UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Dipartimento di Informatica

Corso di Laurea in Informatica

Incidenti stradali nel Comune di Bari

Caso di studio per l'esame di Ingegneria della Conoscenza

A.A. 2023-2024

Membri del gruppo:

REGGIO Francesco Maria (758438)

f.reggio@studenti.uniba.it

RICCI Francesco Pio (758442)

f.ricci32@studenti.uniba.it

Repository di GitHub del progetto:

https://github.com/frankcesco/IngCon24.git

Indice

Introduzione

Il caso di studio in oggetto si pone come obiettivo di svolgere una analisi su dati riguardanti gli **incidenti stradali nel Comune di Bari** negli anni 2022 e 2023. L'analisi è volta a identificare possibili correlazioni tra variabili come le condizioni meteorologiche, le caratteristiche della strada, il livello di traffico e la presenza di feriti. Utilizzando tecniche di apprendimento supervisionato e inferenze basate su logica, il progetto mira a costruire modelli predittivi a partire dai dati disponibili. Viene anche impiegato l'apprendimento non supervisionato con l'obiettivo di identificare pattern ricorrenti in diversi tipi di incidenti.

I dati principali utilizzati provengono dal portale Opendata del Comune di Bari, che mette a libera disposizione di chiunque una serie di dataset riguardanti un grande numero di problematiche legate alla città. In particolare, i dataset che sono stati selezionati per il progetto sono quelli degli incidenti stradali avvenuti sul territorio del Comune di Bari negli anni 2022 e 2023. Questi dataset mostrano in modo dettagliato, oltre ad una serie di informazioni di circostanza di ogni singolo incidenti, anche i punti geografici precisi in cui essi sono accaduti. Ciò ha permesso di corredarli con una serie di ulteriori dati di tipo geografico mediante il software QGIS. In particolare, sono stati arricchiti con i dati dei vari quartieri, presenti anch'essi sul portale Opendata, e le strade scaricate da OpenStreetMap.

I dati provenienti da queste fonti sono stati successivamente puliti e correlati tra loro in fase di pre-processing e con i dati in formato CSV risultanti è stata creata una Knowledge Base (KB) in linguaggio Prolog. Utilizzando i fatti nella KB, è stato possibile definire una serie di regole logiche per fare inferenza sulla base di conoscenza.

Successivamente, i dati così rielaborati sono stati impiegati nella feature selection, per calcolare quali sono le feature più significative su cui eseguire i task di classificazione booleana. La classificazione tenterà di prevedere il valore della feature booleana "lesioni", che indica la presenza di feriti in un incidente. Per poter trovare il modello migliore, sono stati addestrati modelli diversi di apprendimento supervisionato e le performance risultanti saranno confrontate per determinare il modello che si adatta meglio ai dati di addestramento.

Infine, i dati sono stati sottoposti ad un'analisi delle componenti principali (PCA), che riduce la dimensionalità del dataset mantenendo le informazioni più rilevanti e misura la varianza spiegata da ciascuna componente principale. Quindi, è stato applicato l'algoritmo K-Means, un metodo di clustering non supervisionato che cerca di identificare gruppi distinti all'interno dei dati, suddividendoli in cluster basati sulla loro similarità, con l'obiettivo di classificare gli incidenti tipi ben definiti.

Per poter eseguire nell'ordine corretto tutti i passaggi, è stato scritto appositamente il file "python/main.py" che esegue, uno dopo l'altro, tutti gli script necessari per la creazione della KB, il suo aggiornamento con le regole di inferenza e le operazioni necessarie per l'apprendimento supervisionato e non supervisionato.

Dati utilizzati

Come accennato in precedenza, i dati impiegati all'interno del caso di studio provengono da due fonti principali. È stato consultato il portale Opendata del Comune di Bari per i dati riguardanti gli incidenti stradali e i quartieri mentre i dati sulle strade sono stati prelevati da OpenStreetMap.

Incidenti

I dataset "Incidenti stradali anno 2022" e "Incidenti stradali anno 2023" sono stati scaricati dalle rispettive pagine sul portale Opendata e sono stati raccolti dal Corpo di Polizia Locale e dalla Protezione Civile. Sono disponibili in formato CSV, XML o JSON e includono i seguenti campi per ogni record di incidente.

Campo	Significato
Data dell'incidente	Data in formato GG/MM/AAAA, rappresenta la data in cui l'incidente è avvenuto.
Latitudine	La coordinata geografica che indica la posizione nord-sud dell'incidente.
Longitudine	La coordinata geografica che indica la posizione est-ovest dell'incidente.
Centro abitato	Booleano, specifica se l'incidente è avvenuto all'interno di un centro abitato o meno.
Tipo di strada	Descrive la classificazione della strada dove è avvenuto l'incidente (ad esempio strada urbana, strada provinciale entro l'abitato, statale).
Caratteristiche della strada	Informazioni aggiuntive sulle caratteristiche fisiche della strada (come presenza di curve, intersezioni, semafori).
Stato del fondo stradale	Descrive le condizioni del manto stradale al momento dell'incidente (ad esempio asciutto, bagnato, ghiacciato).
Pavimentazione	Tipo di superficie stradale (come asfaltato, lastricato, con buche).
Condizioni meteorologiche	Le condizioni del tempo al momento dell'incidente (ad esempio sereno, pioggia, neve, nebbia).
Condizioni del traffico	Descrive il livello di traffico al momento dell'incidente (normale, scarso, intenso).
Danni a cose	Booleano, specifica se ci sono stati danni materiali a seguito dell'incidente.
Incidente con lesioni	Booleano, indica se l'incidente ha causato lesioni ad almeno una delle persone coinvolte.
Orario di invio della pattuglia	L'ora in cui è stata inviata la pattuglia della polizia al luogo dell'incidente.

Campo	Significato
Orario di arrivo della pattuglia	L'ora in cui la pattuglia della polizia è arrivata sul luogo dell'incidente.

Tabella 1: Campi all'interno del file CSV sugli incidenti disponibile sul portale Opendata del Comune di Bari.

Rete stradale

In OpenStreetMap viene utilizzata la chiave "highway" come chiave principale per etichettare le strade. Il valore del campo "highway" è assegnato in base all'importanza relativa della strada in maniera standardizzata e indipendente dalla classificazione amministrativa o dall'ente di manutenzione della stessa, basandosi sulle caratteristiche della stessa. Essendo dati di tipo geografico, possono essere utilizzati per svolgere una serie di operazioni come verrà analizzato di seguito.

All'interno del programma QGIS, è stato impiegato il plug-in QuickOSM per eseguire una query con la quale stati scaricati in formato vettoriale ESRI Shapefile tutti gli elementi che presentano il campo "highway" presenti a Bari, indipendentemente dal valore salvato in esso, con tutti gli altri campi associati all'elemento. Tra gli altri campi, gli elementi considerati come degni di nota per il contesto del presente caso di studio sono i campi "maxspeed", che indica la velocità massima ammessa nel tratto di strada considerato, "oneway", che indica se il tratto di strada è a senso unico o meno, e "lanes", indicante il numero di corsie disponibili.

Quartieri

I dati geografici relativi ai 17 quartieri di Bari sono stati scaricati dal portale Opendata, dalla sezione del Sistema Informativo Territoriale, in formato vettoriale ESRI Shapefile. Questi dati possono essere elaborati attraverso il software QGIS per condurre analisi geografiche, inclusa l'operazione di pre-processing per integrarli con il dataset degli incidenti stradali.

Il file DBF correlato al dataset include vari ID e i nomi dei quartieri, fornendo le informazioni essenziali necessarie per il contesto del nostro caso di studio. La scelta di utilizzare i 17 quartieri come livello geografico è stato un compromesso tra il livello amministrativo troppo dettagliato e granulare dei civici, e quello troppo grande e poco dettagliato delle 7 circoscrizioni o degli attuali 5 municipi. Inoltre, sono stati trovati i dati sulla popolazione residente riguardanti all'ultimo censimento disponibile del 2011, per i 17 quartieri, che possono essere utilizzati per svolgere ulteriori analisi.

Pre-processing dei dati

Nella fase di pre-processing dei dati sono state eseguite tutte le operazioni necessarie per la pulizia di dati ridondanti o non utili per lo scopo del progetto, una semplice gestione di alcuni dati mancanti, e le operazioni necessarie per l'integrazione dei dati provenienti dai tre diversi dataset.

Per quanto riguarda le **strade**, come prima operazione è stata svolta una intersezione tra le strade scaricate con il plugin QuickOSM e il poligono dei quartieri, affinché ogni segmento di strada sia effettivamente incluso solamente nel territorio del Comune di Bari. Quindi, per ridurre il numero significativamente elevato di segmenti stradali (13402 elementi con tag "highway" presenti a Bari) è stato deciso di rimuovere le strade non utili per lo scopo del nostro caso di studio, quindi si è deciso di lasciare solamente le strade pubbliche che possono essere occupate e attraversate dalle auto, in quanto il dataset include tutti gli incidenti stradali che coinvolgono autovetture.

Per fare ciò, sono stati rimossi tutti i segmenti per i quali il valore del campo "highway", che ricordiamo essere categorico, sia "bridleway", "construction", "corridor", "cycleway", "elevator", "footway", "living_street", "path", "pedestrian", "proposed", "razed", "rest_area", "services", "steps" e NULL. Ciò ha permesso di ridurre il numero di segmenti a 9302. A questo punto, sono stati aggiunti gli attributi della geometria con l'apposito strumento in QGIS. Siccome il tipo di dato per le strade è un vettore, è stato aggiunto solamente un campo riguardante la lunghezza del vettore espressa in metri e computata mediante un calcolo ellissoidico.

Come ultima operazione di pre-processing sui dati delle strade, è stato deciso di gestire alcuni dati mancanti in questa fase poiché la loro natura dipende direttamente dalla classificazione dei dati in OpenStreetMap. In particolare, si assume che tutti i segmenti di strada che abbiano un valore NULL per il campo booleano "oneway", impostato a "yes" se una strada è a senso unico e "no" se è a doppio senso di marcia, siano in realtà a doppio senso di marcia perché è il valore supposto per default. Nelle rotonde, definite dal campo "junction" impostato a "roundabout", il campo "oneway" è assente, mentre per lo scopo del progetto è stato impostato su "yes".

Per i dati mancanti nel campo "lanes", che indica il numero di corsie dei tratti di strada, è stato assunto che, se è assente l'informazione, il numero di corsie è una. Inoltre, si è voluto assegnare ai tratti di strada con campo "name" impostato a NULL il valore del campo "full_id" per poter eventualmente fare analisi su interi tratti di strada che condividono quindi lo stesso nome. Quindi, si è deciso di pulire le feature poco utili per lo scopo del progetto, lasciando solamente come feature "full_id", "highway", "name", "oneway", "maxspeed", "lanes", "length". Tra questi, "maxspeed" è l'ultima feature che include valori nulli, che verranno gestiti con una assunzione di mondo aperto successivamente.

