Università degli Studi di Bari

Dipartimento di Informatica

Documentazione Caso di studio

Ingegneria della Conoscenza

A.A 2023/24

FoodDelivery

Gruppo di lavoro

- Julian Pajo, 758416, pajo@studenti.uniba.it
- Dario Ranieri, 764186, d.ranieri18@studenti.uniba.it
- Emanuele Romito, 758445, e.romito3@studenti.uniba.it

Repository Github:

`https://github.com/DarioRan/ProgettoICON2024`

February 14, 2024

Contents

Introduzione **1**

1 Dataset e preprocessing **1**
1.1 Dataset utilizzati . . . 1
1.1.1 Dataset per ordini effettuati . . . 1
1.1.2 Dataset per chiusure stradali . . . 1
1.1.3 Dataset per drivers . . . 1
1.2 Preprocessing sui dataset . . . 1

2 Ontologie **3**
2.1 GraphML . . . 4

3 Knowledge Base (KB) **7**
3.1 Fatti e regole della KB . . . 7
3.1.1 KB per ordini effettuati . . . 7
3.1.2 KB per rete stradale . . . 8
3.2 Query in Prolog sulla KB . . . 8
3.3 Strumenti utilizzati . . . 10

4 Constraint Satisfaction Problem (CSP) **11**
4.1 Sommario . . . 11
4.2 Strumenti utilizzati . . . 11
4.3 Decisioni di progetto . . . 11
4.4 Validazione . . . 12

5 Ricerca sul grafo **13**
5.1 Sommario . . . 13
5.2 Strumenti utilizzati . . . 13
5.3 Decisioni di progetto . . . 13
5.3.1 Algoritmo A* . . . 13
5.3.2 Algoritmo di Dijkstra . . . 13
5.3.3 Algoritmo A* rivisitato . . . 14
5.4 Validazione . . . 14
5.4.1 Esperimento 1: 0.5 km . . . 15
5.4.2 Esperimento 2: 1 km . . . 15
5.4.3 Esperimento 3: 2 km . . . 15
5.4.4 Esperimento 4: 10 km . . . 16
5.4.5 Esperimento 5: 20 km . . . 16
5.4.6 Esperimento 6: 50 km . . . 16
5.5 Risultati . . . 16
5.6 Scelta dell'accesso alla base di conoscenza . . . 17
5.7 Conclusioni . . . 18

6 Apprendimento supervisionato **19**
6.1 Sommario . . . 19
6.2 Approccio . . . 19
6.3 Pre-Processing feature di input . . . 19

6.4 Regressore Lineare 20
6.5 Regressore Lineare con L1 20
6.6 Regressore Lineare con L2 21
6.7 Regressore Boosted 22
6.7.1 Validazione e test 22
6.8 Rete Neurale 22
6.8.1 Validazione e test 22
6.9 Confronto tra i modelli 24

7 Apprendimento con incertezza **26**
7.1 Sommario 26
7.2 Belief Newtork 26
7.3 Apprendimento della Belief Network 26
7.4 Inferenza esatta 27

8 Conclusioni **27**

9 Riferimenti Bibliografici **1**

Introduzione

L'industria del food delivery sta vivendo una crescita esplosiva, con sempre più persone che scelgono di ordinare cibo comodamente da casa anziché recarsi fisicamente nei ristoranti. Questo cambiamento nelle abitudini di consumo ha generato un'enorme quantità di dati, consentendo alle aziende di food delivery di utilizzare l'**intelligenza artificiale** per ottimizzare i propri servizi e migliorare l'esperienza complessiva del cliente. Il nostro obiettivo è stato quello di creare un **framework** che agisce come una sorta di **recommender system**. Questo sistema, ricevendo in input i cibi che l'utente desidera ordinare e il tempo massimo di attesa, produrrà diversi risultati:

- Consiglierà un ristorante che possa soddisfare l'ordine nel modo più celere possibile.
- Fornirirà una stima del tempo di consegna.
- Assegnerà l'ordine a un rider che sia il più possibile vicino al ristorante.
- Consiglierà al rider un percorso ideale per arrivare a destinazione.
- Mostrerà all'utente la probabilità che il rider incappi in un blocco stradale.

Per produrre tali risultati, è stato progettato un sistema dotato di **interfaccia grafica**, in modo tale che fosse in grado di raccogliere i dati in input forniti dall'utente tramite l'utilizzo di diverse librerie. Tramite l'utilizzo di **ontologie** come *OpenStreetMap*, le quali informzioni sono descritte mediante un grafo definito in linguaggio *GraphML*, e l'utilizzo di una **Knowledge Base** implementata attraverso *Prolog* per gestire l'accesso ai dati, è stata implementata una **ricerca su grafo**, in modo tale da fornire al rider un percorso ideale per arrivare a destinazione, con la relativa stima del tempo di consegna all'utente. Attraverso l'utilizzo dell'**apprendimento supervisionato** invece, è stato possibile implementare una sezione riguardante la stima del tempo di preparazione dei cibi indicati dall'utente tramite interfaccia. Infine, è stata sviluppata una sezione riguardante l'assegnamento dell'ordine a un determinato rider attraverso la modellazione di un **Constraint Satisfaction Problem** e il calcolo della probabilità che il rider incappi in un blocco stradale utilizzando una **Belief Network**.

1 Dataset e preprocessing

1.1 Dataset utilizzati

1.1.1 Dataset per ordini effettuati

Il primo dataset utilizzato rappresenta un'istantanea di questa realtà prendendo il caso d'uso di New York, fornendo una panoramica dettagliata degli ordini effettuati attraverso una piattaforma di food delivery. Ogni riga del dataset corrisponde a un singolo ordine e contiene diverse informazioni chiave, tra cui l'ID dell'ordine, l'ID del cliente, il nome del ristorante, il tipo di cucina ordinata, il costo dell'ordine, il giorno della settimana in cui è stato effettuato l'ordine, il rating assegnato dal cliente, il tempo di preparazione del cibo da parte del ristorante e il tempo di consegna.

1.1.2 Dataset per chiusure stradali

Nel sistema viene utilizzato un ulteriore dataset contenente informazioni riguardanti le chisure stradali a New York per valutare la probabilità di un possibile ritardo nella consegna. Il dataset generato casulamente riguarda le rilevazioni su 7 giorni. Contiene l'ora, in slot di 30 minuti, la strada e la chiusura. Le chiusure sono state generate causlamente attraverso una distribuzione poissioninana il cui parametro dipende dall'ora e dal giorno.

1.1.3 Dataset per drivers

Per l'assegnazione di un ordine a uno specifico driver, si fa ricorso a un dataset contente informazioni riguardanti i drivers, indicando ID, la loro posizione e la loro disponibilità.

1.2 Preprocessing sui dataset

Si è effettuato su tale database una procedura di preprocessing in modo tale da rendere il dataset pronto per l'analisi e l'implementazione di modelli, garantendo che vengano forniti dati completi, bilanciati e pronti per essere utilizzati nei processi decisionali e di ottimizzazione del food delivery. Operazioni di preprocess che si sono ritenute necessarie per avere un dataset completo sono state:

- **Aggiunta di Dati Casuali**: L'aggiunta di dati casuali serve a simulare nuovi ordini nel dataset. Questo è utile per aumentare la varietà dei dati e fornire più informazioni per l'analisi. Inoltre, l'aggiunta di dati casuali può aiutare a generare un dataset più bilanciato, poiché rappresenta meglio la varietà degli ordini effettuati.
- **Aggiunta di record per ogni Tipo di Cucina**: Questo passaggio è importante perché assicura che ci siano abbastanza dati per ciascun tipo di cucina rappresentato nel dataset. In un'applicazione pratica, ci sarebbero molti più ristoranti per ogni tipo di cucina, quindi è necessario garantire che il dataset contenga abbastanza informazioni per ciascun ristorante.
- **Ottenimento delle Coordinate Geografiche dei Ristoranti**: Le coordinate geografiche dei ristoranti sono cruciali per calcolare la distanza tra il ristorante e la posizione del cliente. Sono state acquisite mediante l'API di *OpenStreetMap*. Questo è fondamentale per stimare il tempo di consegna e ottimizzare le rotte di consegna dei corrieri.

