presentate le principali **query** progettate e messe a disposizione che il sistema userà per interagire, in maniere efficente, con la base di conoscenza (KB).

Si focalizzeranno sulle varie opzioni di ricerca progettate per il sistema, recuperando tutte le informazioni necessare ad un suo funzionamento efficente.

Query principali:

1. Restituisce la lista di tutti i treni (`Trains`) che partono da una determinata stazione (`StationName`);

```
% Rule for finding all trains departing from a station
trains_departure_from_station_name(StationName, Trains) :-
    findall(TrainID, (station(DepartureStationID, StationName, _),
train(TrainID, _, DepartureStationID, _, _, _, _)), Trains).

```

Figure 4: Regola n.1

2. Restituisce la lista di treni (`Trains`) che partono da una stazione specifica (`StationName`) a un determinato orario di partenza (`Departure`);

```
% Rule for finding all trains leaving a station after a specific t
ime (HH:MM)
trains_departure_from_station_name_at_time(StationName, Departure,
Trains) :-
    findall(TrainID, (station(DepartureStationID, StationName, _),
train(TrainID, _, DepartureStationID, _, DepartureTime, _, _), equ
al_major_time(DepartureTime, Departure)), Trains).

```

Figure 5: Regola n.2

3. Restituisce la lista di treni (`Trains`) che collegano due stazioni specifiche, partendo dalla stazione `DepartureStationName` e arrivando `ArrivalStationName` dopo un orario specifico `Time` (nella foramto HH:MM);

```
% Rule for finding all trains between two stations after a specifi
c time (HH:MM)
trains_departure_between_stations_name_at_time(DepartureStationNam
e, ArrivalStationName, Time, Trains) :-
    findall(TrainID, (station(DepartureStationID, DepartureStation
Name, _),
        station(ArrivalStationID, ArrivalStationName, _),
        train(TrainID, _, DepartureStationID, ArrivalStationID, De
partureTime, _, _),
        equal_major_time(DepartureTime, Time)),
        Trains).
```

Figure 6: Regola n.3

4. Restituisce la lista di treni (Trains) che collegano due stazioni specifiche, partendo dalla stazione `DepartureStationName` e arrivando `ArrivalStationName`;

```
% Rule for finding all trains between two stations
trains_departure_between_stations_name(DepartureStationName, Arriv
alStationName, Trains) :-
    findall(TrainID, (station(DepartureStationID, DepartureStation
Name, _),
        station(ArrivalStationID, ArrivalStationName, _),
        train(TrainID, _, DepartureStationID, ArrivalStationID, _,
_, _)),
        Trains).
```

Figure 7: Regola n.4

Oltre queste query sono state prodotte altre regole, di supporto, per il funzionamento interno di queste principali, come quella per convertire le stringhe che rappresentano gli orari dal formato `"HH:MM"` in minuti, quella per reperire tutte le info dei treni dall'id, etc...

3 Machine Learning

Una delle funzionalità cardine del sistema è quella di predire se un treno sarà in ritardo, per implementrala è stato necessario addestrare un modello di Machine larning che in base a certe caratteristiche dello specifico treno farà una previsione più o meno corretta sul suo andamento.

In origine si è sperimentato con task di regressione, cioè, basandosi su un insieme di caratteristiche e dati, cercare di predire il valore esatto del ritardo, successivamente a causa di risultati non troppo ottmali ci si è spostati sul task di classificazione binaria, verrà predetto solo se un treno farà ritardo o meno.

3.1 Origine dataset

Il dataset di addestramento e test è stato recuperato per mezzo dell'interrogazione tramite API al serivio viaggiotreno.it, in particolare si sono recuperate le informazoni giornaliere circa i treni (ID, tipo di treno, ...) ed in più il ritardo effettuato della corsa specifica fornendo così un insieme di dati essenziali per l'addestramento e la valutazione del modello di classificazione.