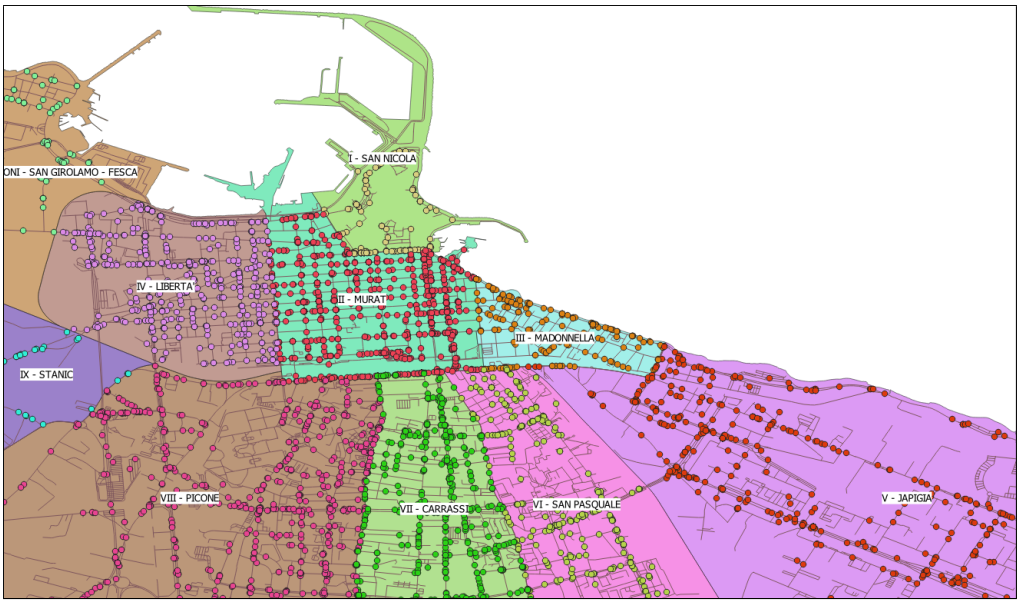

Figura 1: Schermata dal programma QGIS. I punti rappresentano gli incidenti, le linee sono le strade mentre i poligoni rappresentano i quartieri. I punti sono colorati in base al quartiere di appartenenza.

Per quanto riguarda i dati sugli **incidenti**, le operazioni di pre-processing hanno riguardato principalmente l'associazione con i dati delle strade di OpenStreetMap e i quartieri, e la pulizia di eventuali rumori nei dati. Inizialmente, sono stati caricati i dati nel programma QGIS come layer, a partire dai campi delle coordinate geografiche di latitudine e longitudine. Quindi, è stato utilizzato lo strumento di intersezione tra il layer degli incidenti e i poligoni dei quartieri per rimuovere eventuali punti che erano erroneamente fuori dal comune di Bari.

Successivamente, è stato utilizzato l'algoritmo Nearest Neighbors per trovare i segmenti di strade di OpenStreetMap più vicini ai punti degli incidenti, impostando come numero massimo di vicini uno solo, il più prossimo. L'algoritmo utilizza la distanza cartesiana per svolgere i calcoli, e unisce gli attributi delle strade più vicine agli attributi dei punti. Per gli elementi con due o più vicini equidistanti, si è utilizzato lo strumento "rimuovi duplicati" in Microsoft Excel, in modo tale che si lascia arbitrariamente solo il primo elemento non ripetuto. Delle nuove feature associate ai punti degli incidenti, solamente la feature corrispondente all'ID del segmento della strada è stato mantenuto.

L'algoritmo, inoltre, fornisce come feature aggiuntive "nearest_x" e "nearest_y" che corrispondono alle coordinate del punto più vicino sui vettori delle strade trovato dall'algoritmo. Per favorire una maggiore integrazione e coerenza dei dati, si è deciso di utilizzare queste ultime due variabili come variabili geografiche dei punti, al posto di quelle originarie. Inoltre, per rendere l'informazione sullo stato del fondo stradale al momento dell'incidente più significativa, si è creata la

feature booleana “asciutto” che sarà settata a vera se lo stato del fondo stradale presentava il valore “Asciutto”, mentre sarà falsa per tutti gli altri casi.

Infine, sono state associate le informazioni relative ai quartieri mediante lo strumento “Join attributes by location”, un algoritmo che estende gli attributi del layer di base collegandolo con predicati di tipo geografico, nel nostro caso “within” con il layer di join. Quindi, per ogni punto degli incidenti che si trovava all’interno di un poligono corrispondente ad un quartiere specifico, sono stati aggiunti gli attributi del quartiere corrispondente. Di queste nuove feature, è stato deciso di lasciare solamente la feature dell’ID del quartiere.

I dati sui **quartieri**, invece, sono stati ripuliti da piccoli poligoni “spezzettati” che risultavano come elementi separati e successivamente sono stati associati gli attributi della geometria. Di questi, è stato mantenuto solamente il campo “area”, espressa in metri quadrati e computata anch’essa con un calcolo ellissoidico. È stato inoltre creato il campo “pop2011”, che rappresenta il numero totale di residenti nei singoli quartieri secondo il censimento del 2011, l’ultimo per il quale sono disponibili informazioni dettagliate. In questo modo, si potrà calcolare la densità di popolazione e vedere se può influenzare in qualche modo il tasso di incidenti.

La fase di pre-processing dei dati è stata completata, e i dati risultanti sono stati esportati come file CSV denominati “incidenti.csv”, “strade.csv” e “quartieri.csv”. Questi file sono disponibili nel repository del progetto su GitHub all’interno della cartella “data”. Verranno successivamente utilizzati per costruire la Knowledge Base (KB) su cui verrà svolta l’analisi.

Creazione della Knowledge Base

La Knowledge Base è scritta in linguaggio Prolog e consiste di una serie di fatti riguardanti le tre classi di individui, ovvero gli incidenti, le strade e i quartieri, e di una serie di regole per inferire conoscenza a partire dai fatti esistenti. Sarà quindi impiegata la libreria "pyswip", che si interfaccia con SWI-Prolog e permette di lavorare con il linguaggio Prolog direttamente all'interno dell'ambiente Python, per svolgere gli opportuni calcoli sulla KB.

Le classi di individui selezionate per il caso di studio sono le seguenti:

- incidente(I): identifica l'incidente il cui numero sequenziale è I.
- strada(S): identifica la strada il cui id completo di OpenStreetMap è S.
- quartiere(Q): identifica il quartiere il cui codice è Q.

Fatti

Per creare i fatti della Knowledge Base, è stato sviluppato uno script in Python apposito. Questo script utilizza la libreria "csv" per la gestione e l'analisi dei file CSV e la libreria "os" per la gestione dei percorsi dei file all'interno del file system. Lo script legge, analizza e trasforma i file CSV generati durante la fase di pre-processing, preparandoli per l'importazione nella KB.

La funzione principale dello script, "generate_prolog_kb", prende in input i file CSV contenenti informazioni sugli incidenti, sulle strade e sui quartieri. Lo script rimuove eventuali apostrofi che potrebbero creare problemi di formato con i file Prolog, sostituendoli con spazi e trasforma "si" e "no" in interi che rappresentano i valori booleani. Quindi, la funzione scrive nel file di output la dichiarazione di predicati dinamici, per permettere successive operazioni su di essi. Poi, apre i file e genera i fatti in formato Prolog salvati in un file chiamato "facts.pl". Questo file rappresenta l'output della funzione e contiene tutte le informazioni strutturate necessarie che potranno quindi essere caricate in memoria usando la funzione "consult" per poter svolgere le operazioni successive.

Si è optato per l'uso di predicati personalizzati anziché generici al fine di garantire una rappresentazione più compatta ed efficiente dei dati. Questa scelta è particolarmente importante dato l'elevato numero di fatti presenti nella Knowledge Base, che ammontano a 15600. A differenza dei predicati generici, che richiederebbero un fatto di tipo "prop" per ogni proprietà associata a ciascun elemento, i predicati personalizzati sono più concisi.

Ad esempio, invece di avere molteplici fatti generici come "prop('22', data, '04/01/2022')", "prop('22', nearest_y, 41.1312951658515)", e così via, possiamo utilizzare un predicato personalizzato come "incidente('22', '04/01/2022', 1, 'Strada urbana', 'Rettilineo', 1, 'Asfaltata', 'Sereno', 'Normale', 0, 0, 15, 20, 'w30009725', 16.8677559036262, 41.1312951658515, '1')" per rappresentare tutte queste informazioni in un unico fatto. La stessa cosa vale per i fatti riguardanti strade e quartieri.

Per la modellazione del dominio, pertanto, sono state identificate le seguenti feature per le rispettive classi di individui. Per gli individui incidenti, si possono riassumere le feature nella tabella sottostante.

Incidenti	
Nome della feature	**Significato**
id	Codice identificativo, sequenziale, dell'incidente.
data	Data nella quale si è registrato l'incidente.
abitato	Booleana, indica se l'incidente è avvenuto nel centro abitato.
tipo	Categorica, tipo di strada dove è avvenuto l'incidente.
carattistiche	Categorica, indica le caratteristiche della strada.
asciutto	Booleana, specifica se il manto stradale è asciutto o no al momento dell'incidente.
pavimentazione	Categorica, indica il tipo di pavimentazione della superficie stradale.
meteo	Categorica, indica le condizioni meteorologiche al momento dell'incidente.
traffico	Categorica, con relazione d'ordine. Indica le condizioni del traffico al momento dell'incidente.
danni_cose	Booleana, indica se nell'incidente sono avvenuti danni materiali a cose.
lesioni	Booleana, indica se nell'incidente sono presenti feriti. È la feature obiettivo del task di classificazione.
ora	Intero, rappresenta l'ora di chiamata alle forze dell'ordine.
minuto	Intero, rappresenta il minuto di chiamata alle forze dell'ordine.
strada	ID della strada di OpenStreetMap corrispondente.
nearest_x	Longitudine della posizione dell'incidente, posizionato sulla strada.
nearest_y	Latitudine della posizione dell'incidente, posizionato sulla strada.
quartiere	ID del quartiere in cui è avvenuto l'incidente.

Tabella 2: Feature inerenti agli incidenti nella KB appena creata.

Seguono ora nella tabella seguente le feature corrispondenti alle strade. Di tutte le feature originariamente presenti nel dataset di OpenStreetMap, sono state mantenute le seguenti.

Strade	
Nome della feature	**Significato**
full_id	ID del tratto di strada, generato da OpenStreetMap.

Strade	
highway	Categorica, indica il tipo di strada in base all'importanza.
name	Nome della strada a cui i segmenti fanno riferimento.
oneway	Booleana, indica se la strada è a senso unico.
maxspeed	Indica il limite di velocità nel tratto di strada corrispondente, in km/h.
lanes	Indica il numero di corsie del tratto di strada.
length	Indica la lunghezza espressa in metri del tratto di strada.

Tabella 3: Feature inerenti alle strade nella KB appena creata.

Per quanto riguarda i quartieri di Bari, sono definiti dalle seguenti feature.

Quartieri	
Nome della feature	**Significato**
id	Codice identificativo del quartiere.
name	Nome del quartiere di riferimento.
area	Area, espressa in metri quadrati, del quartiere.
pop2011	Numero di residenti permanenti nel quartiere secondo il censimento generale del 2011.

Tabella 4: Feature inerenti ai quartieri di Bari nella KB appena creata.