- **Filtraggio dei Dati per Coordinate Entro i Limiti di New York City**: Per evitare disambiguità dovute all'errata posizione del ristorante ricavata da *OpenStreetMap*, il dataset viene filtrato includendo solo le righe in cui le coordinate del ristorante si trovano all'interno dei limiti geografici di New York City definiti in precedenza.

- **Calcolo della Posizione del Cliente**: Il dataset originario non prevedeva informazioni riguardanti la posizione del cliente. Peranto la posizione approssimata è calcolata sulla base del tempo di consegna e della velocità casuale. Questo è utile per simulare la posizione del cliente al momento della consegna del cibo e può essere utile per analizzare le prestazioni della consegna e ottimizzare le rotte dei corrieri.

- **Aggiunta delle Pietanze al Dataset**: Per ogni riga nel dataset, vengono aggiunte informazioni sulle pietanze ordinate. Ogni ordine può contenere diverse pietanze, alle quali sono associate nome e tempo di preparazione. Per diminuire la randomicità il più possibile, viene calcolata la media e la deviazione standard del tempo di preparazione per ciascun ristorante, giorno della settimana e posizione in modo da prendere in considerazione il comportamento dei singoli ristoranti in diverse situazioni. Queste informazioni vengono utilizzate successivamente per campionare il tempo di preparazione delle pietanze.

2 Ontologie

Un'**ontologia** è una rappresentazione formale e strutturata delle conoscenze di un dominio specifico, che definisce concetti, relazioni e proprietà pertinenti a quel dominio. In altre parole, è un insieme organizzato di concetti e relazioni che descrivono un determinato ambito di conoscenza.

L'ontologia utilizzata nel nostro sistema è *OpenStreetMap* (**OSM**)

OpenStreetMap è un progetto collaborativo che mira a creare e fornire dati geografici gratuiti, dettagliati e liberi a chiunque ne abbia bisogno. **OSM** contiene una vasta gamma di dati geografici, tra cui strade, sentieri, confini amministrativi, edifici, punti di interesse, percorsi ciclabili e molto altro ancora.

I dati su *OpenStreetMap* sono organizzati in una struttura gerarchica che riflette la gerarchia del mondo fisico. Ad esempio, strade e sentieri possono essere organizzati in reti stradali, edifici possono essere raggruppati in quartieri o zone.

Relazioni Semantiche: **OSM** include non solo dati geometrici (come le coordinate di un punto), ma anche dati semantici che forniscono informazioni contestuali sui luoghi. Ad esempio, un edificio può avere etichette che indicano il suo scopo (residenziale, commerciale, industriale), consentendo una rappresentazione più ricca e dettagliata del mondo reale.

Affermazioni e Assiomi Impliciti: Anche se **OSM** non è formalmente definito come un'ontologia nel senso tradizionale, contiene implicitamente affermazioni e assiomi sul mondo fisico. Ad esempio, la presenza di un edificio implica che esiste un'entità fisica corrispondente nel mondo reale.

Nel nostro sistema, **OSM** è stato utilizzato per ricavare informazioni riguardanti i ristoranti durante il preprocessing del dataset. Mediante il nome del ristorante è stato possibile ricavare le coordinate geografiche di esso.

E' stato fatto effettuando una query all'api di *OpenStreetMap* `https://nominatim.openstreetmap.org/search`

```python
def get_coordinates(restaurant_name):
    query = f'{restaurant_name} New York'
    url = 'https://nominatim.openstreetmap.org/search'
    params = {
        'q': query,
        'format': 'json',
    }
    print(f"Elaborazione del ristorante: {restaurant_name}")
    response = requests.get(url, params=params)
    data = response.json()
    if data:
        location = data[0]
        latitude = float(location['lat'])
        longitude = float(location['lon'])
        print(f"Coordinate ottenute: Latitudine={latitude}, Longitudine={longitude}")
        return latitude, longitude  # Restituisce le coordinate come tupla
    else:
        print("Coordinate non disponibili per questo ristorante.")
        return None
```

Figure 1: Utilizzo api OSM per ricavare coordinate

2.1 GraphML

GraphML è un formato di file per la rappresentazione di grafi basato su *XML* (eXtensible Markup Language). *GraphML* fornisce una sintassi standard per rappresentare grafi in modo da poterli salvare, scambiare e visualizzare in modo interoperabile tra diverse applicazioni. Nel sistema, utilizziamo un *GraphML* per rappresentare la città di New York e la sua rete stradale mediante un grafo orientato utilizzando dati presenti in OpenStreetMap.

Un grafo orientato (o grafo diretto) è una struttura matematica composta da nodi (o vertici) e archi direzionati tra di essi. In un grafo orientato, gli archi hanno una direzione, quindi è possibile spostarsi da un nodo all'altro solo seguendo la direzione degli archi.

I nodi e gli archi sono associati a delle informazioni.

Di seguito le informazioni considerate più rilevanti:

- **flowSpeed**: velocità del flusso di traffico (tipo: intero)
- **maxspeed**: velocità massima consentita sulla strada (tipo: stringa)
- **name**: nome della strada (tipo: stringa)
- **highway**: tipo di strada (tipo: stringa)
- **osmid**: identificativo OpenStreetMap della strada e nodo(tipo: stringa)
- **length**: lunghezza della strada (tipo: stringa)

- **x**: coordinata x del nodo (tipo: stringa)
- **y**: coordinata y del nodo (tipo: stringa)

Ogni nodo nel grafo sarà rappresentato nel seguente modo:

```
<ns0:node id="42805336">
  <ns0:data key="d3">40.7380121</ns0:data>
  <ns0:data key="d4">-73.8448543</ns0:data>
  <ns0:data key="d7">10W;10E</ns0:data>
  <ns0:data key="d5">42805336</ns0:data>
  <ns0:data key="d6">motorway_junction</ns0:data>
</ns0:node>
```

Figure 2: Rappresentazione nodo in GraphML

Dove:

- **id**: identificatore unico del nodo
- **ns0:data**: contiene i dati relativi al nodo

Mentre ogni arco, che va da un nodo all'altro, sarà rappresentato nel seguente modo:

```
<ns0:edge source="42496574" target="42496570">
    <ns0:data key="d10">221688408</ns0:data>
    <ns0:data key="d8">206.145989691</ns0:data>
    <ns0:data key="d12">New York Avenue</ns0:data>
    <ns0:data key="d11">residential</ns0:data>
    <ns0:data key="d9">False</ns0:data>
    <ns0:data key="d13">0</ns0:data>
  <ns0:data key="d23">11</ns0:data>
</ns0:edge>
```

Figure 3: Rappresentazione arco in GraphML

Dove:

- **source**: identificatore unico del nodo di partenza dell'arco
- **target**: identificatore unico del nodo di destinazione dell'arco
- **ns0:data**: contiene i dati relativi all'arco

Tali informazioni verranno poi filtrate e definite nella base di conoscenza mediante regole *Prolog*, consentendo di catturare relazioni più complesse e condizioni logiche. Una volta che il grafo è convertito in regole *Prolog*, diventa possibile interrogare il grafo usando la potenza del motore di inferenza *Prolog*. Questo significa che è possibile formulare query logiche complesse per estrarre informazioni, trovare percorsi, o risolvere problemi basati sul grafo.

3 Knowledge Base (KB)

3.1 Fatti e regole della KB

Una *Knowledge Base* (**KB**) è un insieme organizzato di informazioni che un programma o un sistema può utilizzare per rispondere a domande o prendere decisioni. Queste informazioni possono essere estratte da diverse fonti, tra cui documenti, database, manuali di istruzioni, siti web e altro ancora. Un elemento della base di conoscenza è un assioma. Per definire gli assiomi della **KB**, si è usato *Prolog*.

Prolog è un linguaggio di programmazione logica che si basa sul concetto di inferenza logica. *Prolog* permette di creare e interrogare una **KB** usando fatti e regole che descrivono il dominio di interesse.

Nel nostro sistema si è utilizzata la **KB** per rappresentare due realtà: gli ordini effettuati e la rete stradale.

3.1.1 KB per ordini effettuati

Le informazioni della *Knowledge Base* per gli ordini effettuati vengono estratte e filtrate dal dataset CSV degli ordini e poi rappresentate in regole *Prolog*.

