UNIVERSITÀ DI PISA

Scuola di Ingegneria

Dipartimento di Ingegneria dell'Informazione

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

RICONOSCIMENTO DI FLUSSI DI MOBILITÀ URBANA CON MODELLI GRADIENT BOOSTING

Candidato

Pietro Tempesti

Relatore

Prof. Mario G.C.A. Cimino
Prof. Gigliola Vaglini
Prof. Antonio L. Alfeo

Anno Accademico 2020/2021

Indice

Abstract

La presente tesi si propone di riconoscere le dinamiche di mobilità urbana di alcune specifiche zone della città di Istanbul (Turchia) tramite dati provenienti da celle telefoniche. Infatti, attraverso il numero di chiamate effettuate in una determinata fascia oraria, rilevando il flusso di mobilità, è possibile capire a quale delle suddette zone di studio ci stiamo riferendo. Associare i flussi ai vari case study permette di stabilire i comportamenti delle persone, per esempio, all'interno di una zona commerciale o di uffici, ed operare scelte di gestione urbana più consapevoli. Le analisi dei flussi di mobilità sono state fatte confrontando i risultati ottenuti utilizzando diverse tecniche di *machine learning*.

Nello studio presentato si sono implementati due classificatori di tipo *Gradient Boosting*, ovvero il modello *Gradient Boosting* e il modello *XGBoost*, effettuando il *tuning* dei parametri e degli iperparametri dei modelli. Sono stati scelti questi modelli poiché dovrebbero garantire una buona accuratezza anche in problemi aventi classi molto simili tra loro, come il caso analizzato.

Infine, si sono confrontati i risultati empirici ottenuti con altri modelli di tipo ensemble ma basati su tecniche diverse, e si può concludere che, per quanto le tecniche utilizzate garantiscano un'accuratezza sufficiente e in generale superiore ai risultati di approcci generali come quello utilizzato, non riescano a raggiungere performance paragonabili a quelle di modelli più specifici alla risoluzione del problema in questione.

Introduzione

L'imponente crescita dei dispositivi digitali permette di utilizzare un'enorme mole di dati generati dagli utenti per identificare e studiare gli spostamenti delle persone all'interno di una città, al fine di migliorare la mobilità urbana. Alcuni esempi di dati utilizzabili, nel rispetto della privacy degli utenti, sono i GPS di veicoli e smartphone, l'utilizzo di carte di credito, le chiamate telefoniche e i post sui social media.

La comprensione dei comportamenti della mobilità urbana nei contesti cittadini permette di migliorare la qualità della vita delle persone, ad esempio implementando una rete intelligente di trasporti pubblici per diminuire il traffico ed il rischio di incidenti, e di migliorare la gestione delle risorse pubbliche, rendendo più efficiente la manutenzione della città diminuendo al contempo gli sprechi di risorse.

Lo studio presentato si propone di individuare un modello accurato per ricondurre diverse tipologie di flussi di mobilità urbana a determinate zone. Sono stati utilizzati dati provenienti dalle celle telefoniche di Istambul, nello specifico da cinque zone: HISTORIC – TOURISM, LEVENT, MASLAK – INDUSTRIES, TAKSAM – SHOPPING, UNIVERSITY.

Supponendo che i flussi di mobilità siano fortemente correlati all'affluenza delle persone, dato un dataset grezzo, sarà possibile allenare un modello per individuare l'hotspot corrispondente ad un dato flusso di mobilità. Una volta ottenuti i dati dal modello, tramite l'analisi di questi si possono svolgere diverse azioni mirate all'ottimizzazione della mobilità urbana, come regolare i trasporti pubblici in un certo hotspot a seconda dell'affluenza in un dato momento della giornata al fine di offrire un servizio di alta qualità utilizzando il minor numero di risorse e riducendo al minimo l'impatto ambientale, oppure si possono effettuare interventi come un potenziamento dell'illuminazione negli hotspot con maggior affluenza serale, mirati ad una maggior qualità della vita dei cittadini.

Questo studio è dunque principalmente indirizzato alle autorità cittadine, che grazie ad un'analisi delle dinamiche di mobilità all'interno delle città possono pianificare interventi e prendere provvedimenti con una maggiore efficacia. Lo studio può essere funzionale anche ad aziende ed imprese, pubbliche o private, per esempio nella scelta della zona migliore per l'apertura o il trasferimento di una filiale.

Machine Learning

Per compiere lo studio illustrato nella tesi sono stati utilizzati degli algoritmi di machine learning, una branca dell'intelligenza artificiale che impiega algoritmi in grado di risolvere i problemi apprendendo le informazioni dai dati: un programma ha appreso dai dati se alla fine dello svolgimento del compito da svolgere le sue prestazioni sono aumentate. L'obiettivo è quindi che una macchina sia in grado di portare a termine nuovi compiti basandosi sull'esperienza fatta su dei dati di apprendimento precedentemente preparati, andando a creare un modello probabilistico generale che sia in grado di produrre delle previsioni sufficientemente accurate quando vengono sottoposti nuovi casi. La definizione più famosa del Machine Learning è stata fornita da Tom M. Mitchell: *Si dice che un programma apprende dall'esperienza E con riferimento a alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E* [1].

Gli algoritmi di machine learning possono essere divisi in tre macrocategorie: gli algoritmi supervisionati (*Supervised learning*), gli algoritmi rinforzati (*Reinforcement learning*) e gli algoritmi non supervisionati (*Unsupervised learning*). I modelli illustrati in questo testo fanno uso di algoritmi supervisionati: questi algoritmi imparano ad associare un certo output all'input fornito dopo aver passato loro degli esempi di associazione input-output. Questi algoritmi verranno utilizzati per risolvere problemi di classificazione.

Classificazione

Nei problemi di classificazione si richiede all'algoritmo di indicare a quale categoria (output) appartengano i dati passati in input, in base alle caratteristiche intrinseche (parametri) di ogni istanza da classificare; il

modello andrà a produrre ed utilizzare una funzione $f: \mathbb{R}^n \rightarrow \{1 \dots k\}$, dove {1 … k} è l'insieme delle possibili classi, mentre n è il numero delle caratteristiche dei parametri in input. Si distinguono problemi di classificazione binaria, in cui le possibili categorie alla quale può appartenere un input sono due (ad esempio distinguere se una e-mail è spam o meno in base al contenuto del messaggio stesso è un problema di classificazione binaria) e problemi di classificazione a classe multipla, nei quali l'insieme delle classi avrà cardinalità maggiore di 2. Poiché nel caso esaminato ci sono cinque diversi hotspot da distinguere, il problema da risolvere sarà un problema di classificazione a classe multipla.

Algoritmi Gradient Boosting

Gli algoritmi Gradient Boosting sono algoritmi che si basano sul modello ensemble, ovvero sulla combinazione di singoli modelli semplici (i *weak learner*) che insieme creano un modello più completo e performante (lo *strong learner*).

Questa combinazione, nel caso esaminato, viene effettuata mediante una tecnica di *boosting*: viene creato un modello iniziale, e successivamente si crea in sequenza un secondo modello che si concentra sul sottoinsieme di istanze nel quale il primo modello ha avuto le prestazioni inferiori. La combinazione di questi due modelli dovrebbe portare risultati migliori rispetto ai due modelli precedenti. Ogni iterazione del processo di boosting sarà quindi indirizzata a correggere le debolezze principali della combinazione dei modelli precedenti

L'algoritmo Gradient Boosting, per correggere le carenze dei modelli combinati, cerca di prevedere i possibili errori del modello. Questo errore è quantificato dai residui, ovvero la differenza tra i valori osservati dal modello ed i valori previsti. Questa famiglia di algoritmi, ad ogni iterazione del boosting, punta a stimare i residui, al fine di minimizzarli (e di conseguenza minimizzare l'errore).

Tra questi algoritmi negli ultimi anni si è distinto XGBoost (eXtreme Gradient Boosting), che offre un livello di accuratezza migliore rispetto al metodo Gradient Boosting tradizionale e un'ampia gamma di applicazioni e piattaforme compatibili, al punto da essere uno degli algoritmi più usati nelle competizioni di machine learning.

Related Works

Lo studio effettuato sfrutta una serie di approcci basati su ensemble: vengono creati diversi modelli di base (i weak learner precedentemente citati) ed ognuno svolge la propria elaborazione, dopodiché viene effettuando la combinazione (pesata o non pesata) delle predizioni di ogni weak learner. Diversi studi riguardanti aspetti di mobilità hanno mostrato come l'utilizzo dei modelli ensemble porti un vantaggio considerevole rispetto gli approcci classici.

In generale, l'utilizzo di queste tecniche consente di avere risultati più stabili, visto che l'errore prodotto sarà inferiore a quello del singolo modello, e questi errori vengono spesso corretti dalle altre predizioni causando un drastico aumento della precisione dell'intero modello[2]; questi metodi garantiscono inoltre una buona risposta a problemi di overfitting, ovvero l'adattamento eccessivo ai casi specifici del training set che però non hanno riscontro nel resto dei casi[3].

Uno dei principali aspetti di cui tenere conto in un problema di classificazione della mobilità urbana è quello della potenziale similitudine tra alcuni casi di studio: per esempio è possibile che gli orari di maggior affollamento in una zona industriale siano simili a quelli di una zona con uffici amministrativi o ad una zona universitaria.

Una situazione come quella descritta sopra porta a dover cercare un approccio che permetta di distinguere le zone simili con buona precisione, ma spesso per effettuare un'operazione del genere è necessario utilizzare degli approcci specifici per il caso di studio esaminato; per quanto questi metodi permettano di avere delle ottime prestazioni nella risoluzione del problema di classificazione anche con classi simili, c'è bisogno di un nuovo studio e di un nuovo modello ogni volta che si presenta un nuovo caso di studio.

Nel 2020 Silva[1] ha presentato un approccio basato sullo stacking, ovvero un sistema nel quale il problema viene prima diviso in sotto-problemi di classificazione binaria, per poi usare un *meta-learner* per combinare i risultati dei sotto-problemi e risolvere il problema originale. Lo studio da lei prodotto ha dimostrato come l'affidabilità di due diversi approcci *stacking*, ovvero *One vs. One* e *One vs. Rest*, sia assolutamente soddisfacente (oltre l'80% per One vs. Rest, oltre il 90% per One vs. One), ma al contempo richiede una suddivisione in sotto-problemi specifica per il problema che si va ad analizzare ed inoltre richiede tempi di training del modello importanti nel caso di dataset molto grandi (nell'ordine di GB), di conseguenza rendendo il modello difficilmente generalizzabile e riutilizzabile in altri contesti.