Dopo l'esecuzione della funzione "generate_prolog_kb", è stato quindi generato il file "facts.pl" che contiene i fatti relativi agli incidenti, alle strade e ai quartieri mediante predicati personalizzati. Il numero di fatti totali memorizzati all'interno del file "facts.pl", che si riferisce alla KB prima di qualsiasi modifica successiva tramite regole di inferenza, è di 15600. Un esempio è il seguente.

```
incidente('6330', '31/12/2023', 1, 'Strada urbana', 'Rettilineo', 1, 'Asfaltata', 'Sereno', 'Scarso', 0, 0, 10, 55, 'w383966841', 16.8585992384979, 41.0932623500249, '8').
incidente('6331', '31/12/2023', 1, 'Strada urbana', 'Intersezione semaforizzata', 1, 'Asfaltata', 'Sereno', 'Scarso', 0, 1, 12, 45, 'w27535688', 16.8339396479974, 41.1293688462256, '10').
incidente('6332', '31/12/2023', 1, 'Strada urbana', 'Incrocio', 1, 'Asfaltata', 'Sereno', 'Scarso', 0, 0, 15, 10, 'w25585825', 16.8722785131346, 41.1133895639094, '7').
incidente('6333', '31/12/2023', 1, 'Strada urbana', 'Intersezione semaforizzata', 1, 'Asfaltata', 'Sereno', 'Normale', 0, 0, 15, 55, 'w30015974', 16.8594716402886, 41.128190791343, '4').
incidente('6334', '31/12/2023', 1, 'Strada urbana', 'Incrocio', 1, 'Asfaltata', 'Sereno', 'Normale', 0, 0, 16, 45, 'w25654677', 16.866699847247 8, 41.1099214519428, '8').
strada('w14347631', 'secondary', 'Viale Mercadante', 1, null, 1, 43.293622290888692).
strada('w24884016', 'tertiary', 'Piazza Giuseppe Garibaldi', 1, null, 1, 310.509675841758565).
strada('w24884032', 'residential', 'Via Carlo Perrone', 1, null, 1, 110.0758597848833654).
strada('w1289643935', 'unclassified', 'w1289643935', 0, null, 1, 126.206898053143476).
strada('w1289643936', 'unclassified', 'Via Della Felicità', 0, null, 1, 191.877366887614670).
strada('w1289643937', 'unclassified', 'Via Della Felicità', 0, null, 1, 26.250326035405013).
strada('w1292425109', 'residential', 'Largo Nicola Nitti Valentini', 1, null, 1, 27.733913532162127).
quartiere('1', 'I - SAN NICOLA', 815125.800650641205721, 5667).
quartiere('2', 'II - MURAT', 1463167.932754594366997, 21115).
quartiere('3', 'III - MADONNELLA', 503783.474222141725477, 10259).
quartiere('4', 'IV - LIBERTA ', 1728359.618829930899665, 35841).
quartiere('5', 'V - JAPIGIA', 15627563.350708808749914, 29870).
quartiere('17', 'XVII - TORRE A MARE', 5117090.348814829252660, 5108).
```

Figura 2: Figura che rappresenta una porzione del file facts.pl, contenente i fatti in Prolog relativi alle tre classi di individui: incidente, strada e quartiere.

Regole

Le regole nel linguaggio Prolog sono uno strumento utilizzato per l'inferenza logica all'interno della base di conoscenza. Permettono quindi di definire relazioni

e implicazioni tra i fatti memorizzati, consentendo al sistema di ottenere nuove informazioni derivandole logicamente da quelle già presenti.

Per quanto riguarda il caso di studio, sono state definite una serie di regole che permettono di svolgere delle inferenze dalla base di conoscenza e quindi ad ottenere nuove informazioni a partire dai fatti memorizzati, come le feature ingegnerizzate che riguardano la densità di popolazione e la fascia oraria di un incidente. Tramite la funzione "consult", vengono caricati in memoria i fatti definiti precedentemente, prelevandoli dal file "facts.pl" nella cartella "prolog", che così saranno disponibili per essere utilizzati durante l'esecuzione del programma. Le nuove informazioni derivante dalle inferenze vengono quindi integrate nella base di conoscenza attraverso la funzione "assertz".

Le regole definite nel progetto sono progettate per consentire il ragionamento logico su diversi aspetti degli incidenti stradali. Alcune regole sono state impiegate per svolgere calcoli sui dati mancanti, come spiegato successivamente, mentre altre vengono utilizzate per la creazione di feature ingegnerizzate.

Le seguenti regole sono salvate all'interno del file "rules.pl" nella cartella "prolog" ed inferiscono conoscenza derivata mediante la creazione di nuove feature. L'obiettivo è stato quello di ingegnerizzare delle feature che potessero effettivamente fornire conoscenza in più per il task di classificazione, ovvero, vedere come la fascia oraria o la densità di popolazione del quartiere in cui si è svolto l'incidente possa effettivamente influenzare la presenza di incidenti con feriti.

- **fascia_oraria**. Regola che determina in modo categorico la fascia oraria in cui è avvenuto l'incidente. È una feature ingegnerizzata associata agli incidenti e può assumere i valori "notte", "mattina", "pomeriggio", "sera" calcolati in base all'orario.
- **densita_quartiere**. Regola che calcola la densità di popolazione di un quartiere dividendo la popolazione del quartiere per la sua area. È aggiunta ai fatti sui quartieri.

Per quanto riguarda i dati mancanti, la feature "maxspeed" associata alle strade presenta una proporzione molto elevata di osservazioni nulle, il 90.0%, ovvero 8376 elementi con valori nulli sui 9302 totali. In questo caso, è stato deciso di gestire i valori nulli con una assunzione di mondo aperto. In contrasto con l'assunzione di conoscenza completa o di mondo chiuso, utilizzata per esempio nella gestione dei dati mancanti nelle feature "oneway" in cui si è assunto che ogni atomo è falso se non è vero, nell'assunzione di mondo aperto si cerca di imputare i valori mancanti utilizzando le distribuzioni osservate dei valori esistenti.

In particolare, per ogni tipo di strada denotati dai diversi valori del campo "highway", viene calcolata la distribuzione dei valori di "maxspeed" conosciuti, e da questa si calcola la **media ponderata**. Come peso, viene utilizzata la feature "lenght" che indica la lunghezza in metri del tratto di strada. Quindi, si calcola una media ponderata per ogni valore assunto dalla feature "highway" e sarà assegnata al posto del valore nullo all'interno del campo "maxspeed". In

particolare, la formula della media ponderata μ_c per la categoria c con n elementi non nulli è la seguente. Notiamo che l_i indica il valore di "lenght" per l'elemento i, mentre s_i rappresenta il valore di "maxspeed".

$$\mu_c = \frac{\sum_{i=0}^{n} l_i \cdot s_i}{\sum_{i=0}^{n} l_i} \tag{1}$$

Ci potrebbero essere alcune categorie di strade per le quali non esiste alcun valore di "maxspeed" assegnato. In tal caso, si potrebbe gestire, in modo generale, assegnando il valore con la frequenza più alta in tutte le strade, quindi usando la moda dei valori di "maxspeed". Per il contesto del caso di studio, l'unica categoria per la quale non è associata alcuna velocità massima è "track", utilizzata principalmente per catalogare le strade di campagna non asfaltate. Per questo motivo, ci è sembrato ragionevole associare un valore di 30 km/h per questo tipo di strade. Per il resto delle strade, è stato eseguito il calcolo usando l'assunzione di mondo aperto, come descritto precedentemente.

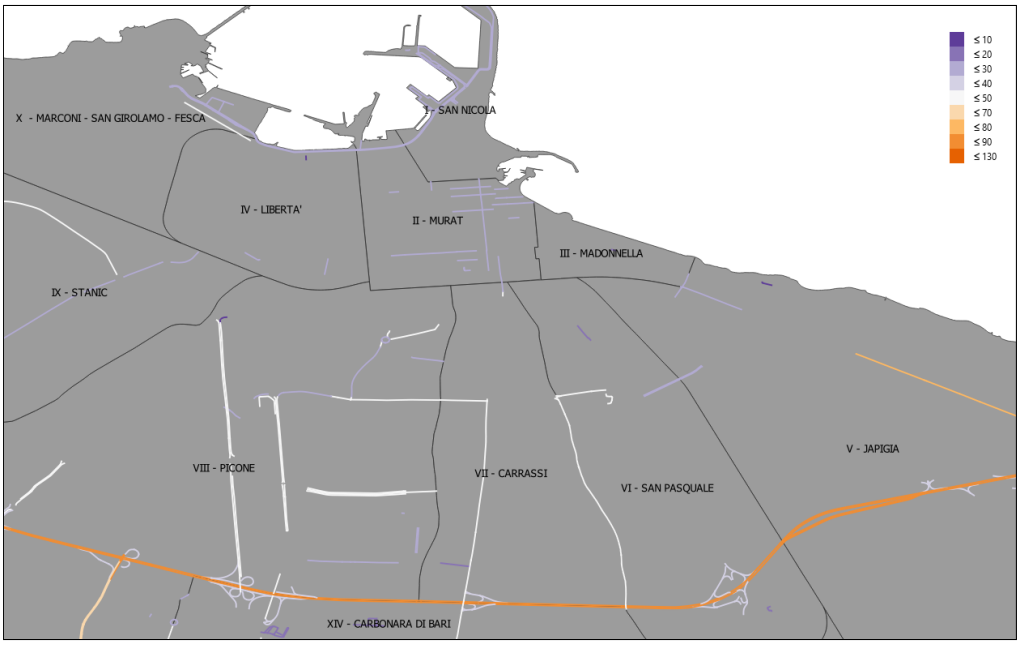

Figura 3: Valori del campo "maxspeed" associato alle strade prima del calcolo di inferenza.

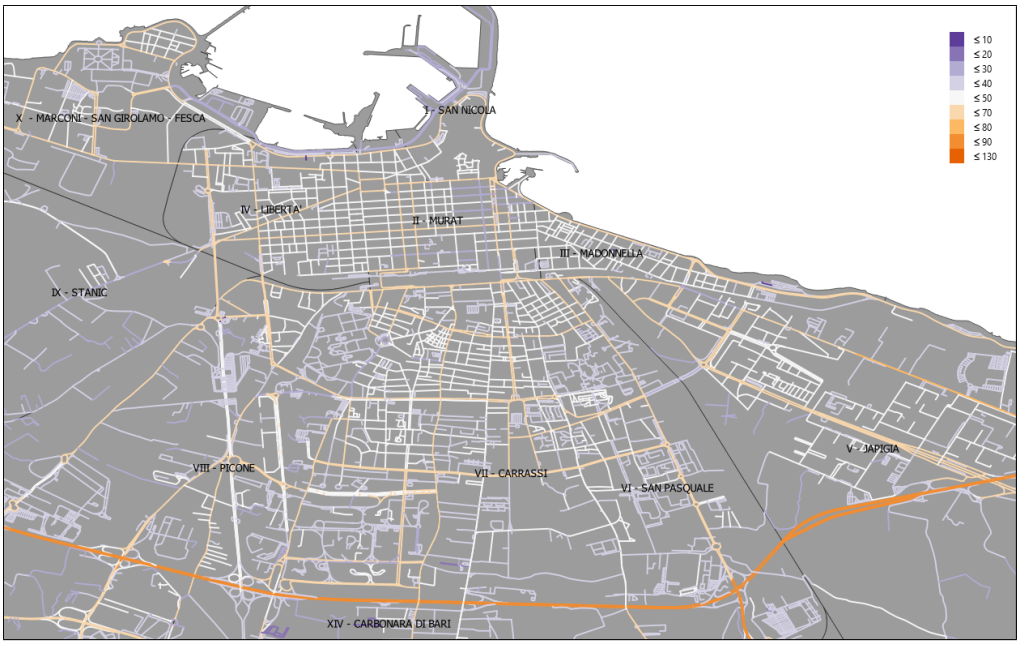

Figura 4: Valori del campo “maxspeed” associato alle strade dopo il calcolo di inferenza.

Nelle immagini precedenti si possono visualizzare i valori del campo “maxspeed” per i singoli tratti di strada prima e dopo l’assunzione di mondo aperto calcolata precedentemente. Ogni colore diverso rappresenta un intervallo di valori diverso. Nell’immagine a sinistra, prima dell’assunzione, la maggior parte dei segmenti delle strade hanno un valore nullo per la velocità massima consentita e quindi non sono colorati.

Per poter effettivamente computare l’assunzione, sono state scritte in Prolog una serie di **regole di inferenza** all’interno del file “data\rules.pl” che vengono chiamate in ordine dalla regola “assign_speeds/0”. In particolare, la prima regola che viene invocata è “assign_track_speed/0” che imposta a 30 il valore del campo “maxspeed” negli elementi “highway” di tipo “track”. Successivamente, viene eseguita la regola “calculate_speed_mean/0” che trova le varie categorie uniche di strade e invoca la regola “calculate_mean_for/1”. Quest’ultima calcola e salva temporaneamente come fatto la media pesata della velocità per ogni categoria di strada, trovando tutti i valori non nulli per “highway” e calcolando la media pesata in base alla lunghezza dei segmenti.