3.2 Analisi del dataset

Il dataset ottenuto è composto da circa 101169 osservazioni per otto features che, come già detto sopra, vanno a descrivere una serie di caratteristiche legate all'andamento giornaliero dei treni, abbiamo:

1. *train_id*[numeric]: identificatore univoco del treno;
2. *origin*[string]: nome della stazione di partenza;
3. *arrival*[string]: nome della stazione di arrivo;
4. *departure_time*[string]: orario di partenza (HH:MM);
5. *arrival_time*[string]: orario di arrivo (HH:MM);
6. *delay*[numeric]: ritardo registrato;
7. *train_type*[string]: tipo di treno, regionale o nazionale;
8. *detection_date*[date]: data della corsa.

	train_ID	origin	arrival	departure_time	arrival_time	delay	train_type	detection_date
0	13	M N CADORNA	LAVENO	06:39	08:23	0	regionale	2023-08-13
1	20	LAVENO	M N CADORNA	06:38	08:09	0	regionale	2023-08-13
2	25	M N CADORNA	LAVENO	08:52	10:23	0	regionale	2023-08-13
3	26	LAVENO	M N CADORNA	07:38	09:09	0	regionale	2023-08-13
4	29	M N CADORNA	LAVENO	09:39	11:23	0	regionale	2023-08-13

Figure 8: Esempio delle prime istanze del dataset

3.3 Preparazione dati

Prendendo il dataset così descritto ci sono una serie di problematiche da risolvere per poter usare i dati, in particolare andando a considerare le osservazioni notiamo che per i valori nominali, in questo caso quelli che esprimono il nome della stazione di partenza e di arrivo, ci sono dei caratteri che andrebbero modificati per formattare meglio il dataset, in particolare si sono andati a sostituire i punti, le virgole, gli accenti e le doppie virgolette con degli underscore per poter meglio gestire il dataset.

3.3.1 Analisi input features

Considerando le feature di input, cioè quelle sulle quali il modello andrà ad imparare, non tutte sono funzionali per per il raggiongimento del nostro scopo, in particolare si sono escluse:

- *origin* (il modello scelto non accetta dati di tipo string);
- *arrival* (il modello scelto non accetta dati di tipo string);
- *detection_ date* (nessuna correlazione sulle feature su cui fare predizione).

Per quanto riguarda le feature rimanenti si è ritenuto opportuno includere nel learning anche *train_ id* perchè si è osservato che determinati treni facevano sempre ritardo, inoltre si è optato per una trasformazionde dei valori per adattarli al modello, specificatamente i valori di *departure_ time* e *arrival_ time* da stringhe nel formato HH:MM si è passati a valori numerici che rappresentano i minuti ($hour * 60 + minutes$), inoltre si è operato anche su *train_ type* eseguendo una binarizzazione:

$$\begin{cases} 1 & \text{if } train\_type(i) = regionale \\ 0 & \text{if } train\_type(i) = nazionale \end{cases}$$

3.3.2 Analisi target feature

La target feature rappresenta l'biettivo sella notra predizione, in questo caso *delay* che rappresenta il ritardo di una detrminata corsa, per la regressione lasciamo il valore così come ci viene dato, mentre per la classificazione anche in questo caso si è eseguita una binarizzazione:

$$\begin{cases} 1 & \text{if } delay(i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Nel dataset abbiamo una distribuzione abbastanza bilanciata dei valori per la feature *delay* (41.3% per treni in ritardo e 58.7% per treni in orario), questo evita stategie per bilanciare i dati come data augmentation, cost-sensitive learning, etc....

