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

Progetto ingegneria della conoscenza

TrainDelay-project

by

Vito Proscia mat. 735975

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 in tremini di stazioni e orari di partenza e di arrivo, ma garantirà un risparmio econimico ai viaggiatori garantendo che la tratta scelta dal sistema sia la minima e necessaria per arrivare alla destinazione.

1.2 Tool utilizzati

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

- **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.
- ...

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. Le knowledge base possono includere fatti, regole, concetti e relazioni tra concetti.

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 viaggiatreno, 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 schedul 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*.

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]).
```

Mentre ogni stazione si è pensato caratterizzarla da:

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

Esempio:

```
station(s11504, "ACQUAVIVA DELLE FONTI", "Puglia").
station(s12026, "ACQUEDOLCI-S.FRATELLO", "Sicilia").
station(s00867, "ACQUI TERME", "Piemonte").
station(s11907, "ACRI BISIGNANO LUZZI", "Calabria").
station(s05420, "ADRIA", "Veneto").
```

2.3 Grafo

2.3.1 Costruzione grafo

Per la possibilità di ricercare l'itinerario di viaggio mugliore, 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.

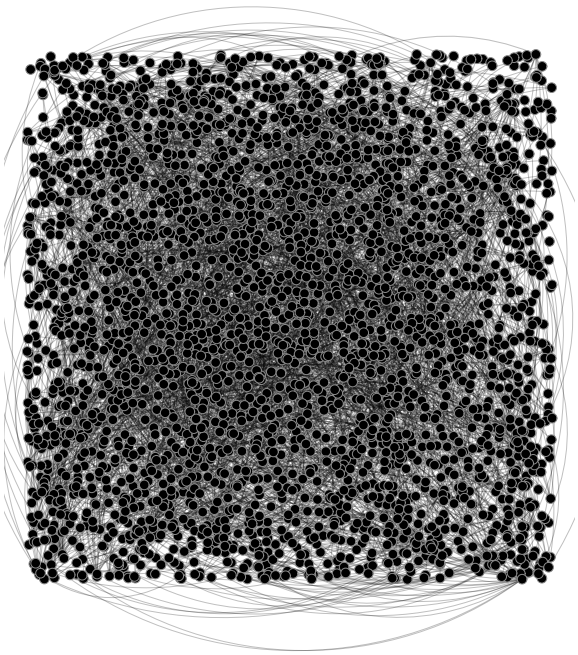

Figure 1: Grafo delle stazioni

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 piccolo 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**.

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
```

Commento sull'algoritmo

"'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas placerat tempus dictum. Vivamus dolor velit, condimentum nec scelerisque nec, blandit at ipsum. Maecenas venenatis sapien vitae rhoncus ultrices. Etiam blandit enim a aliquet sollicitudin. Curabitur ac ligula ac elit dignissim efficitur sed vitae nisi. Aenean porta magna sed pulvinar pretium. Curabitur gravida dolor quis arcu malesuada sodales. Nunc tortor sem, luctus et erat vel, bibendum vestibulum arcu. Integer in tellus mollis, aliquam ipsum vitae, consectetur tortor. Sed vehicula magna ac tristique porttitor. In vel pharetra tellus. Nulla facilisi. "'

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 disposizioni delle query predefinite per poter recuperare le informazioni relative ai treni secondo l'esigenze dell'utente.

In particolare sono state pensate quattro query principali che che verranno eseguite in base alle esigenze dell'utente in base alla ricerca specifica che andrà a fare.

Query messe a disposizione dal sistema:

1) Ricerca di tutti i treni che partono da una determinata stazione

```
% 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 2: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

2) Ricerca di tutti i treni che partono da una determinata stazione da un determinato orario

4) Ricerca di tutti i treni disponibili tra due stazioni

3) Ricerca di tutti i treni disponibili tra due stazioni dopo un determinato orario

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...

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

```

Figure 3: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

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

Figure 4: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

3 Machine Learning

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.

3.2 Analisi del dataset

Il dataset ottenuto è composto da 101169 osservazioni per otto features che, come già detto sopra, vanno a descrivere una serie di caratteristiche legate

```
% Rule for finding all trains between two stations after a specific time (HH:MM)
trains_departure_between_stations_name_at_time(DepartureStationName, ArrivalStationName, Time, Trains) :-
    findall(TrainID, (station(DepartureStationID, DepartureStationName, _),
        station(ArrivalStationID, ArrivalStationName, _),
        train(TrainID, _, DepartureStationID, ArrivalStationID, DepartureTime, _, _),
        equal_major_time(DepartureTime, Time)),
        Trains).
```

Figure 5: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

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.