Infine, tramite la regola “assign_missing_speeds/0” vengono assegnati i valori di “maxspeed” mancanti basandosi sulla media pesata precedentemente calcolata. In questo modo, ogni strada con “maxspeed” nullo riceve la media pesata calcolata per la specifica categoria di strada, salvata nel fatto “speed_mean”. In una versione precedente, si usavano i valori effettivi proporzionato con la lunghezza per calcolare una distribuzione di probabilità da usare per assegnare casualmente

i valori. Questo si è rivelato però instabile in quanto i valori assegnati cambiavano ad ogni esecuzione del progetto.

Per poter eseguire le regole in Prolog è stata utilizzata la libreria "pyswip" che permette di eseguire le query dell'ambiente SWI-Prolog direttamente dal programma Python. È stato scritto il file "update_kb.py" che utilizza la libreria "pyswip" per eseguire le regole citate in precedenza. Lo script carica in memoria la base di conoscenza salvata nei file "facts.pl" e "rules.pl" e tramite il comando "prolog.query" esegue la query "assign_speeds".

A questo punto, tutti i valori mancanti sono stati gestiti. Verrà salvata la base di conoscenza aggiornata nel file "updated_facts.pl" ed eventuali fatti rimanenti con valori nulli residui, per qualsiasi campo, saranno semplicemente eliminati. Solamente due elementi presentavano ancora valori nulli.

Per quanto riguarda le **feature ingegnerizzate**, sono state calcolate nuove variabili come la densità di popolazione e la fascia oraria, come citate in precedenza. Queste nuove feature sono state calcolate mediante inferenze sulla Knowledge Base utilizzando Prolog. Lo script "apply_inference_rules.py" prende in input la KB modificata da "update_kb.py", che rimuove i valori nulli e calcola l'assunzione di mondo aperto su "maxspeed", ed esegue le regole di inferenza salvate "rules.pl" che calcolano la fascia oraria dell'incidente in base alla feature "ora" e la densità di popolazione del quartiere.

Queste due nuove feature saranno quindi aggiunte rispettivamente ai fatti sugli incidenti e sui quartieri, per completare la base di conoscenza, i cui fatti sono ora salvati in "prolog\final_facts.pl". Inoltre, saranno aggiornate le dichiarazioni di predicati dinamici per rispecchiare il numero incrementato di argomenti. La base di conoscenza è ora completa e si può passare alla fase successiva.

Apprendimento supervisionato

Nel contesto del presente caso di studio, è stato adottato l'**apprendimento supervisionato** per affrontare un problema di classificazione. L'obiettivo è cercare di predire se un incidente abbia o meno dei feriti, utilizzando feature di input selezionate dai singoli punti degli incidenti. La feature target oggetto della valutazione è la variabile booleana "lesioni", che indica la presenza o l'assenza di feriti coinvolti nell'incidente. Pertanto, si tratta di un task di classificazione booleana.

Nella fase di feature selection si analizzano tutte le feature disponibili e vengono selezionate le feature più significative per il task di classificazione booleana. Successivamente, bisognerà valutare le prestazioni di una serie di modelli diversi di apprendimento supervisionato per poter scegliere quello che si adatta meglio al presente task di classificazione.

Selezione delle feature

La scelta delle feature di input è stata guidata dal principio di migliorare la generalizzazione, selezionando solo quelle che potrebbero essere di effettivo utilizzo, ovvero quelle maggiormente correlate alla feature target selezionata, la feature booleana "lesioni". La scelta deve essere svolta sia tra le feature già presenti nella Knowledge Base, che tra feature derivate da inferenze e calcoli sulla KB. Tra le feature già presenti, quelle che si suppone essere particolarmente utili per il task di classificazione, ovvero quelle che effettivamente potrebbero influenzare la probabilità che ci siano lesioni, sono le feature che includono informazioni su condizioni meteorologiche e di visibilità, caratteristiche delle strade e livelli di traffico. Pertanto, si è deciso di scartare a priori tutte le feature non utili alla classificazione, quali quelle inerenti a identificatori, nomi, date, ore e posizioni geografiche.

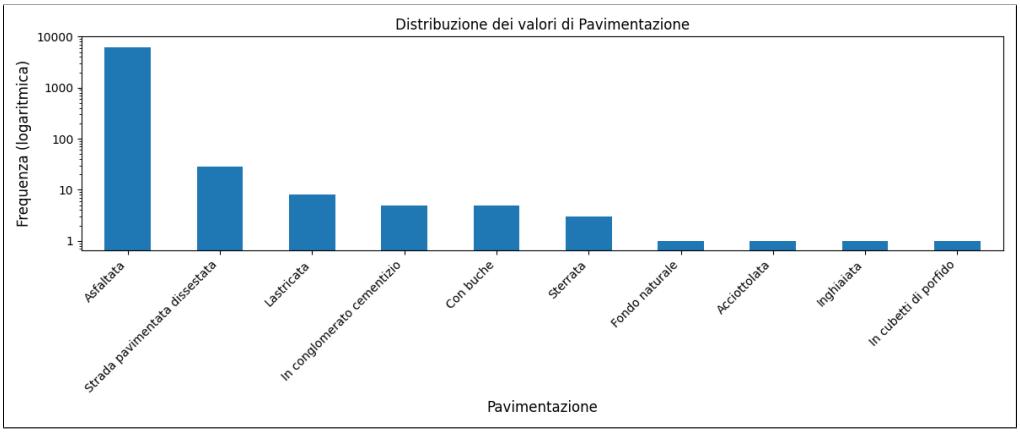

Figura 5: Distribuzione dei valori associati alla feature "pavimentazione" relativa agli incidenti.

La feature categorica “pavimentazione”, che descrive la pavimentazione del manto stradale nel punto dell’incidente, non è significativa al task di classificazione in quanto è al 99.14% su “asfaltata”, ovvero 6280 elementi su 6334 totali. È possibile vedere nel grafico precedente la frequenza, logaritmica per mostrare anche i valori più bassi, dei valori associati alla feature “pavimentazione”.

Una valutazione che occorre fare è decidere se mantenere o meno la feature booleana “danni_cose”, che indica se sono avvenuti danni materiali a cose nel corso dell’incidente. Siccome la feature target selezionata per la previsione, “lesioni”, indica la presenza di feriti durante un incidente, queste due feature potrebbero essere correlate, poiché incidenti che causano danni materiali potrebbero anche causare lesioni alle persone coinvolte. Questa correlazione, se troppo elevata, potrebbe influenzare la capacità del modello di apprendimento supervisionato di fare previsioni accurate sulle lesioni.

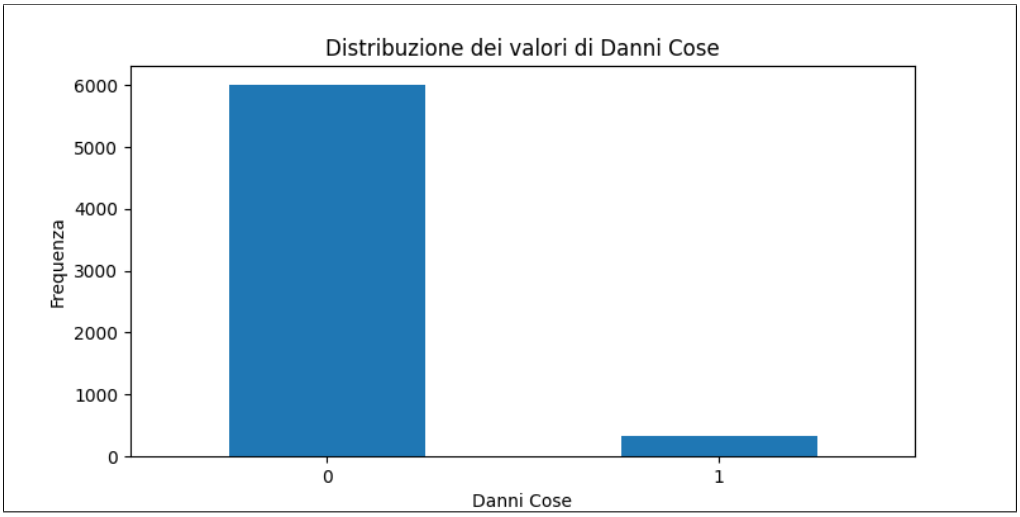

Figura 6: Distribuzione dei valori associati alla feature booleana “danni_cose”.

In particolare, se la correlazione tra "danni_cose" e "lesioni" è così alta da suggerire una relazione lineare o una dipendenza diretta tra le due feature, conviene rimuovere la prima feature. Esattamente il 5.05% degli incidenti nel dataset ha “si”, equivalente al valore 1 dopo la conversione, alla feature “danni_cose”. Ovvero, esattamente 320 osservazioni su 6332 sono positive.

All’interno della cartella “python” è stato creato lo script “correlation.py” per svolgere le relative analisi di correlazione tra le due feature booleane. Inizialmente, è stata generata la tabella di contingenza per valutare il numero di incidenti che hanno e non hanno causato danni materiali e lesioni, in base al valore assunto dalle due feature corrispondenti.

		Lesioni	
		No	Sì
Danni a cose	No	$a = 3297$	$b = 2715$
	Sì	$c = 174$	$d = 146$

Tabella 5: Tabella di contingenza per le feature “danni_cose” e “lesioni”.

Da questa tabella si può vedere che ci sono 3297 incidenti senza danni materiali né lesioni, 2715 con lesioni ma senza danni materiali, 174 con danni materiali ma senza lesioni, e 146 con entrambi. Ciò suggerisce che non ci sia una eccessiva dipendenza tra le due variabili, ma comunque è necessario verificarlo con metodi statistici. Per fare questa decisione, sono stati impiegati il test del chi-quadrato e l'indice di Jaccard.

Il test del χ^2, ovvero **chi-quadrato** o chi-quadro, viene utilizzato per determinare se esiste una dipendenza significativa tra due variabili categoriali. Il chi-quadrato misura la discrepanza tra le frequenze osservate nella tabella di contingenza e le frequenze attese sotto l'ipotesi di indipendenza. La formula per il test del chi-quadrato è la seguente, dove O_{ij} è la frequenza osservata nella cella (i,j) mentre E_{ij} è la frequenza attesa nella cella (i,j).

$$\chi^2 = \sum \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{2}$$

L'ipotesi nulla H_0 è che le variabili siano indipendenti. Il p-value associato al chi-quadrato indica la probabilità di ottenere un chi-quadrato così estremo, o più estremo, se l'ipotesi nulla fosse vera.

Nel nostro caso, il valore del chi-quadrato è 0.02656 e il p-value associato 0.8705. Un p-value elevato che supera un livello di significatività α prefissato, per convenzione $\alpha = 0.05$, indica che non c'è evidenza sufficiente per rifiutare l'ipotesi nulla di indipendenza tra le due variabili. Pertanto, non c'è una dipendenza statistica significativa tra “danni_cose” e “lesioni”.

Per quanto riguarda l'**indice di Jaccard**, viene utilizzato per misurare la similarità tra due set. Si calcola come il rapporto tra il numero di elementi in comune e il numero di elementi totali unici. Nel contesto delle variabili booleane "danni_cose" e "lesioni", l'indice di Jaccard viene calcolato con la formula seguente, in cui i valori a, b, c, d vengono presi dalla tabella di contingenza presentata in precedenza. I valori del dominio dell'indice di Jaccard sono compresi nell'intervallo [0,+1] in cui 0 indica valori completamente diversi mentre +1 indica valori identici.

$$Jaccard = \frac{d}{b + c + d} \tag{3}$$

Usando $b = 2715$, $c = 174$, $d = 146$ della tabella di contingenza, l'indice di Jaccard ottenuto è pari a 0.04810. Questo valore suggerisce una bassa similarità tra gli incidenti con danni materiali e quelli con lesioni, confermando quanto già è stato evidenziato con il test del chi-quadro.