Classe order: Ogni ordine è caratterizzato dai seguenti attributi:

- Nome del ristorante
- Tipo di cucina
- Costo dell'ordine
- Giorno della settimana
- Tempo totale di preparazione
- Tempo di consegna
- Posizione del ristorante
- Posizione del cliente
- Nome e tempo di preparazione delle signole pietanze.

```
% Facts

order('Hangawi','Korean',30.75,'Weekend',30,20,(40.7466917, -73.9846896),'(40.70743206982177, -74.00513535615369)',
[{'dish_name': 'Bulgogi', 'preparation_time': 18}, {'dish_name': 'Kimchi', 'preparation_time': 12}]).
```

Figure 4: Esempio di un ordine in Prolog

Classe dish: Ogni piatto è caratterizzato dai seguenti attributi:

- Tipo di cucina
- Nome del piatto

```
dish('Korean', 'Bibimbap').
dish('Mexican', 'Salsa Verde').
dish('Indian', 'Samosa').
dish('American', 'Pizza').
```

Figure 5: Esempio di una pietanza in Prolog

3.1.2 KB per rete stradale

Per quanto riguarda la *Knowledge Base* per rappresentare la rete stradale, si è convertito il grafo *GraphML* in assiomi in *Prolog*. Il modello utilizza classi per rappresentare le strade e gli incroci, con proprietà specifiche per ogni classe.

Classe node: Rappresenta gli elementi di connessione tra le strade. E' caraterrizzata dai seguenti attributi:

- id OSM
- latitutine
- longitudine

```
node('42467343', '40.655576', '-73.926888').
node('42467346', '40.655632', '-73.925953').
node('42860563', '40.7433053', '-73.9376534').
node('42467350', '40.655694', '-73.924983').
node('42991641', '40.509297', '-74.246057').
node('42860570', '40.7431934', '-73.9367078').
node('42467355', '40.655754', '-73.924014').
node('42991645', '40.509045', '-74.246903').
node('42467360', '40.6558712', '-73.9233908').
```

Figure 6: Esempio di nodo in Prolog

Classe edge: Rappresenta le strade. E' caraterrizata dai seguenti attributi:

- id nodo sorgente
- id nodo target
- lunghezza
- id OSM
- nome della strada
- velocità del flusso

```
edge('42859589', '42859586', '76.0586872811', '5704342', '221st Street', '12').
edge('42859589', '42786183', '118.956028424', '5708673', '107th Avenue', '18').
edge('42859589', '42921111', '157.590243679', '5708673', '107th Avenue', '19').
edge('42860543', '42860546', '114.648108839', '5704407', '47th Avenue', '12').
edge('42860543', '42890982', '31.5134159273', '5710227', 'Skillman Avenue', '20').
edge('42860543', '42872047', '138.142602038', '5710227', 'Skillman Avenue', '14').
```

Figure 7: Esempio di arco in Prolog

3.2 Query in Prolog sulla KB

Le query in *Prolog* servono a interrogare una base di conoscenza (**KB**) che contiene fatti e regole su un determinato dominio. Le query permettono di verificare se una certa affermazione è vera o falsa, o di trovare i valori delle variabili che la rendono vera.

Per quanto riguarda gli ordini, sono state definite le seguenti query:

```
% Queries

restaurants_by_cuisine(CuisineType, RestaurantName) :-
    order(RestaurantName, CuisineType, _, _, _, _, _, _).

dishes_by_cuisine(CuisineType, Dishes) :-
    dish(CuisineType, Dishes).

all_cuisine_types(CuisineTypes) :-
    setof(CuisineType, Dish^dish(CuisineType, Dish), CuisineTypes).

restaurant_loc_by_cuisine(CuisineType, RestaurantName, RestaurantLocation) :-
    order(RestaurantName, CuisineType, _, _, _, _, RestaurantLocation, _, _).

get_dishes_info(RestaurantName,RestaurantLocation,DayOfWeek,Dishes) :-
    order(RestaurantName, _, _, DayOfWeek, _, _, RestaurantLocation, _, Dishes).
```

Figure 8: Query in Prolog sulla KB degli ordini

- **restaurants by cuisine(CuisineType, RestaurantName) :- order(RestaurantName, CuisineType, , , , , ,).**
  Restituisce i ristoranti che offrono una certa cucina, usando il fatto order.

- **dishes by cuisine(CuisineType, Dishes) :- dish(CuisineType, Dishes).**
  Restituisce i piatti di una certa cucina, usando il fatto dish.

- **all cuisine types(CuisineTypes) :- setof(CuisineType, Dish dish(CuisineType, Dish), CuisineTypes).**
  Restituisce tutti i tipi di cucina nel database, usando la funzione setof e il predicato dish.

- **restaurant loc by cuisine(CuisineType, RestaurantName, RestaurantLocation) :- order(RestaurantName, CuisineType, , , , , RestaurantLocation, ,).** Restituisce il nome e la posizione dei ristoranti che offrono una certa cucina, usando il fatto order.

- **get dishes info(RestaurantName,RestaurantLocation,DayOfWeek,Dishes) :- order(RestaurantName, , , DayOfWeek, , , RestaurantLocation, , Dishes).** Restituisce i piatti ordinati in un certo giorno, in un certo ristorante, usando il fatto order.

Per quanto riguarda la KB della rete stradale, sono state definite le seguenti query:

```
% Queries

get_all_nodes(Nodes) :-
    findall([Osmid, Lat, Lon], node(Osmid, Lat, Lon), Nodes).

get_street_name(Source, Target, StreetName) :-
    edge(Source, Target, _, _, StreetName, _).

get_edge_length(Source, Target, Length) :-
    edge(Source, Target, Length, _, _, _).

get_neighbors(Source, Neighbors) :-
    findall(Neighbor, edge(Source, Neighbor, _, _, _, _), Neighbors).

get_edge_flowSpeed(Source, Target, FlowSpeed) :-
    edge(Source, Target, _, _, _, FlowSpeed).
```

Figure 9: Query in Prolog sulla KB della rete stradale

- **get all nodes(Nodes) :- findall([Osmid, Lat, Lon], node(Osmid, Lat, Lon), Nodes).**
  Questa query restituisce una lista di tutti i nodi nel database, usando la funzione findall e il predicato node.

- **get street name(Source, Target, StreetName) :- edge(Source, Target, , , StreetName,).**
  Questa query restituisce il nome della strada che collega due nodi, usando il predicato edge.

- **get edge length(Source, Target, Length) :- edge(Source, Target, Length, , ,).**
  Questa query restituisce la lunghezza dell'arco che collega due nodi, usando il predicato edge.

- **get neighbors(Source, Neighbors) :- findall(Neighbor, edge(Source, Neighbor, , , ,), Neighbors).**
  Questa query restituisce una lista di tutti i nodi adiacenti a un nodo, usando la funzione findall e il predicato edge.

- **get edge flowSpeed(Source, Target, FlowSpeed) :- edge(Source, Target, , , , FlowSpeed).**
  Questa query restituisce la velocità del flusso dell'arco che collega due nodi, usando il predicato edge.

3.3 Strumenti utilizzati

Per la realizzazione delle regole in *Prolog* e quindi la conversione del dataset csv, è stata impiegata la libreria *Python* **pandas** per la manipolazione dei dati. Per la lettura del grafo è stata utilizzata la libreria **xml.etree.ElementTree**. Mentre per l'accesso alla **KB** e per la sottomissione delle query si è utilizzata la classe **Prolog** della libreria *python* **pyswip** che crea un ponte tra *Python* e **SWI-Prolog** permettendo di interrogare SWI-Prolog nei nostri programmi Python.

4 Constraint Satisfaction Problem (CSP)

4.1 Sommario

Abbiamo affrontato il problema di assegnare in maniera efficiente gli ordini ai driver disponibili di un servizio di consegna utilizzando un modello di **CSP**. L'obiettivo è stato quello di minimizzare la distanza totale percorsa dai driver migliorando la soddisfazione del cliente e l'efficienza operativa.

4.2 Strumenti utilizzati

Per la realizzazione e l'utilizzo del CSP sono state impiegate le librerie *Python* **pandas** per la manipolazione dei dati e **PuLP** per la definizione e risoluzione del problema di ottimizzazione. Queste librerie forniscono gli strumenti necessari per modellare il problema e applicare l'algoritmo per la ricerca della soluzione ottimale. E' stata inoltre implementata la libreria **GeoPandas** per la visualizzazione grafica dell'assegnamento dell'ordine rispetto al driver che meglio soddisfava l'objective function.