L'utilizzo di approcci meno specifici però deve garantire buone prestazioni anche a fronte di casi di studio molto simili tra loro, e chiaramente usare un sistema più classico e generale per risolvere il problema non avrà lo stesso grado di precisione rispetto ad un approccio mirato: per quantificare la differenza di accuratezza tra l'approccio *stacking* ed un approccio più tradizionale è stato preso in esame lo studio presentato da Morucci nel 2020, nel quale è stato usato un modello basato sull'algoritmo Random Forest per la classificazione degli hotspot di Istambul. Il modello si basa su un approccio *bagging*, in cui tutti i classificatori generati effettuano una predizione di pari importanza e l'output finale corrisponderà alla classe maggiormente predetta in un modo simile ad un sistema di votazione, ha ottenuto risultati appena sufficienti, nell'ordine del 60%, e oltretutto senza miglioramenti significativi dopo un potenziamento del modello (svolto aumentando il numero di alberi di decisione utilizzati)[4].

APPROCCIO PRESENTATO	ACCURATEZZA MEDIA	VARIANZA
ONE VS. ONE – STACKING	0.935	0.0032
ONE VS. REST – STACKING	0.8025	0.0053
RANDOM FORREST	0.62 ± 0.01	

Tabella 1: prestazioni degli approcci discussi nei related works

In questo studio verranno mostrati due approcci, entrambi basati sul Gradient Boosting, che per sua natura dovrebbe essere in grado di minimizzare gli errori e di conseguenza aumentare l'accuratezza, senza però richiedere un doppio livello di apprendimento e le alte risorse richieste dagli approcci stacking: raggiungere un buon livello di accuratezza renderebbe un modello Gradient Boosting preferibile ai modelli presentati specialmente per imprese/aziende che si trovano a dover effettuare studi sulla mobilità urbana in ambienti molto eterogenei tra loro.

Design ed implementazione

Nella prima metà di questa sezione verranno mostrati gli approcci presentati al termine del capitolo precedente. I risultati ottenuti utilizzando i modelli presentati saranno disponibili nella sezione *Risultati sperimentali*.

In tutto lo studio si farà riferimento ai dati relativi alla mobilità di Istambul spiegati nel capitolo *Case study*.

Design

Approccio basato sul modello Gradient Boosting proposto dalla libreria Scikit-learn

In questo paragrafo verrà spiegato il procedimento svolto per la creazione di un modello basato sul Gradient Boosting in grado di garantire le massime prestazioni sul caso di studio in analisi. A seguire lo pseudocodice dell'algoritmo:

input:

- Training set $\{(x_i, y_i)\}_{i=1}^{N}$
- Una funzione di perdita differenziabile $L\big(y, F(x)\big)$
- Un numero di iterazioni M

Output:

- Funzione di stima $F_M(x)$

Svolgimento:

1. Inizializzo il modello coi valori di partenza $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma)$
2. *for* M = 1 *to* M:
    a. calcolo gli pseudo-residui: $r_{im} = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]$ per i = 1 … n
    b. Creo un weak learner (es. un albero) $h_m(x)$ con gli pseudo-residui e lo alleno col training set
    c. Calcolo il moltiplicatore $\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
    d. aggiusto i pesi $F_m(x) = F_{m-1}(x) + \gamma h_m(x)$

Per migliorare le prestazioni del modello è stato effettuato il tuning dei parametri e degli iperparametri del modello: partendo dai parametri di default, sono stati effettuati test andando a modificare alcuni valori relativi alle caratteristiche dell'algoritmo: viene effettuata una ricerca a griglia utilizzando diverse configurazioni dei parametri e degli iperparametri, andando a cercare quale di queste configurazioni garantisca le maggiori prestazioni.

I parametri sui quali si è intervenuti sono di tre categorie:

1. **Parametri specifici degli alberi:** riguardano le caratteristiche del singolo albero
2. **Parametri di boosting**: riguardano le operazioni di boosting
3. **Parametri generali**

I parametri interessati dal tuning appartengono alle prime due categorie, e verranno presentati nel dettaglio.

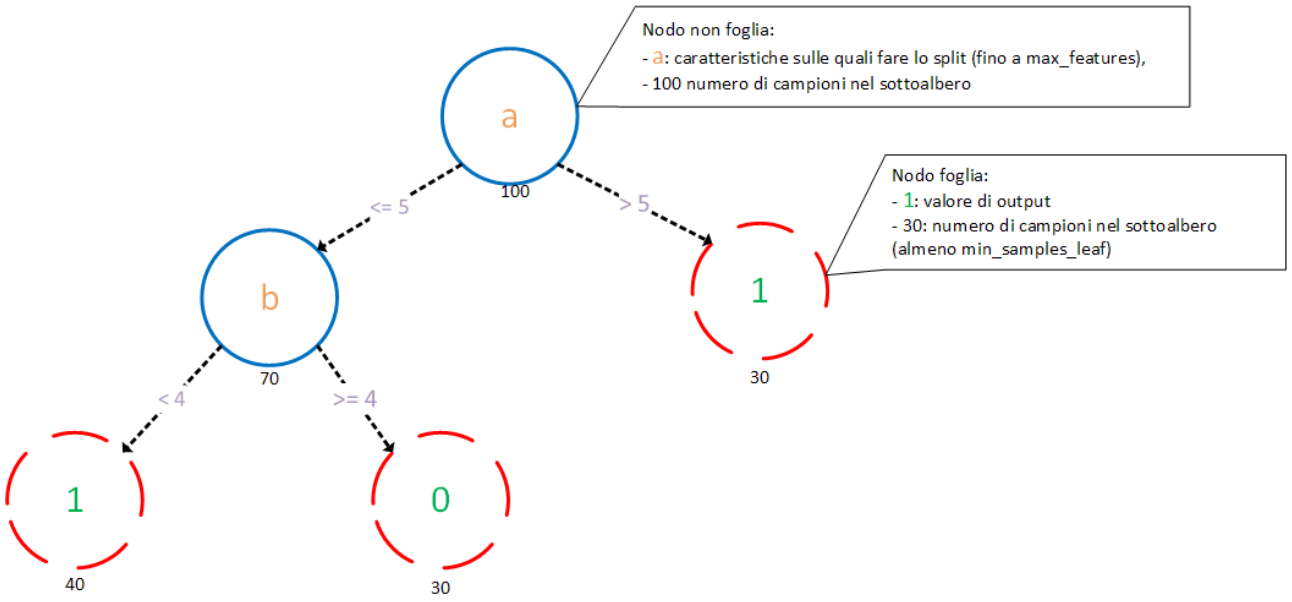


Figura 1: rappresentazione schematica di un albero del Gradient Boosting

Parametri specifici degli alberi:

- **Max_depth**: indica il massimo numero di livelli in ogni albero.
- **Min_samples_leaf:** indica il numero minimo di campioni che si devono trovare in un nodo foglia: un qualsiasi nodo verrà considerato solo se sono presenti almeno min_samples_leaf campioni in entrambi i rami destro e sinistro.
- **Max_features**: massimo numero di caratteristiche dell'input considerate per la valutazione del campione ad ogni nodo.

Parametri di boosting:

- **Learning_rate**: indica il peso di ogni albero nel risultato finale; in genere si scelgono valori abbastanza bassi per evitare problemi di *overfitting*.
- **N_estimators**: numero di alberi che devono essere creati, quindi numero di iterazioni del boosting.

PARAMETRO	VALORE DI DEFAULT	RANGE DI TUNING
MAX_DEPTH	3	2 – 64
MIN_SAMPLES_LEAF	1	3 – 81
MAX_FEATURES	24 *[numero delle caratteristiche]*	1 – 7
LEARNING_RATE	0.1	0.01 – 0.75
N_ESTIMATORS	100	50 - 200

Tabella 2: parametri modificati nel tuning del modello Gradient Boosting

Poiché il modello utilizzato si basa sul *supervised learning*, il dataset viene diviso in due sezioni, una per il training (corrispondente al 75% del dataset) ed una per il testing (il restante 25%).

Approccio basato sul modello XGBoost

Anche in questo caso verrà illustrato il procedimento effettuato per la creazione di un modello mirato a massimizzare l'accuratezza dei risultati del problema di classificazione, in questo caso andando ad utilizzare l'algoritmo XGBoost.

XGBoost segue più o meno lo stesso algoritmo visto per il Gradient Boosting, ma con alcune differenze:

- Utilizza lo sviluppo di Taylor (fino al secondo ordine) per calcolare il valore della funzione di perdita per il singolo weak learner;
- Non va a costruire fin da subito l'intero albero $h_m(x)$ per poi esplorarlo, ma lo costruisce dinamicamente con tecnica *greedy*

Inoltre, XGBoost utilizza una serie di funzionalità che garantiscono una maggiore velocità di esecuzione ed una maggiore stabilità.

Per effettuare il tuning di XGBoost è stato però utilizzato un procedimento diverso: infatti si è fatto uso di un tool chiamato *Optuna*: con questa libreria si può creare uno *study*, ovvero una serie di iterazioni (*trials*) nelle quali si effettua un test utilizzando sempre combinazioni di parametri diversi, in range specifici per ogni parametro (specificati mediante degli appositi metodi forniti dalla libreria); i parametri non vengono scelti casualmente ad ogni trial, ma si utilizza un algoritmo di tipo *tree-structured parzen estimator* al fine di massimizzare ad ogni trial il punteggio ottenuto andando a considerare i singoli parametri uno per volta. Alla fine dello study è possibile andare ad analizzare i migliori risultati ottenuti anche in forma grafica. I risultati e i grafici del tuning verranno mostrati nella sezione *Risultati sperimentali*.