Dall'analisi dei due metodi di correlazione, possiamo concludere che non esiste una forte correlazione tra "danni_cose" e "lesioni". Il test del chi-quadrato indica che non c'è una dipendenza statistica significativa tra le due variabili, mentre l'indice di Jaccard suggerisce una bassa similarità tra gli incidenti con danni materiali e quelli con lesioni. Pertanto, si può ritenere che la feature "danni_cose" non influenzi significativamente la capacità del modello di prevedere "lesioni" e può essere mantenuta nella successiva fase di feature selection senza compromettere le prestazioni del modello.

A questo punto, sono state identificate tutte le feature candidate per l'apprendimento e quindi bisognerà utilizzare un approccio statistico per determinare quelle maggiormente correlate alla probabilità di lesioni da effettivamente mantenere nella fase di addestramento.

All'interno della cartella "python", è presente lo script "feature_selection.py". In una prima fase, lo script estrai i fatti prolog da "final_facts.pl" e li carica in formato DataFrame della libreria "Pandas". Quindi, rinomina le colonne relative ai dati delle strade e dei quartieri per garantire l'univocità dei nomi e unisce ai DataFrame degli incidenti i dati delle strade e dei quartieri corrispondenti utilizzando l'identificativo correlato. I dati unificati saranno quindi esportati in un file CSV dal quale verranno successivamente rimossi campi non utili come ID o nomi di elementi, e verrà applicata la codifica one-hot, in cui ogni valore di una feature categorica viene trasformato in una feature booleane, utilizzando la funzione "get_dummies" della libreria "Pandas", prima di dare in pasto i dati all'algoritmo di feature selection.

L'algoritmo individuato per il task di feature selection è **BorutaPy**, un algoritmo che si basa su un approccio di tipo wrapper per identificare le caratteristiche più significative di un dataset. Questo metodo è progettato per lavorare con i classificatori basati su alberi di decisione, come il RandomForestClassifier, e ha come obiettivo quello di identificare le feature che hanno un impatto reale sulla feature target, nel nostro caso "lesioni", escludendo quelle irrilevanti. Funziona creando delle "shadow feature", ovvero versioni casuali delle feature originali. L'algoritmo addestra il classificatore su un dataset che include sia le feature reali che quelle shadow e poi confronta l'importanza delle feature originali con quella delle shadow. Se una feature originale risulta significativamente più importante di qualsiasi shadow, viene considerata rilevante e selezionata.

Uno dei parametri utilizzati dal selettore è il **percentile**, che determina il livello di importanza che una feature deve raggiungere per essere considerata significativa, ed è impostato di default a 100. Utilizzando il 100° percentile, però, solamente una feature viene selezionata, un numero decisamente insufficiente per la classificazione. Di conseguenza, si è reso necessario valutare il percentile

corretto da utilizzare mediante la valutazione di un eventuale plateau da un grafico.

In particolare, è stato codificato il file "percentile.py" che esplora diversi valori di percentile partendo da 5° fino ad arrivare al 100° ad incrementi graduali di 5 e valuta il numero di feature selezionate. Da questi dati, lo script genera un grafo con il numero di feature selezionate da Boruta sull'asse y e il percentile sull'asse x. Bisogna valutare se esiste un plateau nei dati, ovvero una zona in cui il numero di feature selezionate non cambia aumentando il percentile. Se invece l'andamento è strettamente decrescente, bisognerà cambiare strategia. Il grafico in output è stato il seguente.

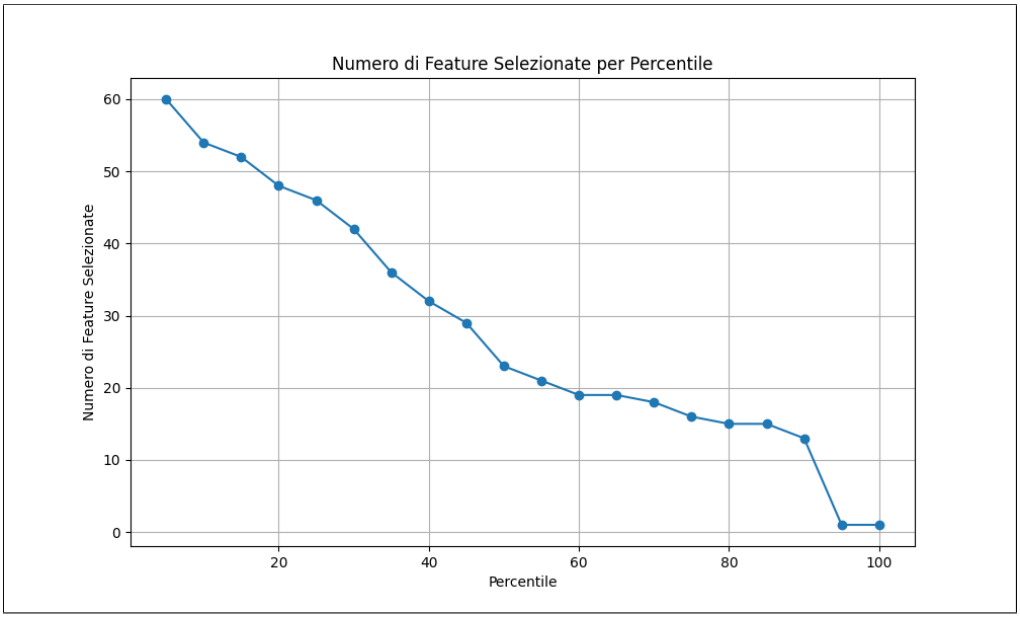

Figura 7: Grafico che mostra come si evolve il numero di feature selezionate da Boruta in base a diversi livelli di percentile, con valori $x = 5, 10, 15, \ldots, 85, 90, 95, 100$.

Sebbene non sia presente un chiaro e netto plateau nel grafico, si può dire che l'andamento decrescente rallenta a partire dal 60° percentile fino ad arrivare al 90°, ultimo valore prima di un quasi azzeramento del numero di feature, che scende ad 1. Per questo motivo, è stato selezionato come valore notevole il 90° percentile in quanto è l'ultimo valore prima del quasi azzeramento del numero di feature mantenute da Boruta, con un numero di feature corrispondente pari a 13. Questo è stato quindi selezionato come valore ottimale da usare come parametro per Boruta, che quindi ha prodotto i seguenti risultati, salvati nel file "data/selected_features.csv". Notiamo come i risultati potrebbero cambiare, sebbene di poco, a seconda dell'esecuzione. Inoltre, da notare come la feature "danni_cose" non è stata mantenuta dall'algoritmo perché ritenuta non sufficientemente rilevante.

Figura 8: Contenuto del file "selected_features.csv", che indica le feature più rilevanti selezionate dall'algoritmo BorutaPy, al 90° percentile. La prima riga è l'intestazione.

Come possibile notare dall'immagine sopra, le feature che sono legate maggiormente alla feature target booleana "lesioni" sono diverse, e provengono in modo indistinto sia dalle feature proprie dei dati sugli incidenti, sia da quelle derivate dai quartieri e dalle strade. La fase di feature selection è ora completata, e si può passare alla fase di addestramento ed esecuzione del modello di classificazione booleana.

Scelta e addestramento dei modelli

Una volta selezionato il numero appropriato di feature, bisogna svolgere una serie di valutazioni atte a trovare il modello migliore per il task di classificazione. Per fare ciò, è stato scritto "python/supervised.py" che svolge tutte le operazioni di apprendimento supervisionato utilizzando la libreria "Scikit-learn". Inoltre, vengono impiegate anche altre librerie quali "Matplotlib" per i grafici, "NumPy" per le operazioni di algebra lineare e "Pandas" per i DataFrame.

Lo script prende in input il dataset combinato con le tutte le feature candidate alla feature selection, e legge le 13 feature risultanti dai calcoli di BorutaPy spiegati in precedenza, che erano state salvate all'interno del file "data/selected_features.csv". Quindi, converte le feature categoriche in numeriche utilizzando la codifica one-hot e il risultato è salvato in un DataFrame. A partire dal DataFrame, si utilizzeranno le 13 feature selezionate da Boruta come insieme di feature di input X, mentre "lesioni" sarà la feature target y.

A questo punto, lo script dividerà il dataset in training set, o insieme di addestramento, e test set, o insieme di validazione. È stata impostata la grandezza dell'insieme di validazione a 0.2. Quindi, il 20% degli esempi sarà utilizzato per la validazione mentre l'80% sarà impiegato per l'addestramento dei modelli. Il parametro "random_state" è stato impostato 42 a fini di riproducibilità dei risultati.

I modelli di apprendimento supervisionato che sono stati selezionati per il confronto nel task di classificazione booleana sono quattro, ognuno con caratteristiche ben definite, e sono elencati di seguito. Inoltre, è stato scelto per ogni

modello presentato un insieme di iperparametri diversi da valutare accuratamente per trovare la combinazione migliore.

- **Random Forest**: Un ensemble di alberi di decisione, particolarmente utile per catturare relazioni non lineari. Gli iperparametri testati includono il numero di alberi, "n_estimators", con valori 100, 200, e 500, e la funzione per selezionare il numero di feature ad ogni suddivisione, "max_features", con opzioni "sqrt" e "log2".
- **SVC (Support Vector Classifier)**: Modello di apprendimento supervisionato utilizzato per problemi di classificazione, che può gestire casi in cui i dati non sono linearmente separabili attraverso l'uso di un kernel. Gli iperparametri considerati sono il parametro di regolarizzazione "C" con valori 0.1, 1, e 10, e il tipo di kernel con scelte tra "linear" e "rbf".
- **KNN (K-Nearest Neighbors)**: Metodo di classificazione che non assume una distribuzione specifica dei dati ma si basa sulle distanze tra i punti. L'iperparametro principale è il numero di vicini, "n_neighbors", con valori presi in considerazione 3, 7 e 21.
- **Regressione Logistica**: Modello lineare generalizzato utilizzato per problemi di classificazione binaria, particolarmente efficace quando la variabile dipendente è booleana. L'iperparametro testato è il parametro di regolarizzazione "C", con valori valutati 0.1, 1, e 10.

Per ottimizzare le performance di ciascun modello, è stata condotta una ricerca approfondita degli iperparametri utilizzando la funzione "GridSearchCV" della libreria "Scikit-learn". Questo approccio consiste nella valutazione sistematica di diverse combinazioni di iperparametri per ogni modello, con l'obiettivo di identificare la configurazione che massimizza le performance secondo la metrica selezionata. In particolare, la ricerca degli iperparametri è stata effettuata su una griglia predefinita di valori, che include vari parametri specifici per ciascun modello, definiti in precedenza.

Per garantire una maggiore robustezza e affidabilità dei risultati ottenuti, è stata utilizzata la tecnica di validazione incrociata ripetuta, nota come **Repeated K-Fold**. Questo metodo ha previsto la suddivisione del dataset in 5 suddivisioni e la ripetizione del processo di validazione 100 volte. Ogni volta, il dataset viene suddiviso in modo diverso e ogni fold viene utilizzato sia per l'addestramento che per la validazione in modo iterativo. In questo modo, è stata prodotta una stima più precisa della performance del modello, riducendo la variabilità dovuta alla casualità nella suddivisione dei dati.