4.3 Decisioni di progetto

Abbiamo modellato i seguenti aspetti nel nostro CSP:

- **Variabili**: I driver disponibili per le consegne.
- **Domini**: La possibilità che ciascun driver accetti o meno l'ordine, rappresentata da variabili binarie.
- **Vincoli**: Ogni driver può accettare al massimo un ordine, e l'ordine deve essere assegnato a un solo driver disponibile.
- **Objective Function**: Minimizzare la distanza totale percorsa dal driver per arrivare al ristorante.

Le variabili nel nostro CSP sono i **driver disponibili**, ciascuno rappresentato da una variabile binaria che indica se il driver è stato **assegnato** all'ordine o meno. Il dominio di queste variabili è **0, 1**, dove '1' indica che il driver è stato assegnato all'ordine e '0' che non lo è. I vincoli del nostro problema includono:

- **Vincolo di Unicità**: Assicura che un solo driver possa essere assegnato a ogni ordine.
- **Vincolo di Disponibilità**: Impedisce ai driver non disponibili di essere assegnati agli ordini.
- **Minimizzazione della Distanza**: L'objective function che calcola la somma ponderata delle distanze di ogni driver dall'ordine, moltiplicata per la variabile di assegnazione corrispondente. La soluzione ottimale del CSP è quella che minimizza questa somma totale.

Il nostro modello CSP è stato risolto utilizzando l'algoritmo *COIN-OR branch and cut*. Esso è basato sull'algoritmo del simplesso, ma combina il simplesso con molti altri algoritmi come branch-and-bound e la generazione di tagli. In particolare, per risolvere i programmi lineari, utilizza il *COIN-OR Linear Programming.*

4.4 Validazione

Tramite l'utilizzio della libreria GeoPandas, siamo riusciti a visualizzare graficamente l'assegnazione dell'ordine al driver, permettendoci di visualizzare la risoluzione del CSP ad ogni inserimento di un ordine.

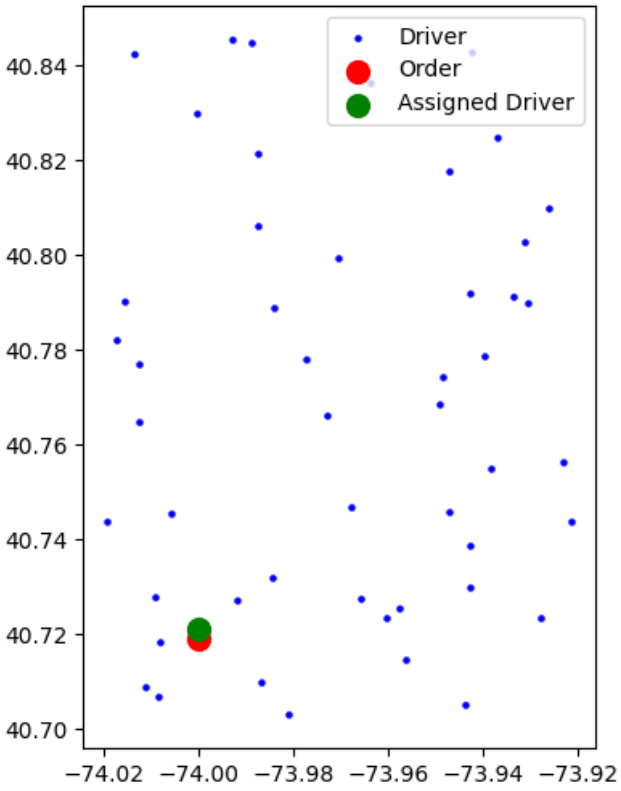

Figure 10: Plot della risoluzione del CSP

5 Ricerca sul grafo

5.1 Sommario

Abbiamo affrontato il problema della ricerca del percorso più veloce per effettuare la consegna dal ristorante al cliente mediante l'implementazione di algoritmi di ricerca sul grafo. L'obiettivo è stato quello di valutare diversi algoritmi, per determinare il migliore in termini di distanza, tempo e complessità computazionale..

5.2 Strumenti utilizzati

Per la lettura e la manipolazione del grafo *GraphML* si è usata la libreria **networkx**. Per la visualizzazione del percorso, è stata utilizzata la libreria **folium** che è in grado creazione di mappe interattive utilizzando **Leaflet.js**. Per definire la distanza tra due coordinate nella definizione dell'euristica è stata usata la libraria **geopy**. Per implementare la coda di priorità negli algoritmi di ricerca si è usata la libreria **heapq**.

5.3 Decisioni di progetto

Nella fase di ricerca sul grafo, il nostro obiettivo primario è stato individuare il percorso ottimale che minimizza il tempo di percorrenza tra ristorante e cliente.

Per raggiungere questo obiettivo, abbiamo testato tre algoritmi principali: **A***, **Dijkstra** e **A* rivisitato**. La scelta di questi algoritmi è stata motivata dalle loro caratteristiche distintive e dalla loro adattabilità al contesto specifico del nostro problema.

5.3.1 Algoritmo A*

L'algoritmo $A*$ è stato scelto per la sua capacità di combinare la completezza dell'algoritmo di ricerca di *Dijkstra* con l'efficienza dell'algoritmo di ricerca euristica. E' stata scelta come funzione euristica la distanza euclidea tra i due punti, che stima il costo rimanente per raggiungere la destinazione finale. L' $A*$, in questo modo, è in grado di dirigere la ricerca verso le aree più promettenti dello spazio di ricerca, riducendo così il tempo necessario per trovare la soluzione ottimale. Questa caratteristica lo rende particolarmente adatto per applicazioni in cui è necessario trovare rapidamente una soluzione di buona qualità, come nel nostro caso di pianificazione dei percorsi urbani. In altre parole, ci concentriamo sull'ottimizzazione della distanza fisica tra i punti di partenza e di arrivo, piuttosto che sul tempo effettivo necessario per attraversare il percorso, che potrebbe essere influenzato da variabili come il traffico, le condizioni stradali e la velocità di percorrenza.

5.3.2 Algoritmo di Dijkstra

Dijkstra è un algoritmo classico per la ricerca del cammino minimo in grafi pesati e orientati. In questo caso il costo degli archi è rappresentato dal tempo di percorrenza di essi. La sua semplicità concettuale e la garanzia di trovare il cammino minimo lo rendono una scelta naturale per la nostra ricerca. Abbiamo deciso di testare *Dijkstra* per confrontare le sue prestazioni con quelle di $A*$ e valutare se l'aggiunta di una componente euristica possa portare a miglioramenti significativi nelle prestazioni.

5.3.3 Algoritmo A* rivisitato

Tale algoritmo è stato incluso nel nostro studio per esplorare un approccio alternativo alla ricerca del cammino minimo. Come nell'*A**, questo algoritmo utilizza una *euristica* (heuristic) per stimare la distanza rimanente dal nodo corrente al nodo di destinazione. Questa euristica contribuisce a guidare l'algoritmo verso la destinazione, rendendo la ricerca più efficiente rispetto a *Dijkstra*. Si considera questa versione "rivisitata" dell'algoritmo in quanto adattata specificamente alle esigenze del nostro problema di ricerca dei percorsi urbani. Una delle modifiche principali apportate alla versione tradizionale dell'algoritmo è stata la scelta di considerare il tempo di percorrenza come costo anziché la distanza fisica.

5.4 Validazione

Per la fase di validazione, abbiamo condotto una serie di test comparativi per valutare le prestazioni degli algoritmi *A**, *Dijkstra* e la versione "rivisitata" dell'*A** in diverse condizioni di input. In particolare, abbiamo variato le distanze tra i punti di partenza e di arrivo per valutare come gli algoritmi si comportassero in scenari con differenti gradi di complessità.

Durante i test, abbiamo misurato tre metriche principali per valutare le prestazioni degli algoritmi:

Tempo di percorrenza: Questa metrica indica il tempo effettivo richiesto per percorrere il cammino ottimale individuato dall'algoritmo. Questo parametro riflette la praticità e l'efficienza del percorso suggerito, considerando la velocità di percorrenza.

Distanza di percorrenza: Misura la lunghezza fisica del percorso ottimale individuato dall'algoritmo. Questo parametro fornisce una valutazione della lunghezza del percorso e può influenzare direttamente il tempo di percorrenza effettivo.