Le categorie dei parametri in XGBoost sono diverse rispetto a Gradient Boosting, e sono:

1. **Parametri generali**: relativi agli aspetti generali dell'algoritmo
2. **Parametri di boosting**: relativi al processo di boosting ad ogni step (sono compresi anche i parametri degli alberi)
3. **Parametri di apprendimento**: relativi alle funzioni utilizzate per l'apprendimento e la valutazione degli input

Si sono modificati principalmente i parametri di boosting e il parametro n_estimators, che svolge la stessa funzione del parametro omonimo nel Gradient Boosting; di seguito la lista dei parametri di tuning oggetto di tuning:

- **Eta**: equivalente del learning_rate di Gradient Boosting, indica il peso del singolo albero sull'output
- **Min_child_weight**: indica il peso minimo delle osservazioni fatte sui nodi figli
- **Max_depth**: equivalente dell'omonimo di Gradient Boosting: indica il numero massimo di livelli in ogni albero
- **Gamma**: indica la minima riduzione della funzione di perdita necessaria ad effettuare uno split
- **Colsample_bytree**: indica la frazione di caratteristiche dell'input da considerare ad ogni split
- **reg_alpha** e **reg_lambda**: indicano il peso delle regolazioni L1 ed L2.

PARAMETRO	VALORE DI DEFAULT	RANGE DI TUNING
N_ESTIMATORS	100	20 – 1000
ETA	0.3	0.005 - 0.5
MIN_CHILD_WEIGHT	1	0 – 10
MAX_DEPTH	6	2 - 50
GAMMA	0	0 - 5
COLSAMPLE_BYTREE	1	0.1 – 1
REG_ALPHA	0	0 - 5
REG_LAMBDA	1	0 - 5

Implementazione

Nella seconda parte della sezione verranno mostrati i diagrammi UML relativi al caso d'uso del software utilizzato nel case study della smart city e successivamente i diagrammi delle classi del software implementato.

Casi d'uso

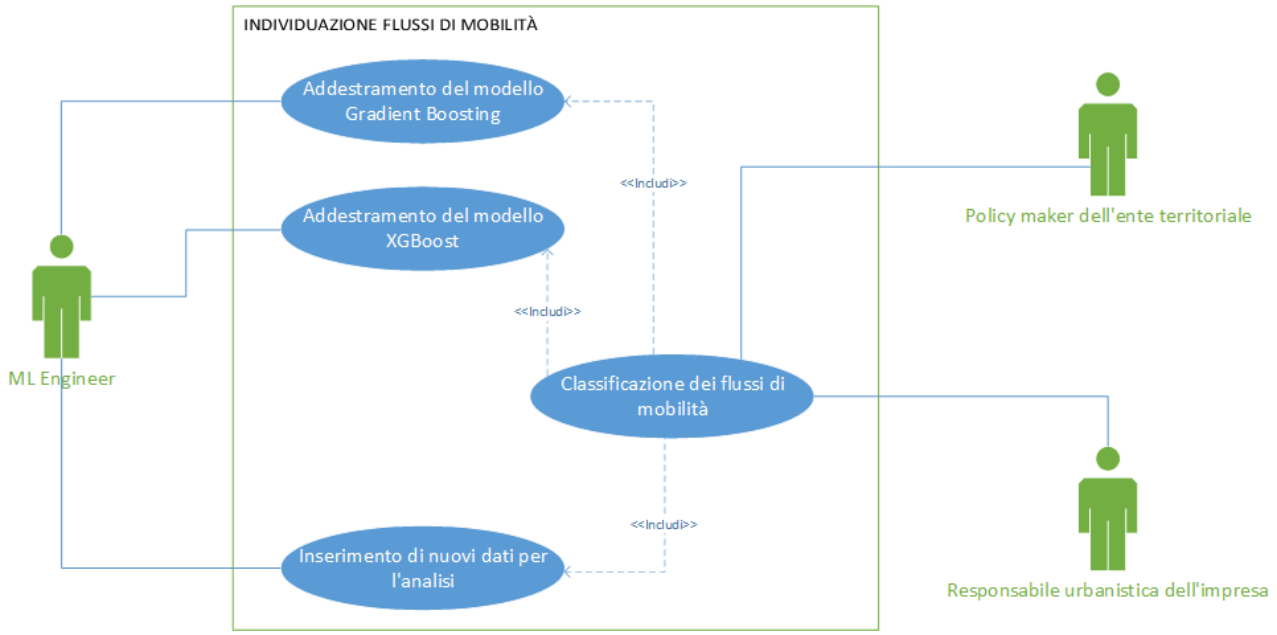


Figura 2: Diagramma del caso d'uso

Nel diagramma mostrato in figura 2 possiamo vedere quali sono le interazioni dei clienti con il sistema. Da una parte è presente il Machine Learning Engineer, responsabile della creazione dei modelli; dall'altra sono indicati i due tipi di fruitori principali dello studio analizzato, ovvero il policy maker di un ente territoriale e il responsabile delle scelte di ambito urbanistico di un'impresa.

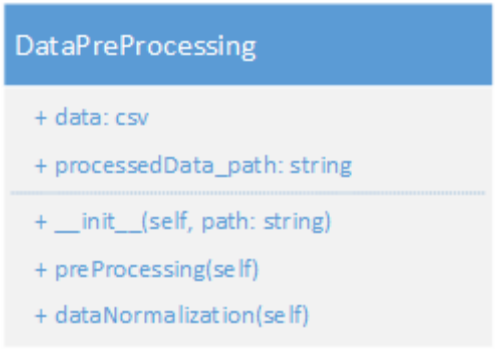
Per effettuare la classificazione dei flussi di mobilità, in primo luogo bisogna acquisire i dati ed effettuare una pre-processazione di questi; successivamente il ML engineer si occuperà della generazione dei modelli di machine learning basati su gradient Boosting/XGBoost.

I clienti potranno soltanto utilizzare i modelli precedentemente ultimati, e nel momento in cui si vogliono utilizzare dei nuovi dati questi andranno prima pre-processati.

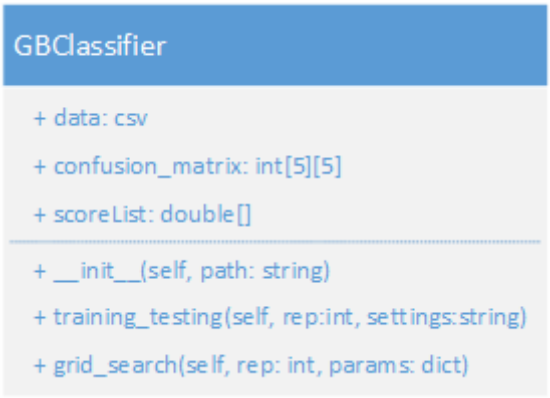
Questo tipo di software può essere usato in vari problemi attinenti alla mobilità urbana: il policy maker di una città può infatti usare questo prodotto per una gestione intelligente dei trasporti pubblici, o ancora il responsabile urbanistico di un'impresa può regolare gli orari di apertura al pubblico, o spostare una filiale a seconda dell'affluenza nelle varie zone della città. Infatti, supponendo la presenza di una correlazione tra i flussi di mobilità e la concentrazione di persone in una determinata area, è possibile usare un modello di riconoscimento dei flussi di mobilità urbana per ottenere dei dati che possano rendere più efficaci le scelte effettuate dai responsabili sopra indicati.

Diagrammi delle classi

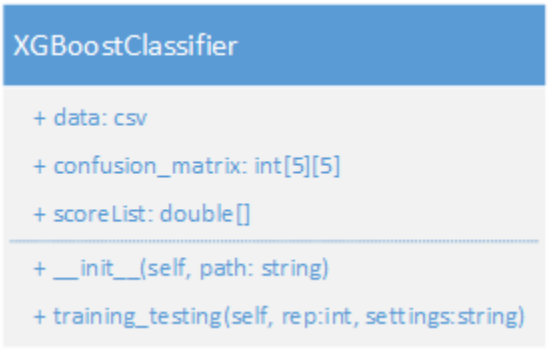
Verranno ora presentati i diagrammi delle classi `DataPreProcessing`, `GBClassifier` ed `XGBoostClassifier`.



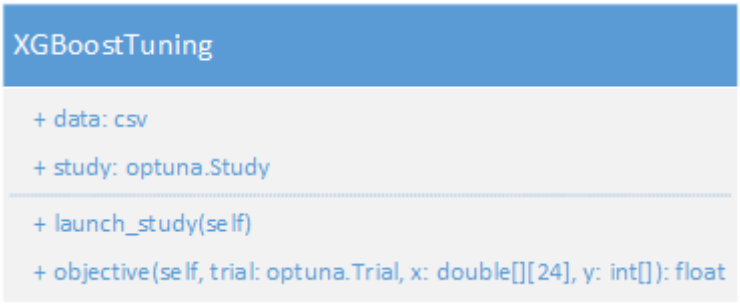
Classe che implementa la pre-processazione dei dati. Le operazioni svolte verranno discusse nel dettaglio nella sezione *Case Study*, alla voce *PreProcessing.*



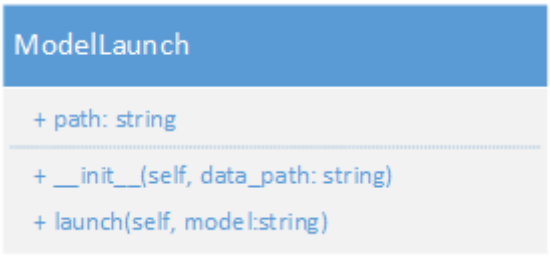
Classe che implementa il modello Gradient Boosting: il metodo `trainig_testing` viene usato per addestrare la rete ed effettuare il test delle prestazioni, il metodo `grid_search` invece serve per svolgere il tuning dei parametri.



Classe che implementa il modello XGBoost: il metodo `trainig_testing` viene usato per addestrare la rete ed effettuare il test delle prestazioni.



Classe che implementa il tuning dei parametri XGBoost mediante il tool Optuna: il metodo `objective` indica le operazioni da svolgere nel singolo *trial*, mentre `launch_study` lancia le operazioni di tuning.



Classe che implementa le operazioni da svolgere per avviare i modelli già ottimizzati. Il costruttore esegue la pre-processazione dei dati caricati, e mediante il metodo launch il chiamante può avviare la valutazione di uno dei due modelli passando il nome dell'algoritmo desiderato come parametro. Questa classe è stata inserita al fine di rendere il modello utilizzabile anche ad un utente non esperto, che vuole vedere i migliori risultati possibili per i modelli, previa ottimizzazione da parte del ML Engineer.