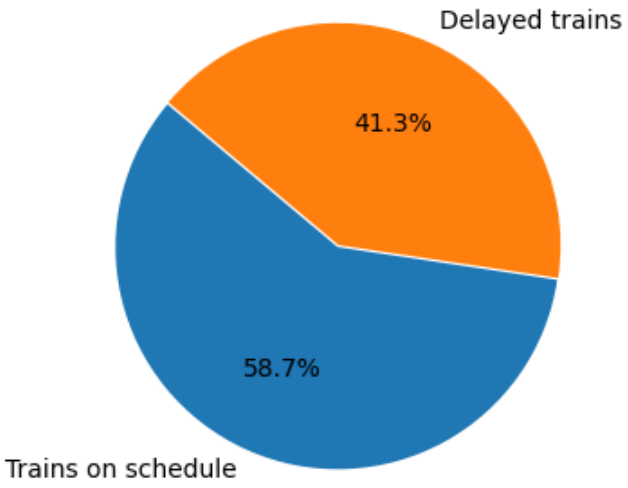

Figure 9: Distribuzione dei treni con e senza ritardo

3.4 Apprendimento automatico

Per produrre un sistema che va predire se un treno farà ritardo o meno si ricorre all'apprendimento automatico o *machine learning* che consiste nell'addestrare un modello ad imparare dai dati e a migliorare le proprie prestazioni nei compiti specifici senza essere esplicitamente programmanti.
Come già accennato si sono fatti esperimenti sia per la regressione che per la classificazione, in particolare per la prima si cerca di trovare una funzione $\hat{y}$, che meglio somiglia alla funzione "reale" che descrive i dati $y = f(X_1, X_2, \ldots, X_n)$, mentre per la seconda si cerca di assegnare ad un'ovvervazione una certa categoria.

3.5 Random forest

Per questo caso speficico, sia per la regressione che per la classificazione si è usata la **Random forest**, modello di apprendimento automatico che combina molteplici alberi decisionali (*decision trees*), entrando così nella categoria di modelli *ensamble*, per migliorare la previsione e la generalizzazione.
La random forest fa parte della gamma dei modelli di **apprendimento supervisionato**, cioè viene addestrato su un insieme di dati di addestramento che includono sia le caratteristiche (input features) che le risposte corrette (output features). Il modello impara a fare previsioni basate su questi esempi etichettati.

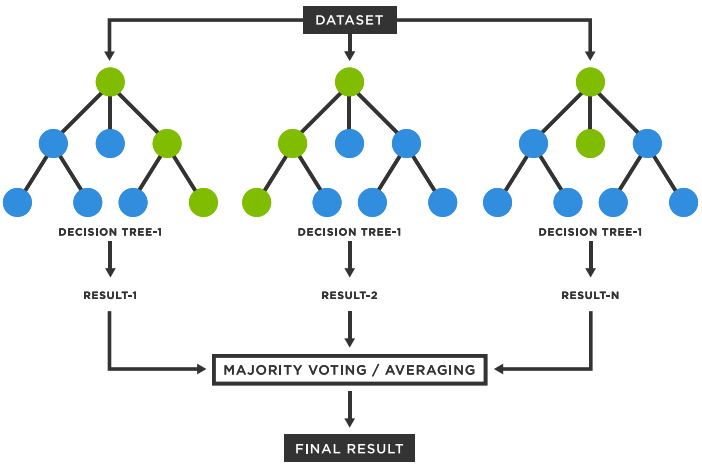

Figure 10: Esempio di random forest www.spotfire.com

3.5.1 Funzionamento

In particolare la Random Forest funziona creando un insieme di alberi decisionali (regressione/classificazione in base al task), ognuno addestrato su un subset casuale dei dati di addestramento e su un subset casuale delle caratteristiche. Questo processo introduce variabilità e riduce il rischio di sovradattamento (*overfitting*) cioè quando il modello si lega troppo ai dati di *training* risultando inefficacie su dati mai visti.
Quando si effettua una previsione, ciascun albero fornisce una previsione, e la Random Forest combina queste previsioni per ottenere un risultato finale più accurato e stabile.

Nel nostro caso per la regressione si è optato per una media delle predizioni di ogni albero, mentre per la classificazione la classe più predetta.

3.5.2 Configurazione scelta

Dopo varie sperimentazioni si riporta la migliore configurazione del modello, in particolare attraverso la libreria `scikit-learn` [2] si è definita una random forest le cui caratteristiche perculiari sono:

Regressione

1. `n_estimators = 35`, che rappresenta il numero di alberi su cui lavorare;
2. `criterion = "squared_error"`, che rappresenta il criterio di *split* degli alberi, in particolare il criterio è *l'errore quadratico medio*, una misura che quantifica la differenza tra i valori previsti da un modello e i valori osservati o reali puntando a minimizzare la varianza, si calcola:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

   dove:

   - n è il numero osservazioni;
   - y_i rappresenta il valore osservato (reale);
   - $\hat{y}_i$ rappresenta il valore predetto.