Tempo di esecuzione dell'algoritmo: Indica il tempo necessario per eseguire l'algoritmo e trovare il percorso ottimale. Questo parametro è cruciale per valutare l'efficienza computazionale degli algoritmi e può influenzare l'applicabilità pratica dell'algoritmo in scenari reali.

Le distanze in linea d'area prese in considerazione sono:

0.5km 1km 2km 10km 20km e 50km

5.4.1 Esperimento 1: 0.5 km

Algoritmo	Distanza	Tempo di percorrenza	Tempo di esecuzione
A*	0.62 km	1 minuti e 19 secondi	0.19849 s
A* rivisitato	0.79 km	0 minuti e 48 secondi	0.21121 s
Dijkstra	0.79 km	0 minuti e 48 secondi	0.20726 s

5.4.2 Esperimento 2: 1 km

Algoritmo	Distanza	Tempo di percorrenza	Tempo di esecuzione
A*	1.28 km	2 minuti e 31 secondi	0.22590 s
A* rivisitato	1.65 km	1 minuti e 46 secondi	0.25944 s
Dijkstra	1.65 km	1 minuti e 46 secondi	0.22826 s

5.4.3 Esperimento 3: 2 km

Algoritmo	Distanza	Tempo di percorrenza	Tempo di esecuzione
A*	3.23 km	4 minuti e 7 secondi	0.31679 s
A* rivisitato	3.23 km	3 minuti e 20 secondi	0.34988 s
Dijkstra	3.23 km	3 minuti e 20 secondi	0.21607 s

5.4.4 Esperimento 4: 10 km

Algoritmo	Distanza	Tempo di percorrenza	Tempo di esecuzione
A*	14.03 km	21 minuti e 49 secondi	1.78533 s
A* rivisitato	18.12 km	15 minuti e 48 secondi	1.95569 s

5.4.5 Esperimento 5: 20 km

Algoritmo	Distanza	Tempo di percorrenza	Tempo di esecuzione
A*	16.88 km	22 minuti e 21 secondi	2.33957 s
A* rivisitato	18.16 km	18 minuti e 9 secondi	2.19555 s

5.4.6 Esperimento 6: 50 km

Algoritmo	Distanza	Tempo di percorrenza	Tempo di esecuzione
A*	41.36 km	54 minuti e 19 secondi	5.22699 s
A* rivisitato	45.53 km	41 minuti e 24 secondi	7.20840 s

5.5 Risultati

Dai risultati ottenuti, emerge che per distanze fino a 2 km, tutti e tre gli algoritmi producono risultati accettabili in termini di tempo di percorrenza. Tuttavia, per distanze superiori ai 2 km, l'algoritmo *Dijkstra* diventa inefficiente e non è in grado di fornire una soluzione entro un tempo ragionevole. Al contrario, sia l'$A*$ che la sua versione rivisitata mantengono tempi di esecuzione accettabili e forniscono risultati competitivi anche su distanze maggiori.

Pertanto, quasi a parità di tempi di esecuzione, per distanze superiori ai 2 km si consiglia di utilizzare l'$A*$ rivisitato per ottenere tempi di percorrenza minimi. La scelta è stata fatta considerando l'obiettivo prioritario di minimizzare il tempo di percorrenza.

Utilizzando la libreria **folium** siamo stati in grado di creare la mappa interattiva indicando il percorso trovato dall'algoritmo usando *Leaflet.js*.

Ristorante: Samurai Mama

Tempo di preparazione: 15 minuti

Tempo di consegna : 9 minuti

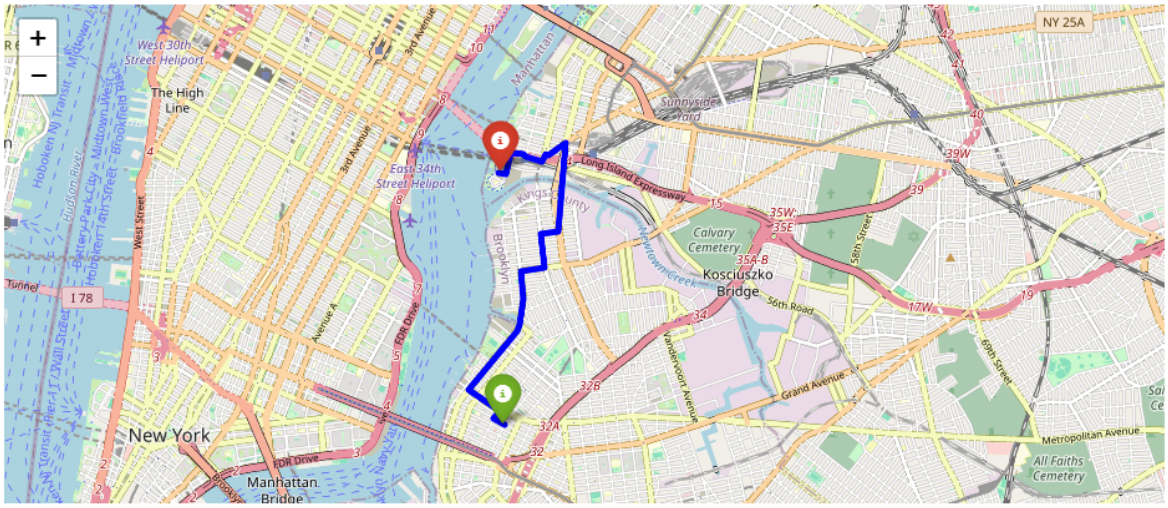

Figure 11: Percorso da ristorante al cliente trovato usando l'A* rivisitato

5.6 Scelta dell'accesso alla base di conoscenza

La scelta dell'accesso alla base di conoscenza è un elemento cruciale per garantire l'efficienza e la tempestività delle operazioni di ricerca nel contesto del progetto. Inizialmente, si è optato per l'utilizzo della base di conoscenza definita mediante assiomi in *Prolog*, poiché questo linguaggio offre una struttura solida per la rappresentazione delle conoscenze e consente di effettuare query in modo intuitivo e flessibile.

Tuttavia, con l'aumentare della complessità del grafo e il conseguente incremento del numero di nodi, si è manifestata una significativa riduzione delle prestazioni in termini di tempo di esecuzione delle query. Questo rallentamento è particolarmente evidente durante la fase di determinazione del ristorante migliore, dove è necessario applicare la ricerca del percorso verso il cliente partendo da molteplici ristoranti.

Al fine di ottimizzare le prestazioni e ridurre il tempo di esecuzione, si è deciso di adottare un'approccio alternativo, sfruttando direttamente il grafo generato dalla libreria *Networkx*. Questa decisione è stata motivata dalla vasta gamma di funzionalità e algoritmi ottimizzati offerti da *Networkx* per la ricerca e l'analisi dei grafi.

Trasferendo il grafo dal formato *GraphML* direttamente a *Networkx*, è stato possibile sfruttare le capacità di questa libreria per accelerare le operazioni di ricerca nel grafo e l'accesso alle informazioni ad esso associate. In questo modo, è stato possibile garantire un tempo di esecuzione minimo durante le operazioni critiche, come la determinazione del percorso ottimale verso il cliente da parte dei ristoranti disponibili.

Di seguito è riportato il risultato di un test in cui è stato valutato l'algoritmo *A** rivisitato utilizzando la libreria *Networkx* e l'accesso diretto alla i (KB). Questo test è stato condotto per confrontare le prestazioni dell'algoritmo considerando gli stessi punti di partenza e destinazione.

```
A* rivisitato senza accesso a KB
La lunghezza totale del percorso è: 8.89 km
Il tempo totale del percorso è: 6 minuti e 47 secondi
Tempo di esecuzione: 0.71351 secondi

A* rivisitato con accesso a KB
La lunghezza totale del percorso è: 8.89 km
Il tempo totale del percorso è: 6 minuti e 47 secondi
Tempo di esecuzione: 17.39492 secondi
```

Figure 12: Confronto A* risivitato con diverso accesso alla KB

Come è possibile notare, a parità di condizioni, con una distanza punto a punto di 5km in linea d'area, l'accesso diretto alla *Knowledge Base* rende la ricerca del percorso circa 24 volte più lenta rispetto all'utilizzo della libreria *Python*.