Nella sezione *Appendice* saranno mostrati esempi di codice Python nei quali sono state utilizzate le classi sopracitate.

Case Study

I dati su cui si basa questo studio provengono dalle celle telefoniche della città di Istanbul nei primi tre mesi del 2018 [gennaio, febbraio, marzo].

Le zone della città considerate sono: HISTORIC – TOURISM, LEVENT, MASLAK – INDUSTRIES, TAKSIM – SHOPPING, UNIVERSITY.

- HISTORIC – TOURISM: zona storica di Istanbul che ricopre l'area della Basilica di Santa Sofia e le zone limitrofe fino al Bosforo. Essendo una zona storica, è ricca di turismo e ha un'alta concentrazione di affluenza.
- LEVENT: La zona industriale di Istanbul, molto affollata negli orari lavorativi ma con poche abitazioni intorno.
- MASLAK – INDUSTRIES: uno dei principali quartieri commerciali di Istanbul, dove ci sono prevalentemente uffici. Zona affollata durante la settimana e tranquilla nel fine settimana.
- TAKSIM – SHOPPING: rappresenta la zona dello shopping. La piazza Taskim è il centro principale dello shopping; la zona è calma durante la settimana, ma molto vivace nel weekend.
- UNIVERSITY: zona universitaria; questa rispecchia la vita studentesca.

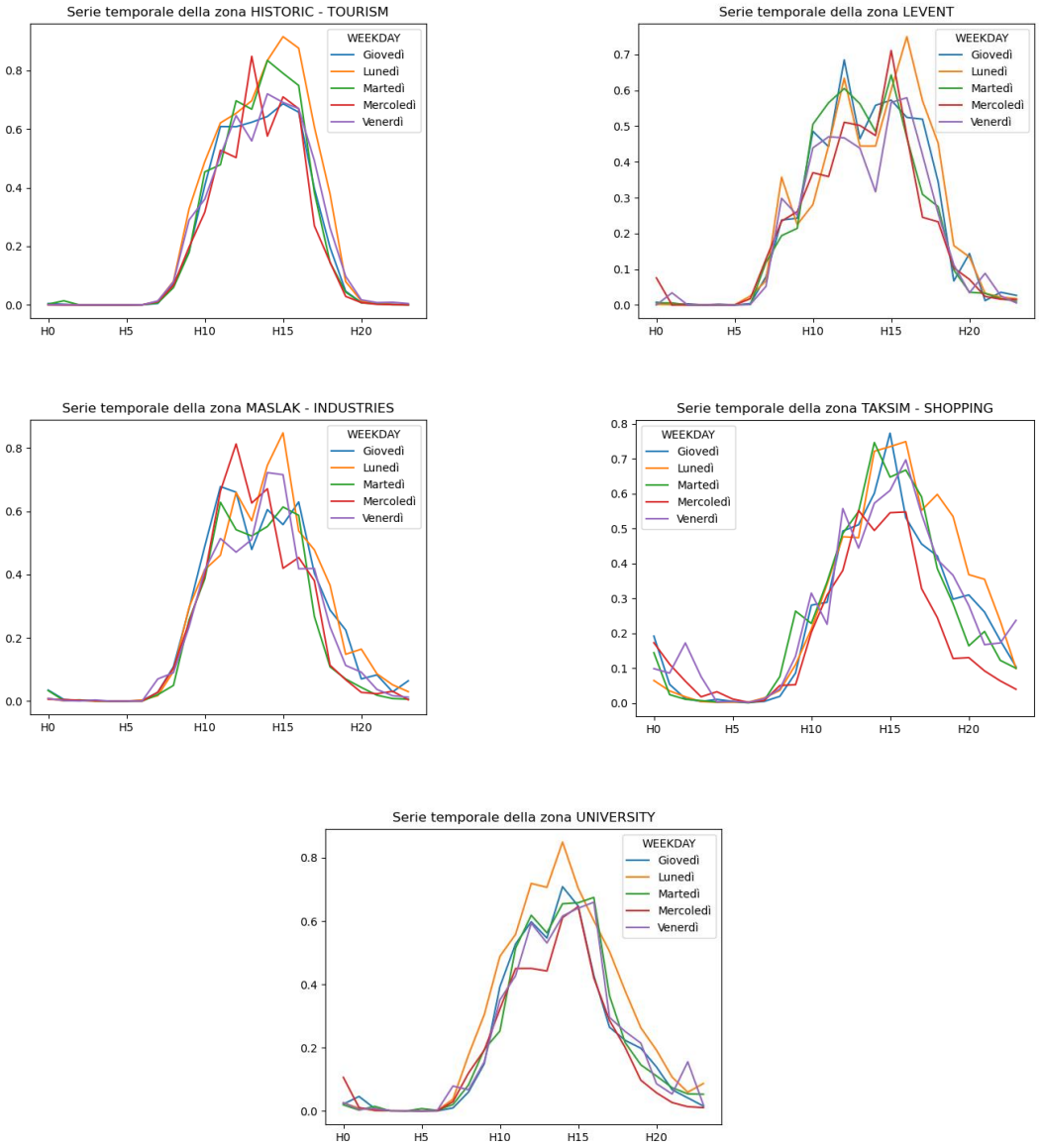


Figura 3: Serie temporali delle cinque zone in esame

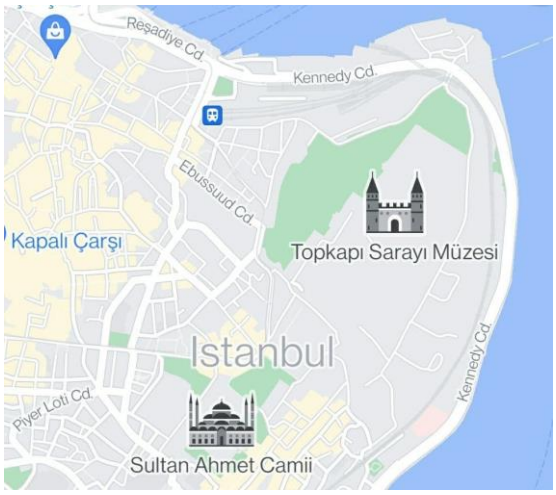


Figura 4: zona Historic

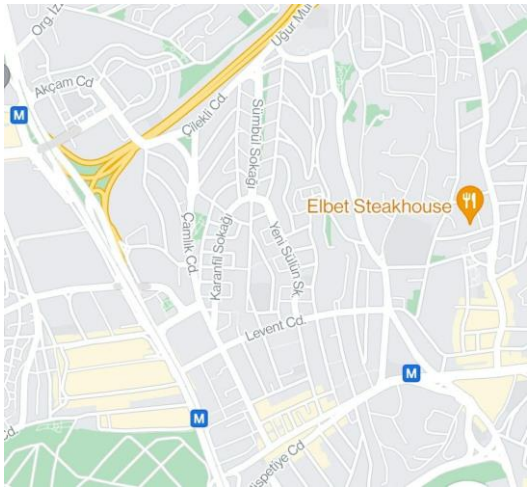


Figura 5: zona Levent

Figura 6: zona Maslak

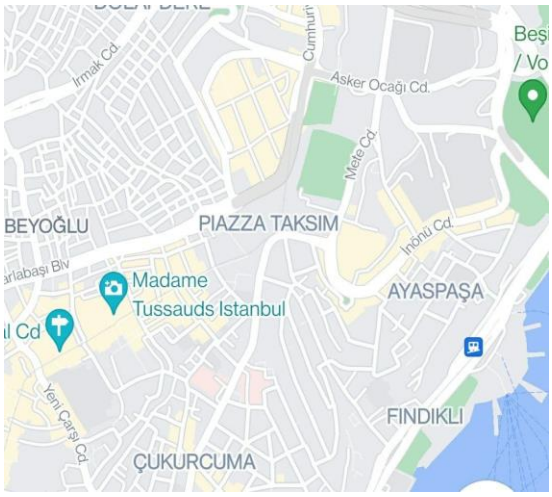


Figura 7: zona Taksim

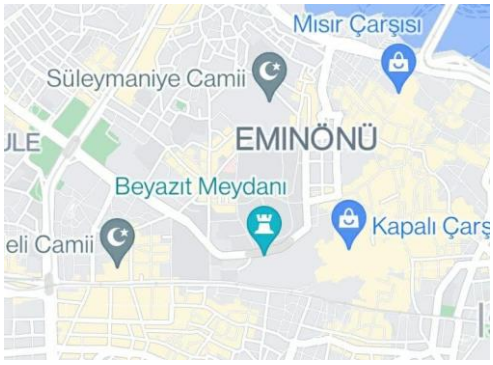


Figura 8: zona University

PreProcessing

Questa sezione si propone di mostrare la fase di pre-processing dello studio nella quale viene eseguita una manipolazione del data set per produrre informazioni significative.

Il dataset inziale, grezzo, si presenta nel formato:

CASESTUDY	MONTH	DAY	WEEKEND[1]	H0	…	H23
UNIVERSITY	1	1	0	123		4353
HISTORIC - TOURISM	3	6	0	542		53
MASLAK - INDUSTRIES	1	5	1	234		563
TAKSIM - SHOPPING	2	12	1	32		3243
LEVENT	2	24	0	56		32
…	…	…	…	…	…	…

Tabella 3: dataset non processato

Il dataset su cui si basa lo studio si riferisce ai dati presi nei primi tre mesi del 2018: gennaio, febbraio e marzo.

Lo scopo di questa fase è quello di eliminare le righe e colonne che sono irrilevanti per lo studio e compiere sia una trasformazione di informazione da testuale a numerica sia una normalizzazione dei dati.