   Il criterio `squared_error` è stato scelto per la sua efficacia e chiarezza di espressione, infatti è espresso nella stessa unità dei dati reali, in questo caso minuti di riatrdo, rendendo più comprensibile di quanto le previsioni del modello si discostano, in media, dai valori reali in termini di minuti.

Classificazione

1. `n_estimators = 50`, che rappresenta il numero di alberi su cui lavorare;
2. `criterion = "gini"`,criterio di *split* degli alberi, in particolare il criterio punta a minimizzare l'impurità dei nodi, quindi per scegliere l'attributo che meglio divide i dati in base alla target feature per ogni nodo si calcola:

$$GINI(v) = 1 - \sum_{i=1}^{|c|} p_i^2$$

dove:

- v è il nodo in esame;
- c rappresenta l'inseme delle classi;
- p_i è la probabilità che un campione nel nodo v appartenga alla classe i.

infine vene scelto il nodo con minor impurità.

L'indice `gini` è stato scelto per la sua efficenza, infatti la complessità per il calcolo dell'indice per uno nodo è dell'ordine di $O(c)$, dove c è il numero di classi.

3.6 Valutazione dei modelli

Per andare a valutare i modelli prodotti dopo l'addestramento si è usata la k-Fold-Cross-Validation, che esegue la riprtizione del dataset D in k sottoinsiemi (folds), $D_1, D_2, \ldots, D_k$ prevede k iterazioni ed all'i-esima iterazione il sottoinsieme D_i sarà usato come dataset di test, metre l'unione degli altri sarà usata per allenare il modello.

Alla fine delle k iterazioni, vengono raccolti i risultati delle valutazioni e calcolate le metriche di performance per valutare quanto bene il modello si comporta in media su dati di test diversi.

3.6.1 Metriche scelte regressione

Ai fini della valutazione si hanno a disposizione moltissime metriche adatte per moltissime situazioni, in questo caso speficiso si sono scelte:

1. Mean absolute error (MEA) che misura la media degli errori assoluti tra le previsioni del modello e i valori osservati

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

2. Mean squared error (MSE) che calcola la media dei quadrati degli errori tra le previsioni del modello e i valori osservati

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

3. R^2-score, indica quanto il modello spiega le variazioni nei dati

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

dove:

- n è il numero osservazioni;
- y_i rappresenta il valore osservato (reale);
- $\hat{y}_i$ rappresenta il valore predetto;
- $\bar{y}_i$ rappresenta il valore medio.

3.6.2 Metriche scelte classificazione

Per andare a valutare il modello prodotto si sono scelte le metriche più comuni:

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

Inoltre per avere un quadro della situazione più dettagliato si presentano altre mertiche che mettono in correlazione le due precedenti, come:

1. $f\beta - score$, combinazione delle metriche classiche di *precision* e *recall* bilanciate da un parametro β nella media armonica

$$f\beta - score = \frac{(1 + \beta^2) \cdot Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$$

2. *AUC−ROC* (Area Under the Receiver Operating Characteristic Curve), area sotto la curva che rappresenta la relazione tra il tasso di veri positivi (True Positive Rate, TPR) e il tasso di falsi positivi (False Positive Rate, FPR) al variare della soglia di classificazione

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{FP + TN}$$

3.7 Risultati

In questa sezione verranno riportati i risultati migliori di tutte le metriche discusse precedentemente:

Regressione:

- **MAE**: 1.27;
- **MSE**: 4.86;
- **R^2 score**: 0.58;

Classificazione:

- **Precision**: 0.94;
- **Recall**: 0.86;
- **F2-score**: 0.92;
- **AUC-ROC**: 0.91.
- **Matrice di confusione**:

Actual_negative	Actual_positive	
11522	425	Predict_negative
1107	7180	Predict_negative

3.8 Considerazioni

In generale il modello di classificazione mostra prestazioni solide, con un'accuratezza elevata, alta precisione e una buona capacità discriminativa.
Tuttavia, il *recall* è leggermente inferiore al 90%, suggerendo possibili aree di miglioramento. L'*F2-score* bilanciato fornisce un buon compromesso tra *precision* e *recall*, indicando un equilibrio nel modello.