5.7 Conclusioni

In conclusione, dopo aver preso in considerazione i risultati degli esperimenti per valutare le prestazioni degli algoritmi di ricerca del percorso nel contesto del nostro progetto, abbiamo optato per l'utilizzo di una versione rivisitata dell'algoritmo A* rivisitato insieme all'accesso diretto al grafo mediante la libreria Python *Networkx*.

I risultati dei test hanno dimostrato che tale approccio si è dimostrato essere la scelta migliore per soddisfare le esigenze del nostro progetto, consentendo di ottenere risultati precisi e tempi di esecuzione minimi durante le operazioni di ricerca nel grafo.

6 Apprendimento supervisionato

6.1 Sommario

Una parte fondamentale del progetto riguarda la predizione dei tempi di preparazione di un ordine da parte di un ristornate così da poter predire, successivamente, il tempo totale necessario per la consegna dell'ordine. Sono stati confrontati diversi modelli:

- **Regressore Lineare**
- **Regressore Lineare con regolarizzazione L1 (Lasso)**
- **Regressore Lineare con regolarizzazione L2 (Ridge)**
- **Regressore Boosted**
- **Rete Neurale**

6.2 Approccio

L'approccio si è basato sul confrontare i **MSE** di ogni modello dopo la fase di tuning degli iperparametri. La fase di tuning prevede l'utilizzo di un algoritmo di ricerca che prende il nome di **GridSearchCV**. Dati i parametri per ogni modello e i valori che possono assumere, l'algortimo di ricerca, in funzione della dimensione k del fold per la cross validation, valuta ogni combinazione possibile dei parametri e restituisce il modello con il **MSE** più basso.

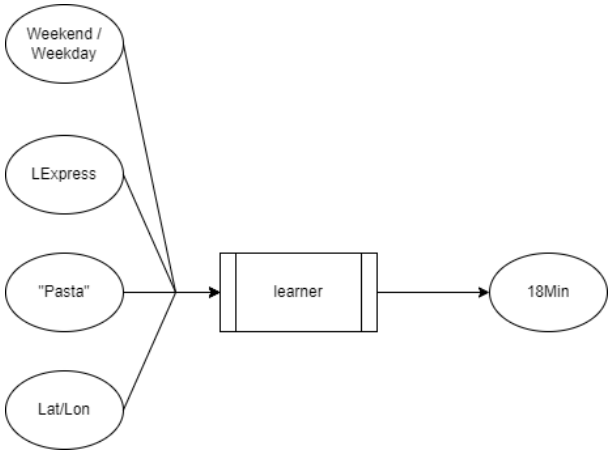

Figure 13: Approccio utilizzato per i diversi learner

6.3 Pre-Processing feature di input

Per le feature categoriche è stata prevista una trasformazione utilizzando il **OneHotEncoding**. È stato effettuato per il dish_name, restaurant_name, e weekday. Avremo che weekday, che può assumere valori 'weekend' o 'weekday', sarà rappresentato come un vettore di 2 elementi dove [1,0] corrisponde a 'weekday' e [0,1] a 'weekend'. Il problema della conversione delle feature categiriche non riguarda direttamente dei problemi con gli algoritmi di apprendimento, ma riguarda l'efficienza di questi nel lavorare con feature numeriche.

6.4 Regressore Lineare

Un **regressore lineare** è un modello che prevede che ad ogni feature di input, venga associato un peso. Il modello non è altro che la combinazione lineare tra feature di input e pesi.

L'obiettivo è quello di trovare dei parametri che minimizzino la somma dei quadrati residui tra valori predetti e valori del dataset.

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

Figure 14: Loss function regressore lineare

In questo caso abbiamo utilizzato **scikitlearn** per implementare il regressore linare, nativamente la libreria prevede l'utilizzo della *decomposizione ai valori singolari* della matrice X (o *equazione normale*). Questo algoritmo risolve analiticamente il problema di regressione a differenza dell'algoritmo di discesa del gradiente stocastica, un algoritmo iterativo di ottimizzazione che garantisce il raggiungimento di un minimo locale (e non per forza globale, questo risultato dipende dall'inizializzazione randomica dei parametri) della funzione di loss. Di solito, però, è preferibile utilizzare la SGD visto che performa molto meglio quando c'è un alto numero di feature (circa dalle 10 000 in su).

6.5 Regressore Lineare con L1

Il regressore lineare con **regolarizzazione L1** prende anche il nome di **Lasso**. Con la regolarizzazione si introduce un iperparametro α che corrisponde al tasso di penalizzazione dei parametri all'interno della funzione di loss.

$$(1/(2 * n\_samples)) * ||y - Xw||_2^2 + alpha * ||w||_1$$

La prima parte corrisponde alla RMSE, mentre la seconda parte, introdotta da alpha, la parte di regolarizzazione. La caratteristica della regolarizzazione L1 è data dal fatto che cerca di portare i parametri di feature correlate a 0. Per poter scegliere il valore migliore per alpha, è stato utilizzato il *GridSearch CV* dove è stato scelto un K pari a 5 e l'algoritmo ha ricercato il migliore modello scegliendo tra questa lista di parametri: [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.001, 0.0015, 0.002, 0.01]

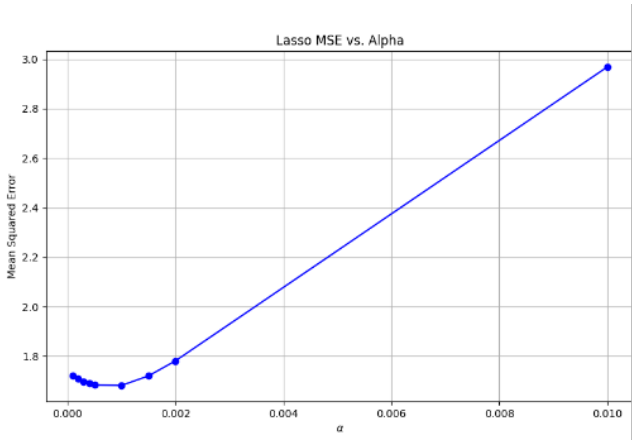

Figure 15: Andamento MSE in funzione di α

Il modello migliore è risultato essere il modello con α=0.001 con un MSE pari a circa 1.36. Un'osservazione sulla regolarizzazione L1 è data dal fatto che la L1 tende ad annulare alcuni parametri, soprattutto quelli correlati. Di conseguenza la L1 potrebbe essere utilizzata per valutare quali feature sono più importanti di altre.

6.6 Regressore Lineare con L2

Il regressore linare con **regolarizzazione L2** prende il nome di **Ridge**. Così come per il lasso, il ridge prevede un iperparametro α che penalizza i parametri. La differenza sta nella funzione da ottimizzare

$$||y - Xw||_2^2 + alpha * ||w||_2^2$$

La parte di regolarizzazione prevede che venga calcolata la norma 2 dei parametri.

Per la determinazione dell'α sono stati presi in considerazione i seguenti valori nel processo di tuning:

[0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.82, 0.84, 0.86, 0.88, 0.9, 0.92, 0.94, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0]

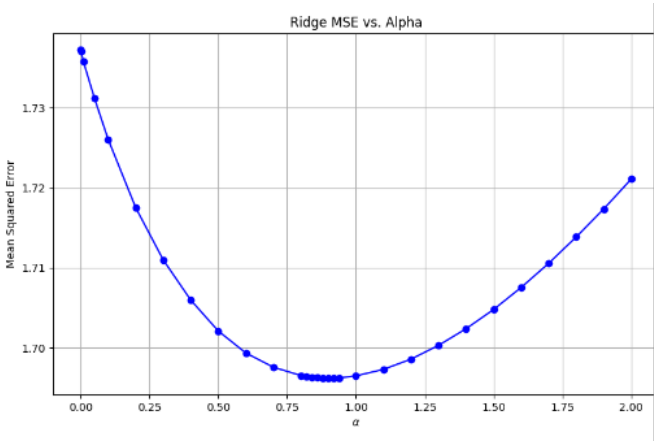

Figure 16: Andamento MSE in funzione di α

Alla fine dell'addestramento e della k cross-validation, il modello migliore è risultato quello con un α=0.9

6.7 Regressore Boosted

Un **regressore boosted** si basa sull'implementazione di una sequenza di learners che imparano dagli errori dei rispettivi learner precedenti. Il boosting del gradiente è un metodo iterativo che adatta i weak learners sui residui del modello precedente in una serie di passaggi. In ogni iterazione, il learner corrente viene migliorato focalizzandosi sugli errori del learner precedente. In particolare, è stato implementato l'**Extreme Gradient Boosting**, che consiste nell'utilizzare alberi di decisione come weak learners. In particolare, è un modello lineare di alberi decisionali con split binari. Questo significa che costruisce una serie di alberi decisionali, dove ogni albero è aggiunto per correggere gli errori del modello complessivo fino a quel punto. Per la regressione, la predizione è data dalla somma dei risultati di ciascun albero. Inoltre, viene introdotto un termine di regolarizzazione per controllare la complessità del modello, aiutando a prevenire l'overfitting. Nell'ottimizzazione del nostro modello *XGBoost*, abbiamo dato particolare attenzione ai seguenti parametri chiave:

- **Profondità massima**: Indica la profondità massima di ciascun albero di decisione usato nel processo di boosting, influenzando direttamente la complessità del modello.
- **Learning rate**
- **Numero di stimatori**: Corrisponde al numero di alberi di decisione da includere nel modello finale.
- **Subsample**: È la frazione di campioni da utilizzare per addestrare ciascun albero, utile per prevenire l'overfitting.