Le fasi di pre-processing eseguite sul dataset sono le seguenti:

1. Eliminazione delle colonne: MONTH e DAY. Le colonne MONTH e DAY sono state eliminate in quanto irrilevanti per la classificazione dei dati.
2. Eliminazione delle righe riferite ai finesettimana ed eliminazione della colonna WEEKEND. Dal momento in cui gli algoritmi di machine learning richiedono una grossa quantità di dati per essere addestrati e il dataset originale mostra i dati dei primi 3 mesi del 2018, il numero di quelli riguardanti il fine settimana risulta minoritario e insufficiente per l'adeguato addestramento di un algoritmo ML. Pertanto, le relative righe sono state eliminate.
3. Eliminazione di righe nulle: il dataset presenta varie righe nulle per ogni hotspot e questo potrebbe andare a influire negativamente sull'addestramento del modello, pertanto devono essere eliminate.
4. Normalizzazione dei dati: i dati delle colonne [H0, … , H23] sono stati normalizzati per avere un' omogeneità di intervallo, andando così a considerare un intervallo limitato da (0,1) e non tra (0, ∞). La formula con cui opera lo scaler è $x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$

Alla fine della fase di pre-processing, il dataset di lavoro si presenta nella seguente forma:

CASESTUDY	H0	…	H23
UNIVERSITY	123		4353
HISTORIC - TOURISM	542		53
MASLAK - INDUSTRIES	234		563
TAKSIM - SHOPPING	32		3243
LEVENT	56		32
…	…	…	…

Tabella 4: dataser pre-processato

[1] Weekend vale 1 se il giorno è nel finesettimana, 0 altrimenti

Risultati sperimentali

In questa sezione sono mostrati i risultati degli approcci presentati in Design ed implementazione. I risultati sono stati calcolati aggregando i risultati di 500 iterazioni del training e del testing della rete.

Per confrontare gli approcci si useranno due metriche: l'accuratezza e la matrice di confusione.

- **Accuratezza:** in un problema di classificazione indica il rapporto tra gli elementi correttamente classificati e l'insieme di tutti gli elementi. Maggiore è l'accuratezza, maggiori saranno le istanze correttamente classificate. Un'accuratezza significativa è attestata da valori superiori al 60%.
- **Matrice di confusione:** matrice utilizzata per descrivere le prestazioni del modello di classificazione per ciascuna classe. Ogni colonna della matrice rappresenta i valori predetti dal modello, mentre ogni riga rappresenta i valori reali. L'elemento sulla riga i e colonna j indica il numero di istanze di classe i che il modello ha predetto essere di classe j, quindi la diagonale i=j rappresenta le istanze correttamente predette dal modello. Grazie alla matrice di confusione si possono evidenziare le classi che il modello confonde con più facilità.

Modello basato su Gradient Boosting

Questo caso di studio era rivolto a mostrare i risultati dell'algoritmo Gradient Boosting prima nella sua versione di default, e poi dopo un opportuno tuning degli hyperparametri.

Si partirà dalle operazioni di tuning: di seguito sono riportate le tabelle che hanno portato alla scelta degli iperparametri (in grassetto le combinazioni selezionate).

N_ESTIMATORS	LEARNING_RATE	SCORE MEDIO	DEVIAZIONE STD	SCORE MASSIMO	SCORE MINIMO
50	**0.01**	0.581647	0.058592	0.74193548	0.4193548
50	**0.1**	0.655795	0.054244	0.80645161	0.5
50	**0.25**	0.6604176	0.0559	0.83870	0.4516129
50	**0.5**	0.667755	0.05673	0.8387	0.5
50	**0.75**	0.65948	0.0573	0.87097	0.46774
100	**0.01**	0.619594	0.05998	0.79032	0.41935
100	**0.1**	0.662	0.05707	0.8387	0.43548
100	**0.25**	0.664846	0.05425	0.82258	0.46774
100	**0.5**	0.66617	0.0522	0.8387	0.5
100	**0.75**	0.6657	0.0537	0.8387	0.48387
150	**0.01**	0.63776	0.0587	0.82258	0.4516
150	**0.1**	0.660547	0.0536	0.8548	0.51613
150	**0.25**	0.6599	0.0553	0.80645	0.5
150	**0.5**	0.658575	0.0571	0.80645	0.5
150	**0.75**	0.66682	0.05895	0.80645	0.5
200	**0.01**	0.643	0.0551	0.80645	0.4516
200	**0.1**	0.66	0.05816	0.822581	0.4516
200	**0.25**	0.662	0.05592	0.8387	0.5
200	***0.5***	**0.6667**	**0.0594**	**0.82258**	**0.5**
200	**0.75**	0.6635	0.05767	0.80465	0.43548

Tabella 5: grid search dei parametri n_estimators e learning_rate

MAX_DEPTH	MIN_SAMPLES_LEAF	SCORE MEDIO	DEVIAZIONE STD	SCORE MAX	SCORE MIN
2	***3***	**0.6699**	**0.05682**	**0.8225**	**0.4516**
2	**9**	0.6557	0.05596	0.80645	0.5
2	**27**	0.6377	0.05688	0.80645	0.4677
2	**81**	0.5527	0.05836	0.7258	0.3387
8	**3**	0.648	0.059	0.80645	0.4677
8	**9**	0.6527	0.05817	0.8548	0.4677
8	**27**	0.64235	0.05318	0.7903	0.4516
8	**81**	0.557	0.0561	0.7419	0.4032
16	**3**	0.6461	0.0545	0.8226	0.41935
16	**9**	0.6416	0.05485	0.7903	0.4032
16	**27**	0.645	0.0567	0.8226	0.4677
16	**81**	0.5555	0.059	0.7258	0.4032
64	**3**	0.6482	0.0543	0.80645	0.4839
64	**9**	0.6399	0.0569	0.7903	0.4839
64	**27**	0.6421	0.05625	0.80645	0.4516
64	**81**	0.56045	0.05995	0.7258	0.3871

Tabella 6: grid search dei parametri max_depth e min_samples_leaf

Per il parametro *max_features* è stata prima effettuata una ricerca dei valori più efficaci con *GridSearchCV,* i cui risultati sono riportati qui sotto. In seguito si sono effettuati i test con i parametri più frequentemente indicati da *GridSearchCV*.

MAX_FEATURES	N° DI VOLTE IN CUI È STATO SELEZIONATO
1	**183**
2	**147**
3	74
4	40
5	26
6	12
7	18

Tabella 7: parametri ideali secondo GridSearchCV

MAX_FEATURES	SCORE MEDIO	DEVIAZIONE STD	SCORE MASSIMO	SCORE MINIMO
1	**0.6982**	**0.05206**	**0.871**	**0.5645**
2	0.6972	0.0534	0.8226	0.5484
SQRT	0.6873	0.05357	0.8226	0.5484

Tabella 8: confronto tra le prestazioni tra i parametri ideali consigliati da GridSearch e il parametro SQRT, radice quadrata delle features

Alla fine delle operazioni di tuning, si possono dunque confrontare i parametri di default con quelli ottimizzati e i risultati ottenuti nelle due configurazioni.

PARAMETRO	VALORE DI DEFAULT	VALORI OTTIMIZZATI
MAX_DEPTH	3	2
MIN_SAMPLES_LEAF	1	3
MAX_FEATURES	24 *[numero delle caratteristiche]*	1
LEARNING_RATE	0.1	0.5
N_ESTIMATORS	100	200

Tabella 9: confronto tra i parametri di default e i parametri ottimizzati

APPROCCIO	SCORE MEDIO	VARIANZA
DEFAULT	0.6574	0.00313
TUNED	0.69265	0.00264

Tabella 10: risultati ottenuti con gradient boosting

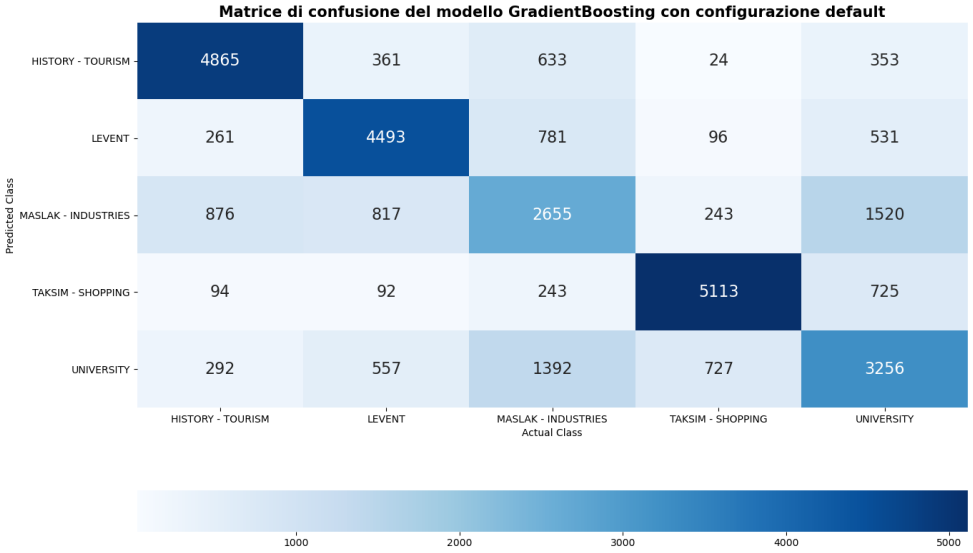


Figura 9: matrice di confusione ottenuta dal modello gradient boosting coi parametri di default

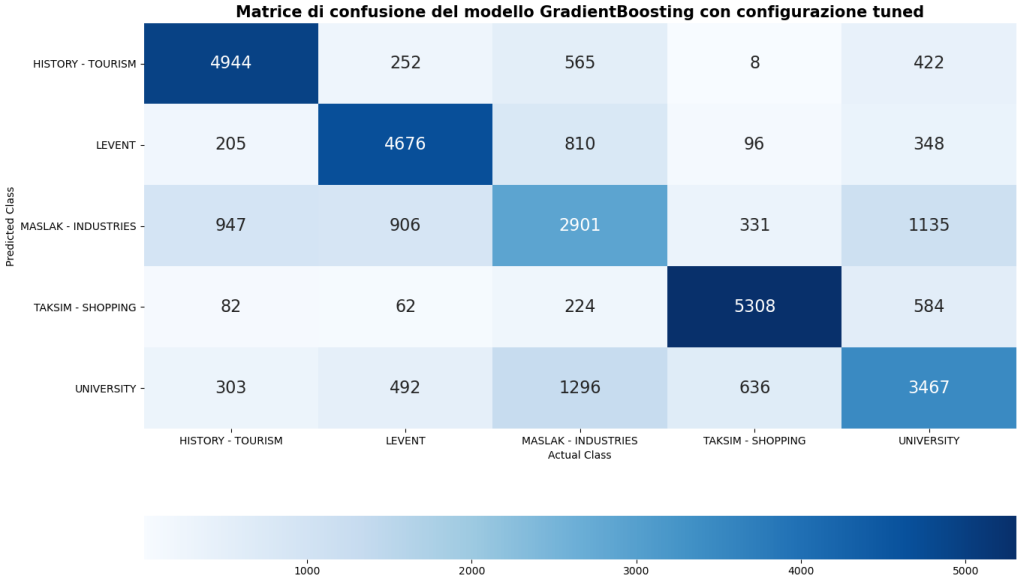


Figura 10: matrice di confusione ottenuta dal modello gradient boosting coi parametri ottimizzati

Come possiamo vedere dalla matrice di confusione, in entrambi i casi il modello commette molti errori nel riconoscere correttamente le classi Maslak ed University.