Durante la validazione incrociata, è stata scelta l'**accuratezza**, o "accuracy", come metrica principale per valutare le performance dei modelli. L'accuratezza, in particolare, misura la percentuale di classificazioni corrette rispetto al totale delle classificazioni effettuate, e viene massimizzata per identificare il miglior modello. Per ciascun modello, il valore medio dell'accuratezza è stato calcolato

come la media delle accuratezze ottenute in tutte le iterazioni della Repeated K-Fold Cross Validation.

Inoltre, è stata calcolata la **deviazione standard** dell'accuratezza per ciascun modello per valutare la variabilità dei risultati. In questo modo è stato possibile ottenere non solo una misura della performance media, ma anche una valutazione della consistenza e della stabilità del modello attraverso le diverse suddivisioni e ripetizioni.

Per ogni modello sarà anche calcolata la **curva ROC**, "Receiver Operating Characteristic", e l'AUC, "area sotto la curva", che permettono di valutare la capacità di discriminazione del modello misurando il miglioramento rispetto ad un modello casuale. La curva ROC mostra la performance di un classificatore binario ed è tracciata mettendo in relazione il tasso di veri positivi (True Positive Rate) e il tasso di falsi positivi (False Positive Rate) a vari livelli di soglia. L'AUC rappresenta l'area sotto questa curva e fornisce una misura del potere discriminante del modello. Un valore AUC di 0.5 indica una prestazione casuale, mentre un AUC di 1 indica una perfetta capacità di distinguere tra le classi.

Analisi dei risultati

Una volta completata la fase di addestramento e ottimizzazione dei modelli, il passo successivo è l'analisi dei risultati ottenuti per poter quindi selezionare il miglior modello tra quelli testati. Per fare ciò, bisogna interpretare le metriche di performance che abbiamo raccolto durante la validazione.

Dopo aver completato l'addestramento e l'ottimizzazione dei modelli, è fondamentale analizzare i risultati ottenuti per identificare il modello migliore. Questa analisi si basa su due metriche principali: l'accuratezza, definita da media $\bar{x}$ e deviazione standard ρ, e l'area sotto la curva ROC. I risultati prodotti sono stati i seguenti.

Modello	**Configurazione migliore**	**$\bar{x}$ e ρ della accuratezza**	**Area sotto la curva ROC**
RandomForest	"max_features": 'sqrt', "n_estimators": 500	$\bar{x} = 0.5241$, $\rho = 0.0002$	0.5101
SVC	"C": 0.1, "kernel": 'rbf'	$\bar{x} = 0.5444$, $\rho = 0.0041$	0.5538
KNN	"n_neighbors": 21	$\bar{x} = 0.5337$, $\rho = 0.0041$	0.5211
LogisticRegression	"C": 1	$\bar{x} = 0.5561$, $\rho = 0.0000$	0.5463

Tabella 6: Prestazioni dei modelli classificatori in termini di accuratezza media e deviazione standard e ROC AUC, usando le feature selezionate da BorutaPy.

Valutiamo ora singolarmente i risultati ottenuti per comprendere quale modello è migliore tra quelli analizzati e se, in generale, le prestazioni del task di classificazione rispecchiano le aspettative iniziali. Di seguito, oltre ai risultati precedenti, notiamo in figura anche le relative curve ROC di ogni modello di classificazione.

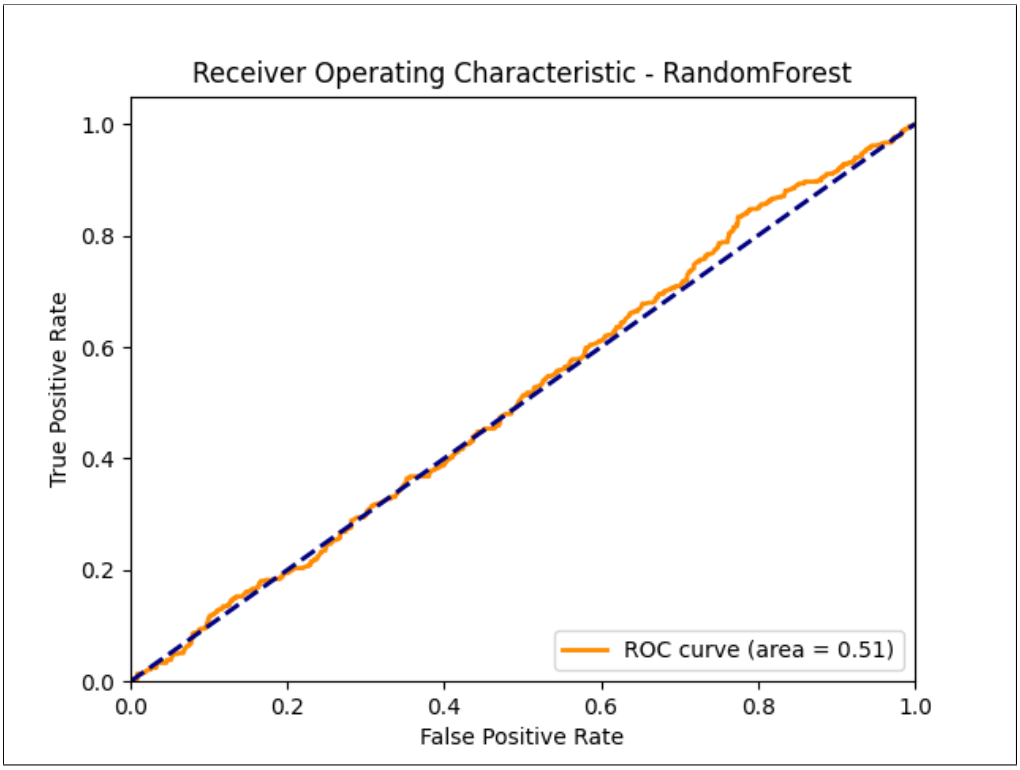

Figura 9: Curva ROC del modello RandomForest.

Il primo modello analizzato è stato il **Random Forest**, che ha prodotto una accuratezza media di $\bar{x} = 0.5241$ e una deviazione standard pari a $\sigma = 0.0002$. Come è possibile vedere dalla figura seguente, l'area sotto la curva ROC è di 0.5101. Un valore dell'area sotto la curva ROC così basso indica che il modello presenta quasi le stesse prestazioni di un modello casuale. Di conseguenza, questi valori sono decisamente poco ottimali, e suggeriscono che il modello non è particolarmente efficace per il nostro task di classificazione, con prestazioni che migliorano di poco rispetto ad un modello casuale.

Anche l'accuratezza è molto vicina al valore di casualità e il modesto valore dell'area sotto la curva indicano che il modello presenta difficoltà a distinguere tra le classi nel dataset. È quindi necessario esaminare ulteriori modelli per valutare se sono delle difficoltà dovute al modello impiegato oppure ai dati utilizzati.

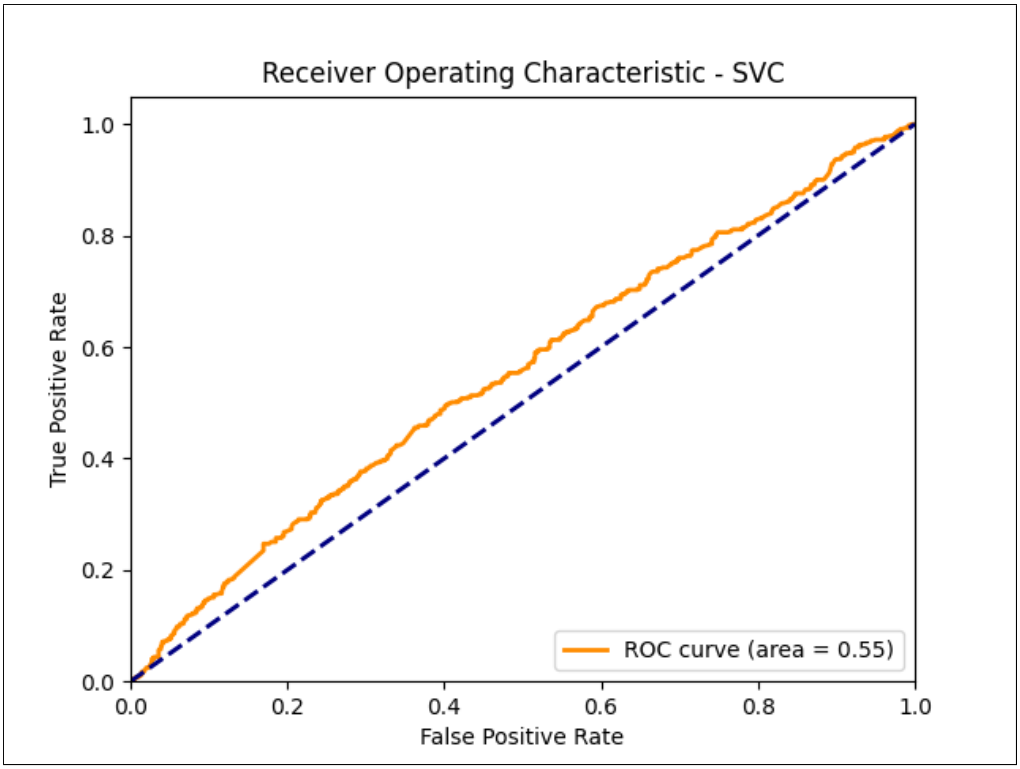

Figura 10: Curva ROC del modello SVC.

Il secondo modello considerato nel nostro studio è stato il **Support Vector Classification** (SVC). Questo modello ha prodotto un'accuratezza media di $\bar{x} = 0.5444$ e una deviazione standard di $\rho = 0.0041$. L'accuratezza media di 0.5444 indica che il modello, in media, ha classificato correttamente il 54.44% delle osservazioni. La deviazione standard relativamente bassa di 0.0041 suggerisce che c'è poca variabilità nei risultati del modello tra diverse suddivisioni del dataset, indicando una certa stabilità e coerenza nel suo comportamento.

In aggiunta all'accuratezza, è stata calcolata l'area sotto la curva ROC per il modello SVC, riportata nella figura precedente, ottenendo un valore di 0.5538. Un valore di AUC di 0.5538 è leggermente superiore a quello ottenuto dal modello Random Forest, suggerendo che SVC potrebbe avere una migliore capacità discriminativa rispetto a Random Forest. Questo valore di AUC, sebbene superiore, è comunque relativamente vicino a 0.5, il che indica che il modello SVC ha una performance solo moderatamente migliore rispetto a una classificazione casuale.

Il miglioramento rispetto a Random Forest, in termini di AUC, suggerisce che il Support Vector Classification potrebbe essere più adatto al nostro dataset specifico per compiti di classificazione. Tuttavia, va notato che l'accuratezza e l'AUC complessiva rimangono modeste, suggerendo che il modello potrebbe non essere particolarmente efficace nel distinguere tra le classi nel nostro caso specifico.