6.7.1 Validazione e test

Dopo la fase di validazione con tutte le combinazioni si sono raggiunti i migliori risultati con questa combinazione di parametri

- **Learning rate**: 0.2
- **Profondità massima**: 3
- **Stimatori**: 400
- **Subsample**: 0.8

6.8 Rete Neurale

Una **Rete Neurale** è composta da più layer, ogni layer è composto da una sequenza di unità. Ogni unità è composta da una combinazione lineare tra le feature di input e parametri ad essi associati. Le feature di input corrispondono agli output dei layer precedenti. Infine, la combinazione lineare, passa attraverso una funzione di attivazione, nel nostro caso un **Rectified Linear Unit** o **RLU**. Questo modello prende anche il nome di *percettrone* e una rete neurale formata da più layer di percettroni prende il nome di *Multi Layer Perceptron MLP*.

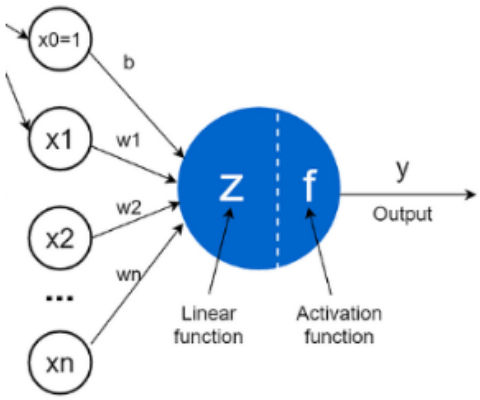

Figure 17: Esempio di unità

La rete neurale implementata prevede 3 layer:

- **Primo Layer**: in input 256 features e 128 in output
- **Secondo Layer**: in input 128 features e 64 output
- **Terzo Layer**: in input 64 features e 1 output (stima dei minuti)

Come algoritmo di ottimizzazione è stato utilizzato l'**ADAM**, *adaptive moments*, una versione modificate dell'algoritmo di discesa del gradiente stocatsica in cui viene calcolato un learning rate per ogni parametro. Ogni learning rate definisce il *momentum* cioè la velocità con cui si converge verso il punto di minimo della funzione di loss.

6.8.1 Validazione e test

Per quanto riguarda la cross-validation, è stata utilizzata una validazione hold-out in cui vengono considerate delle porzioni fisse per il training e validation set. Nel nostro caso è stato scelto un 80 percento per il trainig set e un 20 percento per il test set. Per il set di validazione è stato considerato un 20 percento dal training set. La fase di validazione consiste nel effettuare il trainig con determinati iperparametri e successivamente valutarli con il set di validazione. Si considerano gli ipeparametri che performano meglio sul validation set.

Dopo la fase di validazione con tutte le combinazioni si sono raggiunti i migliori risultati con questa combinazione di parametri

- **Learning rate**: 0.01

- **Batch Size**: 10
- **Epochs**: 500
- **Weight Decay**: 0

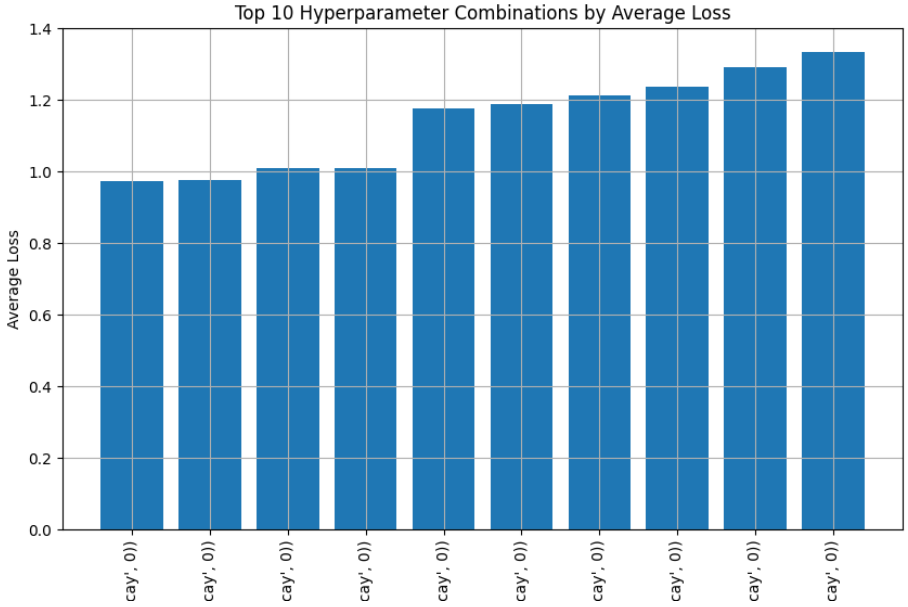

Figure 18: Top 10 assegnazioni di variabili in funzione della loss media sul set di validazione. Sfortunatamente non siamo riusciti a plottare le assegnazioni in maniera leggibile, la prima colonna da sinistra corrisponde all'assegnazione descritta sopra.

Figure 19: Andamento MSE tra train e test con i parametri migliori

6.9 Confronto tra i modelli

Per il confronto dei modelli è stato considerato il *RMSE* medio dei modelli sui dati di test e il **Bayesian Information Criterion**. Il *BIC* è un criterio di selezione dei modelli che mette in relazione le performance dello stimatore e il numero di parametri utilizzati. Maggiore sono i parametri, migliori saranno le stime, ma comporta una più alta probabilità di overfitting. Per questo motivo *BIC* introduce una penalità per modelli che utilizzano molti parametri.

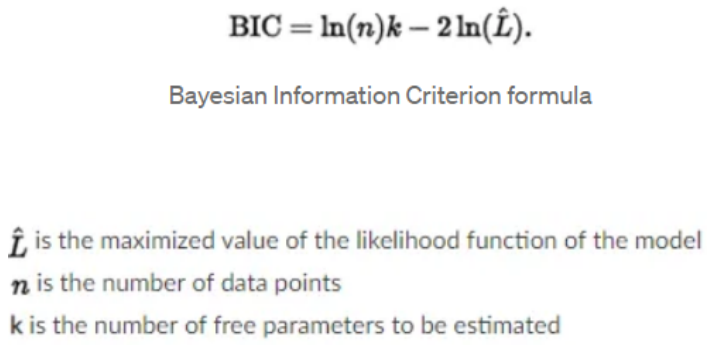

Figure 20: Formula per il calcolo del BIC

Il modello più performante è il modello con il *BIC* e il *MSE* più basso.