Modello basato su XGBoost

Questo caso di studio era rivolto a mostrare i risultati dell'algoritmo XGBoost prima nella sua versione di default, e poi dopo un opportuno tuning degli hyperparametri. Come specificato in precedenza il tuning dei parametri XGBoost è stato effettuato mediante il tool OpTuna.

Per effettuare il tuning sono state svolte 3 esecuzioni dello study; alla fine di ogni iterazione, si sono modificati gli intervalli dei parametri seguendo le indicazioni dei grafici riportati in seguito. Questi grafici, ovvero gli slice plot forniti da Optuna, permettono di vedere quante volte è stato ottenuto un certo punteggio per ogni iperparametro oggetto del tuning: di conseguenza si andranno a cercare per ogni iperparametro le zone più scure in cui ha ottenuto il punteggio più alto, e al contempo si cercheranno di evitare zone scure presenti a punteggi più bassi, poiché indicano valori del parametro poco efficaci in molte combinazioni.

Primo study:

PARAMETRI	MINIMO	MASSIMO
N_ESTIMATORS	20	1000
MAX_DEPTH	2	25
REG_ALPHA	0	5
REG_LAMBDA	0	5
MIN_CHILD_WEIGHT	0	5
GAMMA	0	5
ETA	0.005	0.5
COLSAMPLE_BYTREE	0.1	1

Tabella 11: range dei parametri nel primo study

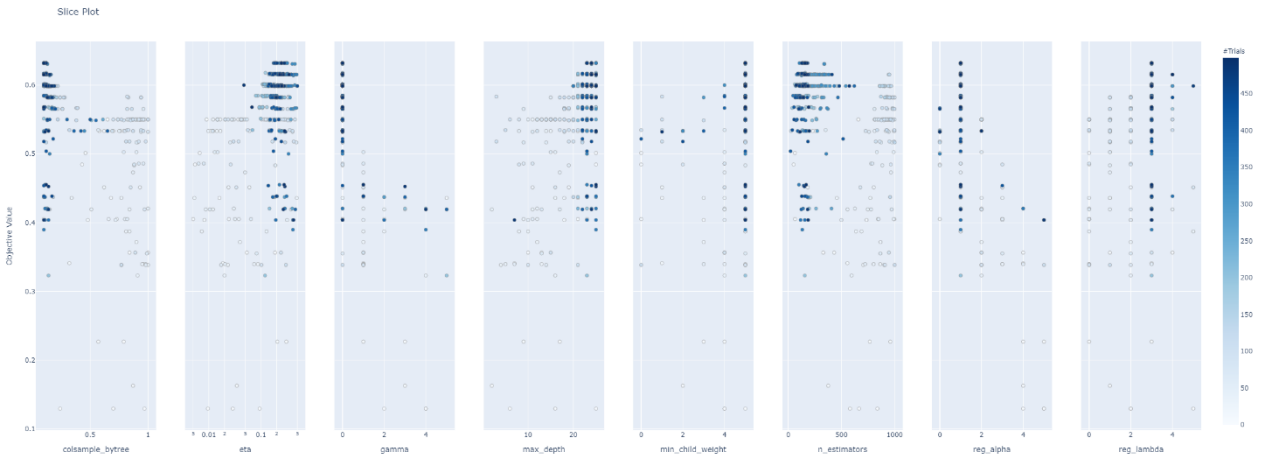


Figura 11: grafico plot_slice del primo study

Dopo il primo study si può già raffinare il tuning: viene eseguito un incremento degli iperparametri max_depth e min_child_weight, e si procede a restringere i campi o stabilire dei valori definitivi per gli altri parametri.

Secondo study:

PARAMETRI	MINIMO	MASSIMO
N_ESTIMATORS	80	200
MAX_DEPTH	23	50
REG_ALPHA	1	
REG_LAMBDA	2	4
MIN_CHILD_WEIGHT	4	10

GAMMA	0	
ETA	0.15	0.5
COLSAMPLE_BYTREE	0.1	0.3

Tabella 12: parametri del secondo study

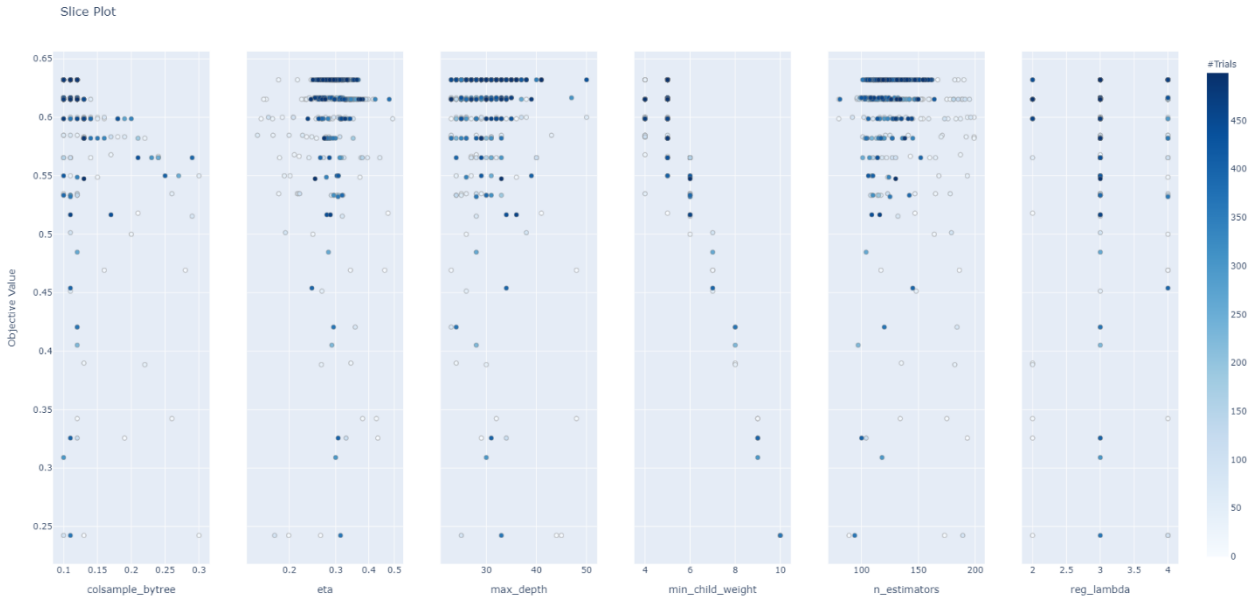


Figura 12: grafico slice plot del secondo study

Alla fine di questa seconda iterazione si possono affinare ulteriormente i parametri, e fissarne alcuni in maniera definitiva.

Terzo study:

PARAMETRI	MINIMO	MASSIMO
N_ESTIMATORS	100	150
MAX_DEPTH	25	35
REG_ALPHA	1	
REG_LAMBDA	3	
MIN_CHILD_WEIGHT	5	
GAMMA	0	
ETA	0.25	0.35
COLSAMPLE_BYTREE	0.11	

Tabella 13: range dei parametri utilizzati nel terzo study

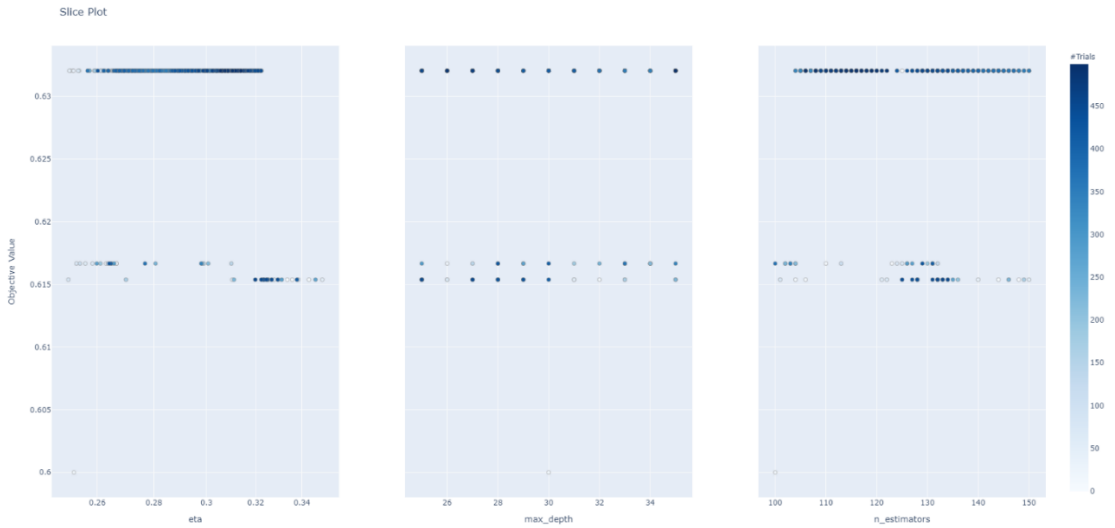


Figura 13: grafico slice plot dei risultati del terzo study

Grazie a questo grafico si può vedere come in realtà gli intervalli di valori scelti per i parametri consentano di scegliere quasi qualsiasi valore, di conseguenza per affinare il tuning sono stati scelti i valori medi e le zone dove ci sono state meno cadute dello score.

Si può passare adesso al confronto tra i parametri di default ed i parametri ottimizzati, e successivamente verranno mostrati i risultati ottenuti nelle due configurazioni.