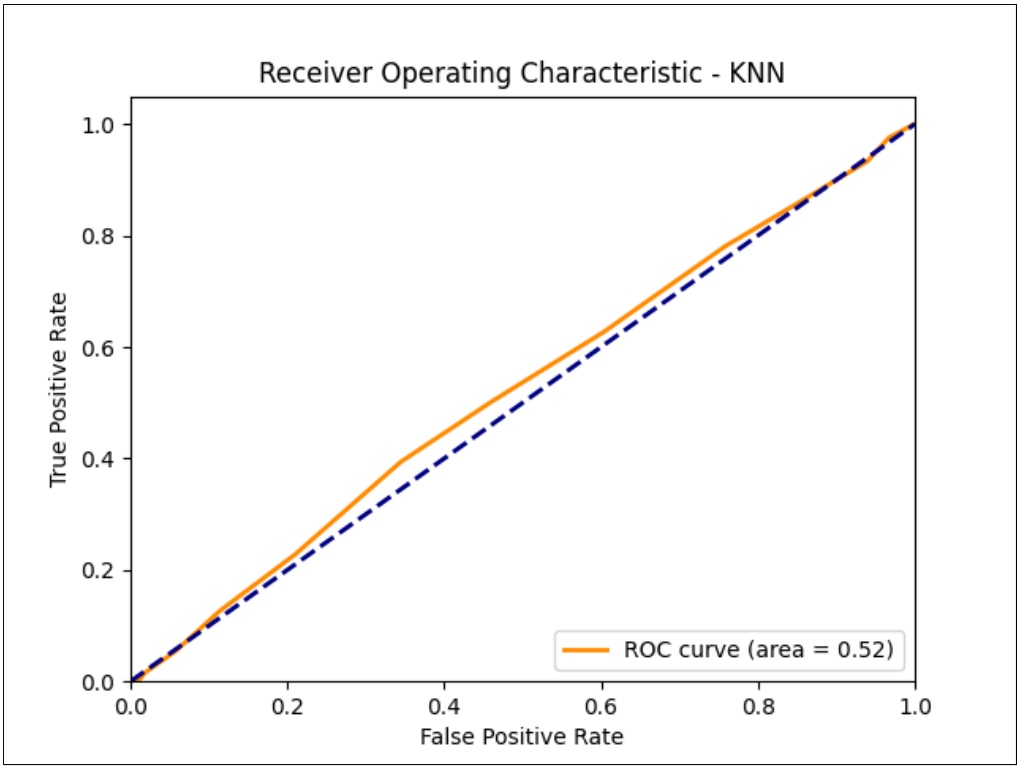

Figura 11: Curva ROC del modello KNN.

Il terzo modello esaminato è il **K-Nearest Neighbors** (KNN). Questo modello ha prodotto un'accuratezza media di $\bar{x} = 0.5337$ e una deviazione standard di $\rho = 0.0096$. L'accuratezza media di 0.5337 indica che, in media, il modello ha classificato correttamente il 53.37% delle osservazioni. La deviazione standard di 0.0096 suggerisce che i risultati del modello sono relativamente stabili attraverso diverse suddivisioni del dataset, ma anche che c'è una piccola variabilità nei risultati tra le diverse esecuzioni del modello.

Per quanto riguarda l'area sotto la curva ROC, in figura, è stato ottenuto un valore di 0.5211. Questo valore, che è molto vicino a 0.5, indica che la capacità del modello di distinguere tra le classi positive e negative è poco migliore rispetto a quella di una classificazione casuale.

I risultati indicano quindi che il KNN non mostra un miglioramento notevole rispetto agli altri modelli esaminati. Questo modello può offrire alcune variazioni minori nelle prestazioni, ma non rappresenta un avanzamento significativo rispetto alle soluzioni alternative.

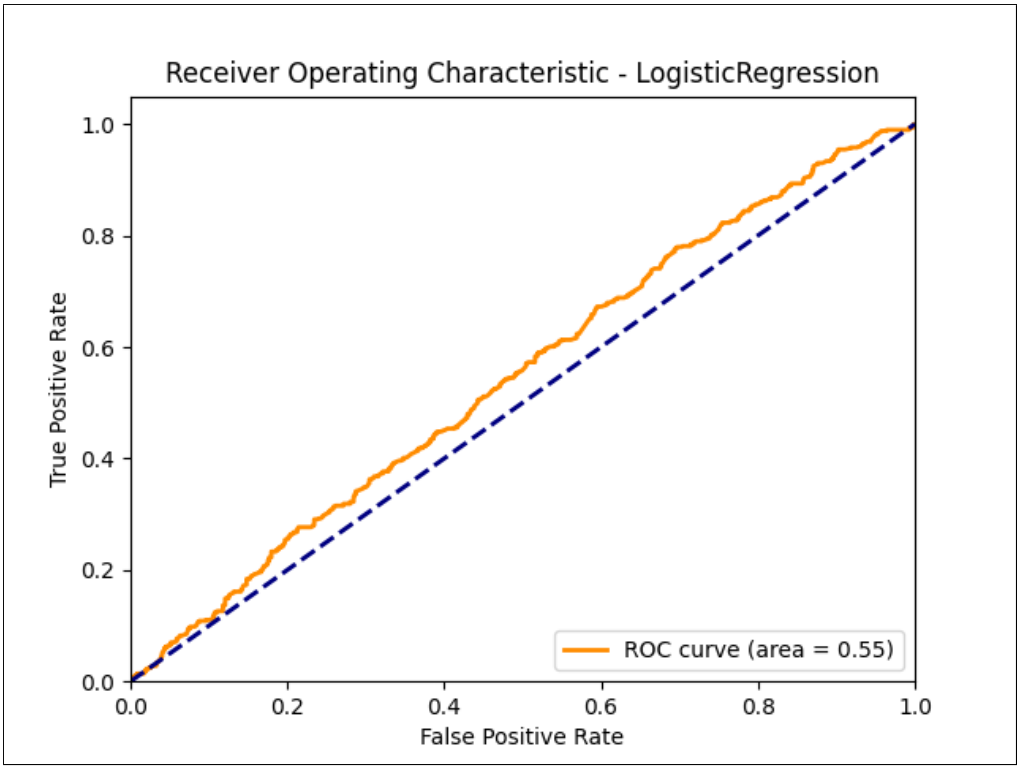

Figura 12: Curva ROC del modello LogisticRegression.

Infine, la **Regressione Logistica** presenta i valori più alti tra i modelli testati, con una accuratezza media di $\bar{x} = 0.5561$ e una deviazione standard di $\rho = 0.0000$. L'area sotto la curva ROC è di 0.5463. Questi risultati indicano che la Regressione Logistica ha il miglior equilibrio tra accuratezza e capacità di discriminazione tra le classi nel nostro dataset. Pertanto, il modello di Regressione Logistica sembra essere il più adatto per il nostro compito di classificazione.

Da notare come la deviazione standard ρ nulla sia in linea con la natura deterministica della Regressione Logistica. In un modello deterministico, la deviazione standard dei risultati dovrebbe essere zero, poiché le stesse condizioni di addestramento e test producono risultati identici.

Per concludere con certezza quale modello è il migliore a partire dai dati, però, occorre fare affidamento a metodi statistici. È fondamentale confrontare le loro prestazioni utilizzando metriche statistiche per determinare quale modello offre il miglior equilibrio tra accuratezza e capacità di discriminazione. In questo contesto, è stato utilizzato il test t di Student per confrontare le accuratezze medie dei modelli.

Il **test t di Student** è una procedura statistica utilizzata per confrontare le medie di due gruppi e determinare se le differenze osservate tra di esse sono statisticamente significative. È particolarmente utile quando il campione è di

dimensioni relativamente piccole e le varianze dei due gruppi sono simili. La formula del test t è la seguente:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{4}$$

dove:

- $\bar{x}_1$ e $\bar{x}_2$ sono le medie dei due gruppi;
- s_1^2 e s_2^2 sono le varianze dei due gruppi;
- n_1 e n_2 sono le dimensioni dei due gruppi.

Il valore ottenuto per t viene confrontato con una distribuzione t di Student con gradi di libertà calcolati in base alle dimensioni e varianze dei gruppi. Il p-value associato al test indica la probabilità di osservare una differenza almeno così grande tra le medie se in realtà non ci fosse alcuna differenza.

Per implementare il test di Student è stato sviluppato lo script "python/student.py" presente all'interno della cartella "python" che, a partire dai dati sulle accuratezze medie e sulle relative deviazioni standard calcolati dallo script "python/supervised.py" esportati in "data/model_accuracies.csv", calcola le statistiche t e i p-value per le coppie di modelli confrontati. Come valori per n_1 e n_1 verrà utilizzato il numero totale di valutazioni, calcolato come $n_{splits} \cdot n_{repeats} = 5 \cdot 100 = 500$ a causa della Repeated K-Fold, ed è quindi uguale per tutti i modelli. I risultati del test t di Student tra le accuratezze medie dei vari modelli sono i seguenti.

- **Confronto tra RandomForest e SVC:**
  - Statistica t: -3.5155
  - p-value: 0.0004
  - Risultato: Significativo. Le medie delle accuratezze tra RandomForest e SVC sono significativamente diverse. Questo suggerisce che l'accuratezza media di SVC è significativamente migliore rispetto a quella di RandomForest.
- **Confronto tra RandomForest e KNN:**
  - Statistica t: -0.7105
  - p-value: 0.4774
  - Risultato: Non significativo. Non c'è una differenza statisticamente significativa tra le medie delle accuratezze di RandomForest e KNN. Questo indica che le prestazioni di RandomForest e KNN sono simili.
- **Confronto tra RandomForest e LogisticRegression:**
  - Statistica t: -98.5491

  - p-value: 0.0000
  - Risultato: Significativo. Le medie delle accuratezze tra RandomForest e LogisticRegression sono significativamente diverse. LogisticRegression ha un'accuratezza media molto più alta rispetto a RandomForest.

- **Confronto tra SVC e KNN:**
  - Statistica t: 0.7283
  - p-value: 0.4664
  - Risultato: Non significativo. Non c'è una differenza statisticamente significativa tra le medie delle accuratezze di SVC e KNN. Le prestazioni dei due modelli sono comparabili.

- **Confronto tra SVC e LogisticRegression:**
  - Statistica t: -2.0182
  - p-value: 0.0436
  - Risultato: Significativo. Le medie delle accuratezze tra SVC e LogisticRegression sono significativamente diverse. LogisticRegression ha un'accuratezza media più alta rispetto a SVC.

- **Confronto tra KNN e LogisticRegression:**
  - Statistica t: -1.6531
  - p-value: 0.0983
  - Risultato: Non significativo. Non c'è una differenza statisticamente significativa tra le medie delle accuratezze di KNN e LogisticRegression. Le prestazioni di KNN e LogisticRegression sono simili.

Sulla base dei risultati del test t di Student, il modello di Regressione Logistica emerge come il miglior modello in termini di accuratezza media, mostrando una differenza significativa rispetto agli altri modelli testati. La Regressione Logistica presenta un'accuratezza media superiore rispetto a RandomForest, SVC e KNN, dimostrando così una migliore performance complessiva. Gli altri confronti rivelano differenze significative solo tra alcuni modelli, mentre altre coppie di modelli non mostrano differenze statisticamente rilevanti nelle loro accuratezze medie.

Questi risultati suggeriscono che, per il nostro compito di classificazione, la Regressione Logistica è la scelta preferibile rispetto agli altri modelli considerati. Nonostante ciò, i valori di accuratezza e di area sotto la curva rimangono bassi anche per la Regressione Logistica e non permettono di generare una predizione accurata.

Per confermare che il lavoro di feature selection sia stato soddisfacente, è stato sviluppato lo script "python/supervised_all.py" in cui sono state svolte

le medesime operazioni sui quattro modelli descritti in precedenza, senza però utilizzare il sottoinsieme di feature identificato da BorutaPy. Se l'algoritmo impiegava in media al più un'ora utilizzando solo il sottoinsieme di feature, ha impiegato quasi un giorno intero per computare utilizzando tutte le feature disponibili, con quelle categoriche sempre in codifica one-hot. In ogni caso, sono stati prodotti i risultati seguenti.

Modello	**Configurazione migliore**	**$\bar{x}$ e ρ della accuratezza**	**Area sotto la curva ROC**
RandomForest	"max_features": 'log2', "n_estimators": 100	$\bar{x} = 0.5203$, $\rho = 0.0003$	0.5196
SVC	"C": 0.1, "kernel": 'rbf'	$\bar{x} = 0.5446$, $\rho = 0.0038$	0.5455
KNN	"n_neighbors": 21	$\bar{x} = 0.5277$, $\rho = 0.0109$	0.5242
LogisticRegression	"C": 0.1	$\bar{x} = 0.5545$, $\rho = 0.0001$	0.5448

Tabella 7: Prestazioni dei modelli di apprendimento utilizzando tutte le feature disponibili all'interno del file "data/merged_data_reduced.csv".