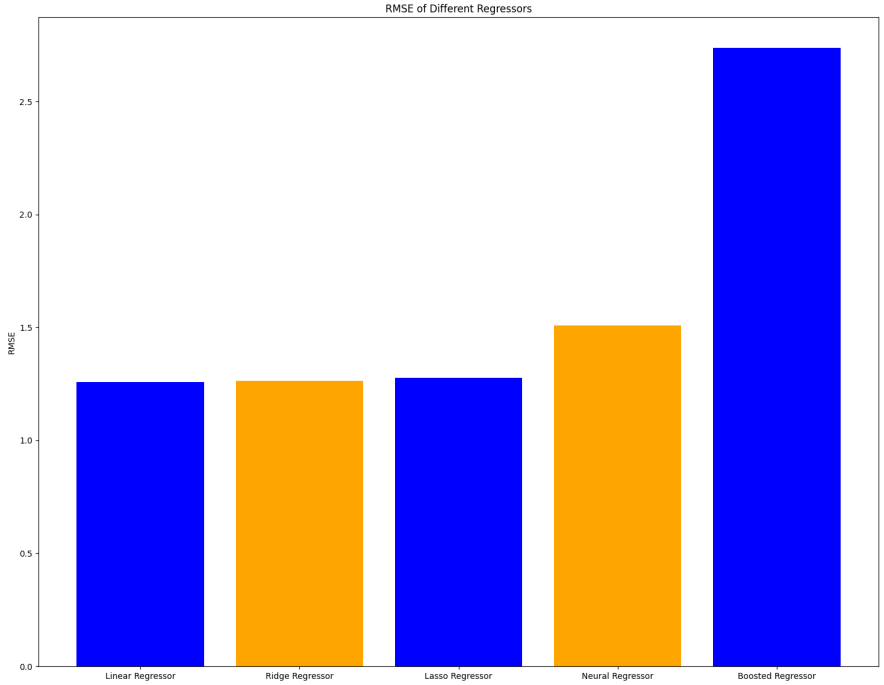

Figure 21: Comparazione RMSE sugli esempi di test dei vari modelli

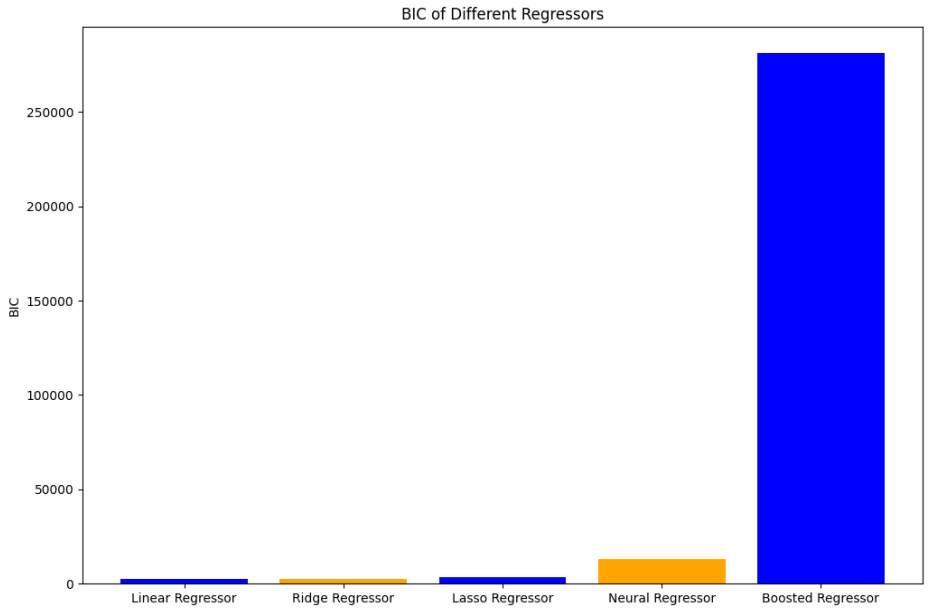

Figure 22: BIC a confronto

Da come si può vedere, sia per quanto riguarda l'errore che per il BIC, il modello migliore corrisponde al modello di *Regressione Lineare*.

7 Apprendimento con incertezza

7.1 Sommario

Abbiamo costruito un dataset con lo stato delle strade principali di New York. Una strada può essere chiusa, a causa di un blocco stradale oppure aperta. Abbiamo simulato una rilevazione per ogni 30 minuti dove, attraverso una distribuzione poissoniana, si indicava se la strada fosse chiusa o meno.

7.2 Belief Newtork

Attraverso l'utilizzo di Belief Network, prevediamo la probabilità che una strada possa essere chiusa, prevediamo la probabilità che la variabile *road_closure* possa essere *true*. La variabile dipende direttamente da *street* e *time*.

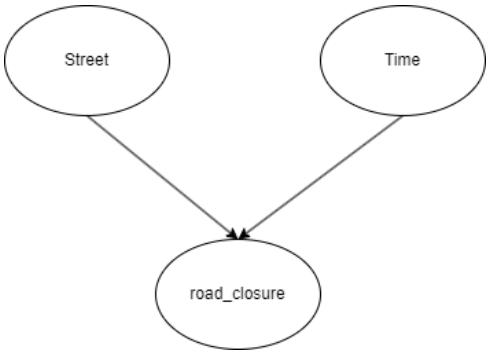

Figure 23: Struttura belief network

L'obiettivo è quello di calcolare la probabilità congiunta per la chiusura stradale con *Street* che sia uguale al nome della strada data in input e con *Time* che sia uguale all'ora attuale arrotondata alla mezz'ora più vicina. Questa operazione viene effettuata per tutti i segmenti del percorso più corto e, successivamente, si considera la probabilità più alta di incontrare un blocco stradale.

Avremo che la proabilità congiunta rappresentata dalla belief network sarà data da

$$P(road\_closure, Street, Time) = P(Street)P(Time)P(road\_closure|Street)P(road\_closure|Time)$$

Per effettuare una query, la libreria calcolerà la probabilità condizionata date le osservazioni, cioè *Street* e *Time*

$$P(road\_closure|Street = "East74thStreet" \wedge Time = "18:00")$$

7.3 Apprendimento della Belief Network

L'obiettivo è quello di apprendere le probabilità condizionate dato un dataset. Per fare ciò si può utilizzare qualsiasi algoritmo di apprendimento supervisionato e visto che le feature target sono comprese tra [0,1], è possibile utilizzare lo *stimatore di massima verosimiglianza* per calcolare le probabilità su cui si eseguiranno le query.

7.4 Inferenza esatta

La libreria *pgmpy* mette a disposizione meccanismi di inferenza esatta. In questo caso, per effettuare query, perciò effettuare inferenza, è stato utilizzato il meccanismo di *variable elimination*, una versione dell'algoritmo di *condizionamento ricorsivo*.

8 Conclusioni

Attraverso l'unione di diversi concetti, è stato possibile realizzare questa idea che presenta comunque margini di miglioramento per un possibile impiego commerciale. Il sistema è stato testato solo con dati sintetici e creati appositamente per il caso di studio. Non ci sono risultati di alcun tipo su dati reali a causa della mancanza di dataset adatti. Nel complesso, sui dati di test, il sistema ha dato risultati verosimili. Da notare il tempo e le risorse necessarie per il tuning dei parmateri, per le reti neurali sono state necessarie circa 3 ore.

9 Riferimenti Bibliografici

- https://nominatim.org/release-docs/develop/
- Tempelmeier, N., Gottschalk, S., Demidova, E. (2021). GeoVectors: A Linked Open Corpus of OpenStreetMap Embeddings on World Scale. 30th ACM International Conference on Information and Knowledge Management (CIKM 2021) https://arxiv.org/abs/2108.13092
- OpenStreetMap contributors. OpenStreetMap database [PostgreSQL via API].
- OpenStreetMap Foundation: Cambridge, UK; 2021 http://openstreetmap.org
- https://help.openstreetmap.org/questions/83255/how-do-i-cite-osm-in-an-academic-
  //wiki.openstreetmap.org/wiki/API
- Brandes, U., Eiglsperger, M., Lerner, J., Pich, C. (n.d.). Graph Markup Language (GraphML). University of Konstanz, Swiss Re1. https://cs.brown.edu/people/rtamassi/gdhandbook/chapters/graphml.pdf
- https://www.swi-prolog.org/pldoc/man?section=execquery
- GitHub - yuce/pyswip: PySwip is a Python - SWI-Prolog bridge enabling to query SWI-Prolog in your Python programs. It features an (incomplete) SWI-Prolog foreign language interface, a utility class that makes it easy querying with Prolog and also a Pythonic interface.
- https://artint.info/3e/html/ArtInt3e.Ch3.S6.html
- https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
- https://benihime91.github.io/blog/machinelearning/deeplearning/python3.x/tensorflow2.x/2020/10/08/adamW.html
- https://www.coin-or.org/Cbc/cbcuserguide.html