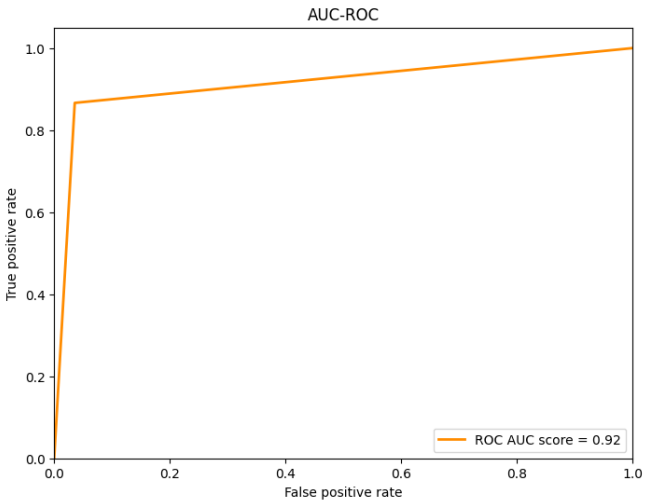

Figure 11: Grafico AUC-ROC

4 Interfaccia Grafica

Per interagire andare ad interagire con il sistema, all'avvio viene mostrato un menù, navigabile con i tasti direzionali "↑" e "↓", con tre opzioni:

1. `Cerca treno tra due stazioni`, per rintracciare tutti i treni tra la stazione di partenza e la stazione di arrivo dopo un determinato orario;
2. `Cerca itinerario`, fornice un itinerario, cioè tutti le informazioni relative ai treni da prendere (orario, tipo, etc...) tra due stazioni;
3. `Uscire`, per terminare l'applicazione.

Di seguito verranno riportate alcune immagini relative alla UI del sistema, includendo il menù principale e le diverse funzionalità:

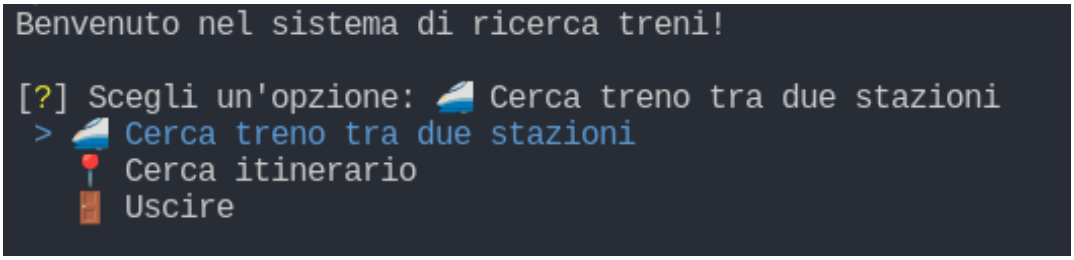

Figure 12: Distribuzione dei treni con e senza ritardo

```
Inserisci la stazione di partenza: Bari centrale
Inserisci la stazione di destinazione: Milano centrale
Inserisci l'ora di partenza (HH:MM): 06:00
+-------------------------------------------------------------------------------------------+
|                              BARI CENTRALE --> MILANO CENTRALE                            |
+------------+--------------+------------------+----------------+--------------------------+
|   TrainID  |   TrainType  |   DepartureTime  |   ArrivalTime  |   Predicted delay (AI)   |
+------------+--------------+------------------+----------------+--------------------------+
|     9806   |   nazionale  |       06:35      |      13:25     |          Yes             |
|     8830   |   nazionale  |       16:30      |      23:55     |          Yes             |
|     8894   |   nazionale  |       16:30      |      23:55     |           No             |
+------------+--------------+------------------+----------------+--------------------------+
```