PARAMETRO	VALORE DI DEFAULT	VALORE OTTIMIZZATO
N_ESTIMATORS	100	115
ETA	0.3	0.31
MIN_CHILD_WEIGHT	1	5
MAX_DEPTH	6	28
GAMMA	0	0
COLSAMPLE_BYTREE	1	0.11
REG_ALPHA	0	1
REG_LAMBDA	1	3

Tabella 14: confronto tra i parametri di default e i parametri ottimizzati

APPROCCIO	SCORE MEDIO	VARIANZA
DEFAULT	0.6442	0.00301
TUNED	0.6845	0.0026

Tabella 15: prestazioni del modello XGBoost

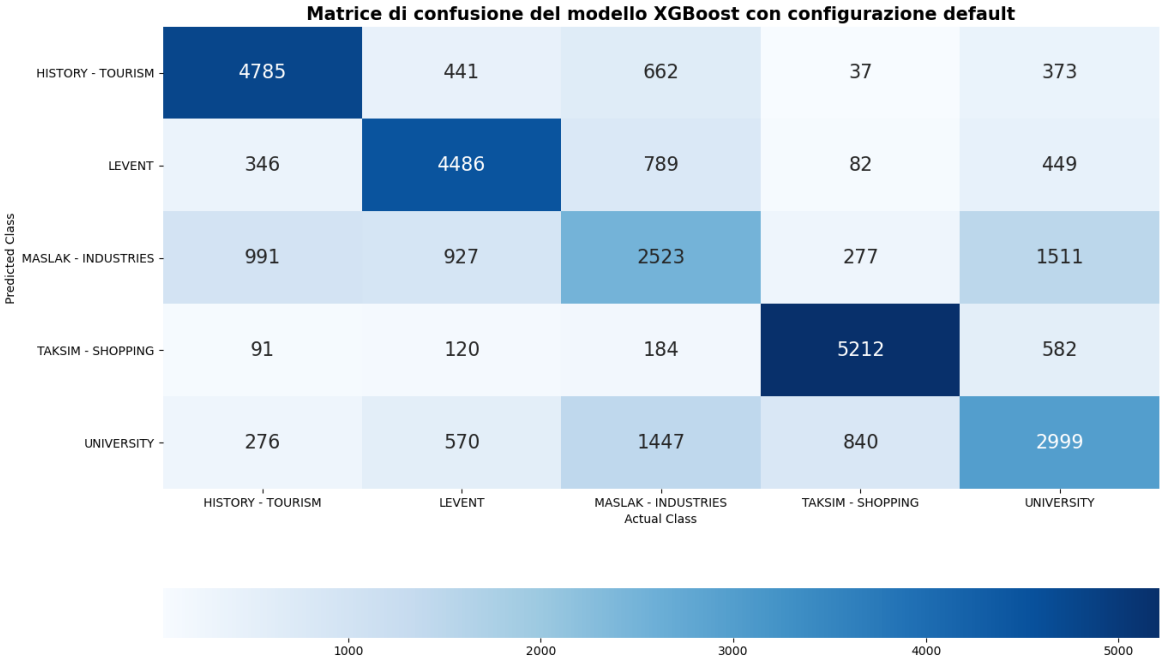


Figura 14: matrice di confusione del modello XGBoost con i parametri di default

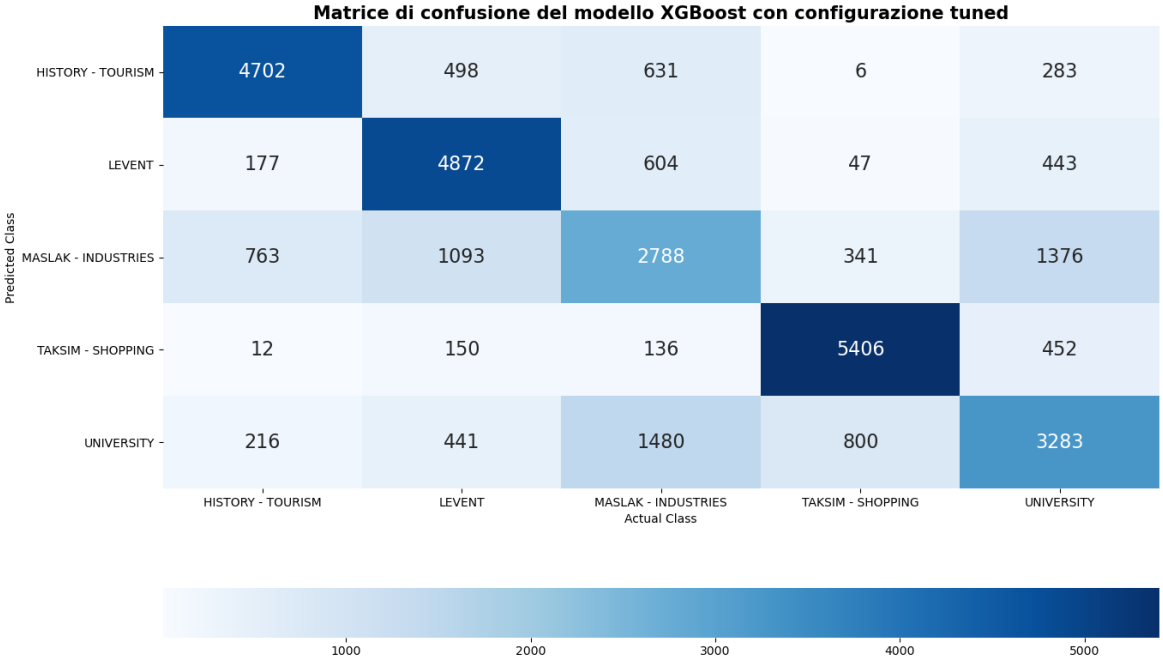


Figura 15: Matrice di confusione del modello XGBoost con i parametri ottimizzati

Anche in questo caso il modello continua a commettere molti errori nel riconoscere correttamente le classi 2 (Maslak) e 4 (University).

Per dimostrare che la causa principale del rendimento non esaltante dei modelli basati su Gradient Boosting sia da ritrovarsi nella somiglianza tra le classi UNIVERSITY e MASLAK si sono confrontati gli andamenti temporali medi di tutti gli hotspot ed in particolare di questi ultimi e di LEVENT: si nota una forte somiglianza nel pattern della concentrazione oraria di persone in queste due zone, che quindi effettivamente mette in difficoltà il modello nel riconoscimento di questi spot.

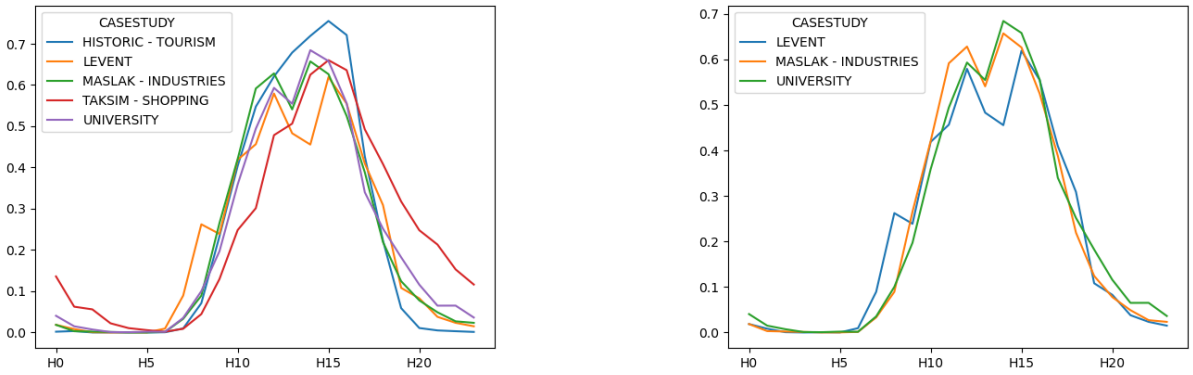


Figura 16: a sinistra il confronto tra tutte le serie temporali, a destra solo Maslak, Levent e University

Conclusioni

I risultati ottenuti in questo studio mostrano che l'approccio seguito offre dei buoni risultati se confrontato con altri metodi generalizzabili come Random Forest, visto che i modelli Gradient Boosting, utilizzando i parametri di default, ottengono un'accuratezza leggermente superiore, che diventa superiore di quasi un punto percentuale se andiamo a considerare i modelli ottimizzati.

A livello di performance, sussiste sempre una certa differenza tra i modelli proposti e quelli generati mediante un approccio stacking, che hanno un'accuratezza fino al 20% superiore rispetto agli algoritmi utilizzati nello studio.

Da questa analisi si può quindi dedurre che, per quanto i modelli basati su Gradient Boosting siano sufficientemente precisi nel problema affrontato, rimangono meno performanti rispetto ai modelli con approccio stacking, specialmente nelle situazioni in cui sono presenti casi di studio simili (come Maslak e University).

I modelli gradient boosting risultano quindi consigliati in situazioni in cui si necessita di svolgere analisi rapide, vista la velocità impiegata dall'algoritmo nel training della rete, e rimangono consigliati anche in caso in cui il caso di studio cambi spesso, vista la forte generalizzazione che caratterizza il modello.