Ecco un esempio:

3.3 Preparazione dati

Prendedo 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.

	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 6: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

3.3.1 Analisi input features

Considerando le feature di input, cioè quelle sulle quali il modello andrà ad imparare, non tutte sono importanti 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 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:

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

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, essendo un task di classificazione, cioè prevedere se un determinato treno farà ritardo o meno, anche in questo caso si è eseguita una binarizzazione:

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

Nel dataset abbiamo una distribuzione abbastanza bilanciata dei valori per la feature *delay*

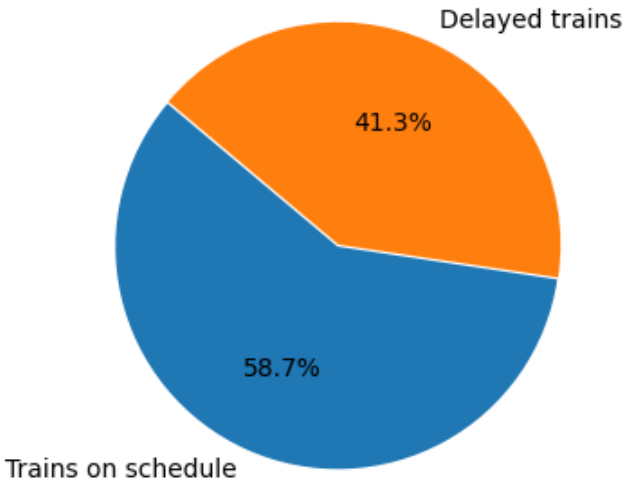

Figure 7: Distribuzione dei treni con e senza ritardo

3.4 Apprendimento automatico

Per produrre un sistema che va predire se un treno farà ritardo o meno di 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.

3.5 Random forest

Per questo caso speficico 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.

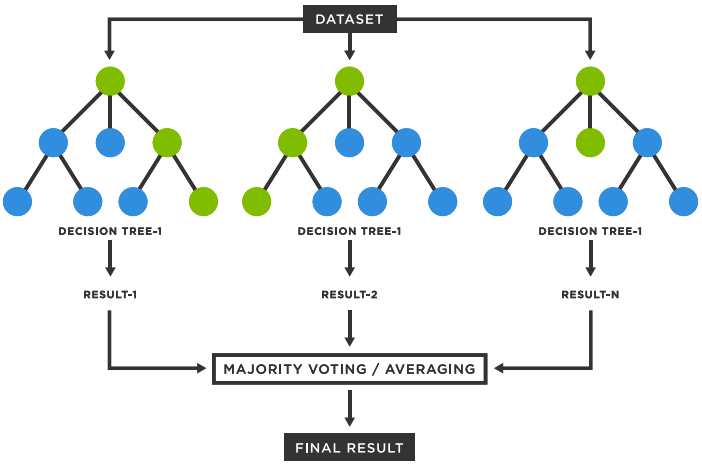

Figure 8: Esempio di random forest www.spotfire.com

3.5.1 Funzionamento

In particolare la Random Forest funziona creando un insieme di alberi decisionali, 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*). 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.

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:

1. `'n_estimators = 50`, che rappresenta il numero di alberi su cui lavorare
2. `criterion = "gini"`, che rappresenta il 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.

3.6 Valutazione dei modelli

3.6.1 Metriche scelte

4 Risultati

4.0.1 Considerazioni

5 Sviluppi futuri

Il sistema presentato è aperto a sviluppi futuri che possano rendere il sistema ancora più efficiente e all'avanguardia.
Di seguito sono descritti alcuni dei possibili sviluppi futuri che intendiamo esplorare:

1. Espansione della copertura ferroviaria andando a completare le informazioni relative a treni e staziono ed integrazione con altri servizi ferroviari (Italo, Frecciarossa, ...)
2. Implementazione di una vera e propria interfaccia grafica
3. Miglioramento della ricerca andando a suggerire all'utente, sulla base dei caratteri inseriti, le stazioni che iniziano con quei caratteri
4. Integrazione di dati in tempo reale andando a fornire all'utente informazioni in *real time*

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.