Figure 13: Distribuzione dei treni con e senza ritardo

```
Inserisci la stazione di partenza: Gioia del colle
Inserisci la stazione di destinazione: Roma termini
+-------------------------------------------------------------------------------+
|                        GIOIA DEL COLLE --> BARI CENTRALE                      |
+-------------+-------------------+-----------------+---------------------------+
|   TrainID   |   DepartureTime   |   ArrivalTime   |   Predicted delay (AI)    |
+-------------+-------------------+-----------------+---------------------------+
|    19836    |       09:30       |      10:10      |            No             |
+-------------+-------------------+-----------------+---------------------------+
+-------------------------------------------------------------------------------+
|                         BARI CENTRALE --> ROMA TERMINI                        |
+-------------+-------------------+-----------------+---------------------------+
|   TrainID   |   DepartureTime   |   ArrivalTime   |   Predicted delay (AI)    |
+-------------+-------------------+-----------------+---------------------------+
|     8348    |       06:10       |      10:03      |           Yes             |
|     704     |       16:05       |      22:29      |           Yes             |
|    36166    |       16:05       |      22:29      |           Yes             |
+-------------+-------------------+-----------------+---------------------------+
```

Figure 14: Distribuzione dei treni con e senza ritardo

5 Conclusioni

L'obiettivo principale era sviluppare un sistema che potesse aiutare le persone a trovare la migliore soluzione per il viaggio in treno, andando a combinare la ricerca automatica di un itinerario che tenesse conto del numero di stazioni da percorrere e l'intelligenza artificiale per predire se il treno farà ritardo, andando così a limitare il disagio, in caso, da parte dei viaggiatori.

5.1 Rappresentazione e ricerca

Per la parte relativa alla ricerca, in primo luogo si sono reperiti i dati necessari attraverso chiamate API a vari servizi, successivamente si è andati a costruire una base di conoscenza (KB) per rappresentare formalmente la conoscenza ottenuta ed infine si è andati a progettare un grafo delle stazioni per applicare algoritmi di *path-finding* in modo da minimizzare il numero di fermate.

5.2 Apprendimento

Per la seconda parte, relativa all'addestramento di un modello di machine learning con l'obiettivo di predire il ritardo di un treno, inizialmente si sono reperiti i dati relativi alle corse dei treni attraverso chiamate API giornaliere ai servizi precedentemente citati, successivamente si è analizzato il dataset formato, andando a eseguire una pre-eleborazione dei dati rimuovendo osservazioni problematiche per causa di dati mancanti, incoerenza e valori errati, eseguendo l'ingegnerizzazione delle feature separando le features di input da quelle di output, rimuovendone alcune e trasformandone i valori di altre.
Come modello per addestrare si è selto la Random Forest, che combina vari alberi decisionali per fornire una previsione.
I risultati ottenuti sono molto buoni avendo molti indicatori superiori al 90%, dimostrando la solidità del modello nel nostro contesto.

Consideando il tutto si piò dire di aver raggiunto l'obiettivo preposto, ottenendo un sistema efficente e soprattutto d'aiuto per noi viaggiatori.

6 Sviluppi futuri

Nonostante il buon funzionamento ed risultati ottenuti, il sistema presentato è aperto a sviluppi futuri che possano rendere il tutto ancora più efficiente e all'avanguardia.

Di seguito sono descritti alcuni dei possibili sviluppi futuri da esplorare:

1. Espansione della copertura ferroviaria andando a completare le informazioni relative a treni e staziono ed integrazione con altri servizi ferroviari come Italo, Frecciarossa, etc. . . ;

2. Miglioramento della ricerca andando a suggerire all'utente, sulla base dei caratteri inseriti, le stazioni che iniziano con quei caratteri;

3. Integrazione di dati in tempo reale andando a fornire all'utente informazioni in *real time*;

4. Creazione di client mobile per poter usare il sistema in mobilità.

Queste sono alcune delle possibili migliorie/feature da poter implementare per rendere il sistema ancora più efficente e funzionale.

References

[1] @sebas. *trenitalia: scraping di viaggiatreno*. 2012. URL: https://github.com/sabas/trenitalia.

[2] G. Varoquaux A. Gramfort D. Cournapeau O. Grisel **and** A. Mueller. *Scikit-learn Machine Learning in Python*. **version** 1.3.1. URL: https://scikit-learn.org/stable/index.html.