È possibile adesso confrontare i risultati della Tabella 7, calcolati su tutte le feature disponibili, con quelli all'interno della Tabella 6, che usano le 13 feature identificate da BorutaPy al 90° percentile. Se analizziamo l'accuratezza media $\bar{x}$, possiamo notare come utilizzare tutte le feature fa decrementare leggermente i valori ottenuti. Per quanto riguarda la deviazione standard ρ, sono stati ottenuti anche in questo caso valori molto bassi, mentre i valori delle aree sotto la curva ROC fluttuano leggermente, ma non sono completamente confrontabili in quanto il ROC AUC non è il parametro di ottimizzazione utilizzato dall'algoritmo. Quindi, impiegare la feature selection è stato soddisfacente in quanto ha prodotto risultati leggermente migliori a fronte di un tempo di addestramento di gran lunga inferiore, rispetto all'utilizzo dei modelli con tutte le feature a disposizione.

Possiamo concludere che, sebbene sia stato individuato il modello migliore per il task di classificazione booleana predisposto, c'è una carenza nei dati che impedisce al modello di raggiungere prestazioni ottimali.

Apprendimento non supervisionato

Nel contesto del caso di studio, si è voluto anche verificare se l'apprendimento non supervisionato potesse fornire ulteriori spunti di analisi per l'identificazione di pattern latenti nel dataset degli incidenti. A tal fine, è stata condotta un'analisi delle componenti principali (PCA) seguita da un'analisi di clustering utilizzando l'algoritmo K-Means, all'interno dello script "python/unsupervised.py". L'obiettivo è stato, quindi, di determinare se ci sono tipologie significative di incidenti e, in tal caso, analizzarle.

Analisi delle Componenti Principali

L'**analisi PCA**, acronimo di Principal Component Analysis, è una tecnica statistica fondamentale per la riduzione della dimensionalità dei dati, mantenendo il più possibile l'informazione originale. In particolare, la PCA riduce la complessità del dataset proiettandolo su uno spazio a dimensione ridotta, pur mantenendo la maggior parte delle informazioni contenute nei dati originali. Le variabili originali del dataset vengono trasformate in un nuovo insieme di variabili, chiamate componenti principali, ortogonali tra loro e ordinate in base alla quantità di varianza che spiegano nei dati. La PCA cerca quindi di trovare una nuova serie di componenti principali che rappresentano le direzioni lungo cui i dati variano di più. La prima componente principale è quella lungo cui i dati mostrano la maggiore dispersione; la seconda componente principale è ortogonale alla prima e spiega la massima varianza residua, e così via.

Per il nostro scopo, abbiamo utilizzato la PCA per ridurre la dimensionalità del dataset, puntando a identificare un numero di componenti principali sufficienti a spiegare almeno l'80% della varianza totale. Questo valore di soglia è stato scelto per garantire che la maggior parte dell'informazione utile fosse preservata, permettendo al contempo una significativa semplificazione dei dati.

Il dataset è stato inizialmente standardizzato, in modo che ciascuna feature avesse media pari a zero e deviazione standard pari a uno, garantendo che il PCA non fosse influenzato da diverse scale di misura. Successivamente, il PCA è stato eseguito sul dataset standardizzato. Il risultato dell'analisi ha mostrato che le prime 39 componenti principali erano necessarie per spiegare cumulativamente circa l'81.06% della varianza totale, primo valore ottenuto oltre il valore soglia dell'80% prefissato, come riportato nel seguente output.

Cumulative Variance: [0.06666039, 0.10598143, 0.14381495, 0.17689035, 0.2058434, 0.23085014, 0.25584237, 0.27966974, 0.30257268, 0.32474606, 0.34632789, 0.36720047, 0.38753113, 0.4069233, 0.42590938, 0.44420682, 0.46220465, 0.48003499, 0.49732688, 0.51440213, 0.53118602, 0.54784526, 0.56415892, 0.58030936, 0.59628219, 0.61209932, 0.62780996, 0.64344973, 0.65905724, 0.67450836, 0.68992293, 0.70529786, 0.72060556, 0.73587774, 0.75111712, 0.76628911, 0.7812371, 0.79606325, 0.81064191]

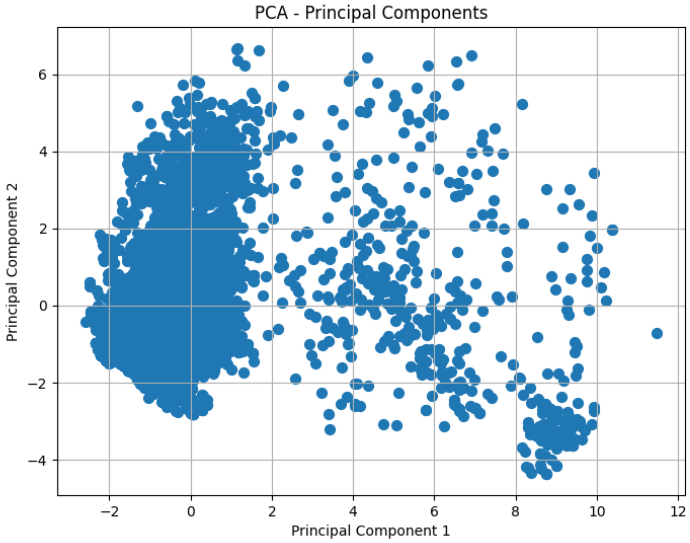

Figura 13: Distribuzione dei dati nelle prime due componenti principali (PCA). Ogni punto rappresenta un singolo campione trasformato nello spazio delle due componenti principali.

Il risultato ottenuto dall'analisi PCA, che ha evidenziato la necessità di 39 componenti principali per spiegare circa l'81.06% della varianza totale, suggerisce che i dati hanno una struttura dispersa. Questa complessità si riflette nel fatto che un numero elevato di componenti è necessario per catturare la maggior parte delle informazioni presenti nel dataset, mentre idealmente sarebbero necessarie solo alcune componenti principali per mantenere gran parte della varianza significativa nei dati. Nonostante ciò, si è deciso di proseguire con l'applicazione dell'algoritmo K-Means sui dati ridotti dimensionalmente per confermare ulteriormente l'ipotesi che non è possibile individuare cluster significativi.

Clustering con K-Means

L'**algoritmo K-Means** è stato utilizzato per cercare di individuare possibili gruppi distinti nel dataset, rappresentativi di differenti tipologie di incidenti stradali. Per l'analisi, sono stati ipotizzati inizialmente 3 cluster, con l'intento di esplorare la presenza di eventuali raggruppamenti nei dati.

Il risultato ha prodotto un Silhouette Score di 0.05892, un valore molto vicino allo zero, che indica una bassa coesione interna ai cluster e una scarsa separazione tra essi. Questo punteggio suggerisce che i cluster individuati non sono ben separati né coerenti, come evidenziato anche dalla Figura 14, dove i punti colorati in base al cluster di appartenenza mostrano una distribuzione piuttosto sovrapposta e indistinta.

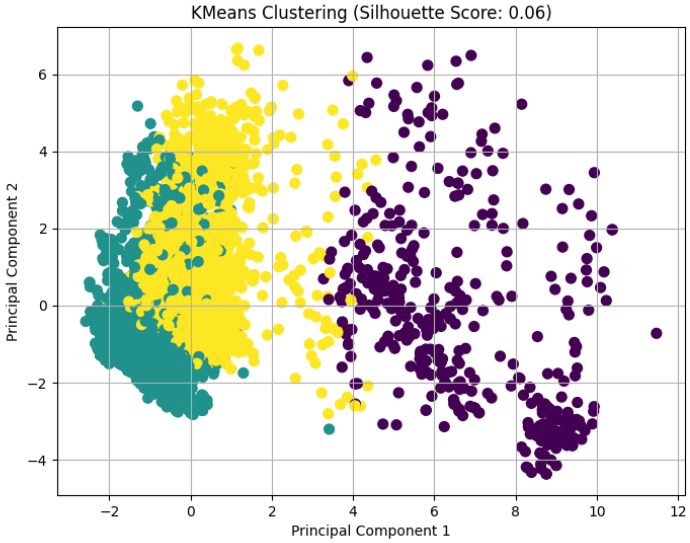

Figura 14: Risultati del clustering K-Means sui dati ridotti dimensionalmente tramite PCA. I punti sono colorati in base al cluster di appartenenza, e il punteggio Silhouette è indicato nel titolo.

Anche se l'analisi PCA abbia confermato la possibilità di ridurre, sebbene di poco, la dimensionalità del dataset mantenendo una buona parte dell'informazione originale, l'applicazione del clustering K-Means sui dati ridotti non ha portato a una chiara identificazione degli incidenti in gruppi distinti e ben delimitati.

Questo risultato conferma che, almeno con le feature e la metodologia impiegata, non è stato possibile identificare pattern latenti significativi nel dataset degli incidenti. Pertanto, si può concludere che, in questo contesto e con i dati a disposizione, l'apprendimento non supervisionato non ha fornito ulteriori informazioni utili.

Conclusioni finali

I risultati ottenuti dal presente caso di studio sono sicuramente uno spunto di riflessione per eventuali progetti futuri. La metodologia applicata ha permesso di correlare dati provenienti da diverse fonti per creare una base di conoscenza espansa e coerente. Nonostante ciò, le basse prestazioni ottenute dai modelli di apprendimento supervisionato e non supervisionato hanno suggerito una carenza nei dati disponibili pubblicamente, che non permettono di svolgere predizioni affidabili o di identificare correttamente pattern negli incidenti.

Stabilire se in un incidente sono presenti feriti, avendo a disposizione solamente un ridotto insieme di variabili di input, è un compito molto complesso. Sono presenti sicuramente un numero di elevatissimo di altri fattori, oltre a quelli disponibili nei dataset presi in considerazione, che influenzano l'esito di un incidente, riguardo la presenza o meno di feriti. Tra questi, giusto per citarne alcuni, potrebbero esserci i tipi di veicoli coinvolti, lo stato di alterazione dei conducenti, la velocità effettiva del mezzo o l'utilizzo di cinture di sicurezza.

Alcuni di questi dati esistono e sono stati raccolti, ma non erano disponibili come dati aperti sul portale Opendata del Comune di Bari, e quindi non potevano essere oggetto di valutazione. Probabilmente, con un numero maggiore di feature si potrebbero migliorare le prestazioni dei modelli. Migliorando la qualità dei dati a disposizione si potrebbe creare un modello che, con ogni probabilità, stima la gravità di un incidente e, di conseguenza, potrebbe potenzialmente anche salvare vite umane.

Tra gli sviluppi futuri, si potrebbe integrare il seguente progetto nelle applicazioni di navigazione per quanto riguarda la gestione dei percorsi alternativi in base alla gravità di un incidente. Infatti, allo stato attuale delle cose, diverse app di navigazione presentano la funzionalità di segnalazione degli incidenti da parte degli utenti. Si potrebbe raffinare il presente modello in modo tale da poter raccogliere tutte le feature necessarie a partire dalle coordinate dell'incidente segnalato e il relativo timestamp. In questo modo, si stima la gravità di un incidente segnalato in base alla presenza stimata di feriti, con l'obiettivo di riprogrammare il percorso in tempi più rapidi, rispetto a quanto avviene attualmente in base ai dati stimati di traffico, se l'incidente è stimato come grave.

Le conseguenze maggiori potrebbero essere una migliore ottimizzazione dei percorsi da parte delle applicazioni di navigazione e una riduzione del traffico generato dagli incidenti stradali, permettendo ai soccorsi di agire in modo più efficiente.