APPROCCIO	ACCURATEZZA MEDIA	VARIANZA
RANDOM FOREST	0.62	
ONE VS. ONE	0.935	0.0032
ONE VS. REST	0.8025	0.0053
GRADIENT BOOSTING	0.6927	0.0026
XGBOOST	0.6845	0.0026

Tabella 14:accuratezza di tutti gli approcci discussi nello studio

Appendice

In questa sezione sono indicate le librerie utilizzate, con riferimento alla documentazione, e il codice di implementazione delle classi descritte nel capitolo *Design e implementazione*. Il codice è scritto in linguaggio Python 3.9

Librerie:

- Pandas: https://pandas.pydata.org/docs/
- Matplotlib: https://matplotlib.org/stable/contents.html
- NumPy: https://numpy.org/doc/
- Seaborn: https://seaborn.pydata.org/
- Scikit-learn: https://scikit-learn.org/stable/
- XGBoost: https://xgboost.readthedocs.io/en/latest/
- Optuna: https://optuna.readthedocs.io/en/stable/
- Plotly: https://plotly.com/python/

A seguire verrà mostrato il codice delle quattro classi presentate e una funzione di utility usata per la stampa dei dati e il plotting della matrice di confusione.

a. GBClassifier
b. XGBoostClassifier
c. XGBoostTuning
d. DataPreProcessing
e. Utility

Appendice A: classe GBClassifier

```
from pandas import read_csv
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import GridSearchCV


class GBClassifier:
    def __init__(self, path):
        self.data = read_csv(path)
        self. confusion_matrix = np.zeros((5, 5))
        self.scoreList = []

    def training_testing(self, rep, settings):
        # split del dataframe in training e testing
        x = self.data.iloc[:, 1:]
        y = self.data.iloc[:, 0]
        train_data, test_data, train_label, test_label = train_test_split(x, y)

        # addestramento del modello Gradient Boosting a seconda del tipo di
richiesta: default o tuned
        if settings == 'default':
            modello = GradientBoostingClassifier()

        else:
            modello = GradientBoostingClassifier(
                        n_estimators=200,
                        learning_rate=0.5,
                        max_depth=2,
                        min_samples_leaf=3,
                        random_state=0,
                        max_features='sqrt'
                        )
        modello.fit(train_data, train_label)

        # calcolo lo score e lo salvo in un array per il calcolo delle
statisitche finali
        prediction = modello.predict(test_data)
        test_score = accuracy_score(test_label, prediction)
        testo = "\n Test " + str(rep) + "; score: " + str(test_score)
        path = "results/GradientBoosting/scores.txt"
        file = open(path, "a")
        file.write(testo)
        self.scoreList.append(test_score)

        cm = confusion_matrix(test_label, prediction)
        self.confusion_matrix = np.add(self.confusion_matrix, cm)

    def grid_search(self, rep, params):
        # split del dataframe in training e testing
        x = self.data.iloc[:, 1:]
        y = self.data.iloc[:, 0]
        train_data, test_data, train_label, test_label = train_test_split(x, y)
        # effettuo la grid search e stampo i parametri selezionati dalla ricerca
        gsearch = GridSearchCV(estimator=GradientBoostingClassifier(),
param_grid=params)
        gsearch.fit(train_data, train_label)
        print(str(gsearch.best_params_))
        print(str(gsearch.best_score_))
```

- **Training_testing:** metodo che lancia il training del modello e successivamente effettua il test e l'aggiornamento degli array di valutazione
- **Grid_search:** metodo che effettua la grid search dei parametri ottimi e stampa i parametri selezionati

Appendice B: classe XGBoostClassifier

```
from pandas import read_csv
import numpy as np
from sklearn.model_selection import train_test_split
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import confusion_matrix, accuracy_score


class XGBoostClassifier:
    def __init__(self, path):
        self.data = read_csv(path)
        self. confusion_matrix = np.zeros((5, 5))
        self.scoreList = []

    def training_testing(self, rep, settings):
        # split del dataframe in training e testing
        x = self.data.iloc[:, 1:]
        y = self.data.iloc[:, 0]
        train_data, test_data, train_label, test_label = train_test_split(x, y)

        if settings == 'default':
            modello = XGBClassifier()
        else:
        # addestramento del modello XGBoost
            param = {
                'n_estimators': 115,
                'max_depth': 28,
                'reg_alpha': 1,
                'reg_lambda': 3,
                'min_child_weight': 5,
                'gamma': 0,
                'eta': 0.31,
                'colsample_bytree': 0.11,
                'eval_metric': 'mlogloss',
                'use_label_encoder': False
            }
            modello = XGBClassifier(**param)
        modello.fit(train_data, train_label)

        # calcolo lo score e lo salvo in un file e in un array per il calcolo
della varianza
        prediction = modello.predict(test_data)
        test_score = accuracy_score(test_label, prediction)
        testo = "\n Test " + str(rep) + "; score: " + str(test_score)
        path = "results/xgBoost/scores.txt"
        file = open(path, "a")
        file.write(testo)
        self.scoreList.append(test_score)

        cm = confusion_matrix(test_label, prediction)
        self.confusion_matrix = np.add(self.confusion_matrix, cm)
```

- **Training_testing:** metodo che lancia il training del modello e successivamente effettua il test e l'aggiornamento degli array di valutazione

Appendice C: classe XGBoostTuning

```
import optuna
from optuna import Trial
from optuna.samplers import TPESampler
from optuna.visualization import plot_optimization_history, plot_slice
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from pandas import read_csv
import plotly


class XGBoostTuning:
    def __init__(self, path):
        self.data = read_csv(path)
        self.study = optuna.create_study(direction='maximize',
sampler=TPESampler())

    def launch_study(self):
        x = self.data.iloc[:, 1:]
        y = self.data.iloc[:, 0]
        self.study.optimize(lambda trial: self.objective(trial, x, y),
n_trials=500)

        print('Best trial: score {},\nparams
{}'.format(self.study.best_trial.value, self.study.best_trial.params))

        plotly.io.show(plot_optimization_history(self.study))
        plotly.io.show(plot_slice(self.study))

    def objective(self, trial: Trial, x, y) -> float:

        train_x, test_x, train_y, test_y = train_test_split(x, y,
random_state=0)

        param = {
            'n_estimators': trial.suggest_int('n_estimators', 20, 1000),
            'max_depth': trial.suggest_int('max_depth', 2, 25),
            'reg_alpha': trial.suggest_int('reg_alpha', 0, 5),
            'reg_lambda': trial.suggest_int('reg_lambda', 0, 5),
            'min_child_weight': trial.suggest_int('min_child_weight', 0, 5),
            'gamma': trial.suggest_int('gamma', 0, 5),
            'eta': trial.suggest_loguniform('eta', 0.005, 0.5),
            'colsample_bytree':
trial.suggest_discrete_uniform('colsample_bytree', 0.1, 1, 0.01),
            'eval_metric': 'mlogloss',
            'use_label_encoder': False
        }

        model = XGBClassifier(**param)
        model.fit(train_x, train_y)
        return cross_val_score(model, test_x, test_y).mean()
```

- **Launch_study**: metodo che lancia uno study, stampa i risultati a schermo e mostra i grafici
- **Objective**: metodo lanciato ad ogni trial dello study, esegue un testing con i parametri scelti dalle funzioni *suggest*

Appendice D: DataPreProcessing

```
from pandas import read_csv
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
```

```
class DataPreProcessing:
    def __init__(self, path):
        self.data = read_csv(path)
        self.processedData_path = None

    def preProcessing(self):
        # elimino le colonne MONTH e DAY
        self.data = self.data.drop(['MONTH', 'DAY'], axis=1)

        # elimino le righe dei weekend (weekend = 1) e la relativa colonna
        self.data = self.data.loc[self.data['WEEKEND'] == 0]
        self.data = self.data.drop(['WEEKEND'], axis=1)

        # elimino tutte le righe vuote (tutti i valori a 0
        self.data = self.data.loc[
            (self.data[['H0', 'H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7',
                        'H8', 'H9', 'H10', 'H11', 'H12', 'H13', 'H14', 'H15',
'H16', 'H17',
                        'H18', 'H19', 'H20', 'H21', 'H22', 'H23']] !=
0).any(axis=1)]

        # numero i case study
        encoder = LabelEncoder()
        self.data['CASESTUDY'] = encoder.fit_transform(self.data['CASESTUDY'])

    def dataNormalization(self):
        scaler = MinMaxScaler()

        # seleziono solo i valori orari (da H0 ad H23)
        x = self.data.iloc[:, 1:]
        self.data[['H0', 'H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7', 'H8',
                   'H9', 'H10', 'H11', 'H12', 'H13', 'H14', 'H15', 'H16', 'H17',
'H18',
                   'H19', 'H20', 'H21', 'H22', 'H23']] =
scaler.fit_transform(x.T[:]).T

        self.processedData_path = 'data/processedData.csv'
        self.data.to_csv(self.processedData_path, index=False)
```

- **PreProcessing:** eliminazione delle colonne e delle righe superflue per lo studio tramite i modelli. Trasformazione dei nomi delle zone in numeri (encoder).
- **Normalization**: normalizzazione dei dati tramite la funzione MinMaxScaler.

Appendice E: funzione di utility

```
import matplotlib.pyplot as plt
import seaborn as sn
from pandas import DataFrame
from numpy import mean, var


def printResults(scoreList, confusionMatrix, model, conf):
    # stampa di valor medio, varianza, valore massimo e minimo dei test svolti
    print('Score medio: {}\nDeviazione standard: {}\nMiglior risultato:
{}\nPeggior risultato: {}'.format(
        mean(scoreList), var(scoreList), max(scoreList),
        min(scoreList))
    )
    # plotting della matrice di confusione
    casestudy = ['HISTORY - TOURISM', 'LEVENT', 'MASLAK - INDUSTRIES', 'TAKSIM -
```

```
SHOPPING', 'UNIVERSITY']
    df_cm = DataFrame(confusionMatrix, index=casestudy, columns=casestudy)
    plt.figure(figsize=(15, 10))
    sn.heatmap(df_cm, annot=True, cbar_kws={"orientation": "horizontal"},
               annot_kws={"fontsize": 16}, cmap='Blues', fmt='g')
    titolo = "Matrice di confusione del modello " + model + " con configurazione
" + conf
    plt.title(titolo, fontweight='bold', fontsize=15)
    plt.xlabel("Actual Class")
    plt.ylabel("Predicted Class")
    plt.show()
```

Il metodo **printResults** stampa a video l'accuratezza media, la varianza, il valore massimo e il valore minimo del test eseguito, e mostra la matrice di confusione.

Appendice F: classe ModelLaunch

```
from xgBoost import XGBoostClassifier
from gradientboosting import GBClassifier
from utility import printResults
from preprocessing import DataPreProcessing


class ModelLaunch:
    def __init__(self, data_path):
        preProcessing = DataPreProcessing(data_path)
        preProcessing.preProcessing()
        preProcessing.dataNormalization()
        self.path = preProcessing.processedData_path

    def launch(self, model):
        if model == 'Gradient Boosting':
            classifier = GBClassifier(self.path)
        elif model == 'XGBoost':
            classifier = XGBoostClassifier(self.path)
        else:
            print("Wrong model name\n")
            return
        classifier.training_testing(500, 'tuned')
        printResults(classifier.scoreList, classifier.confusion_matrix, model,
'tuned')
```

il metodo **launch** crea un'istanza della classe del modello passato come parametro, ne esegue addestramento e testing ed infine stampa i risultati.

Bibliografia

1. Mitchell Tom M., *Machine Learning*, McGraw-Hill, 1997
2. Tekouabou S. C. K., Abdellaoui Alaoui E. A., Cherif W., Silkan H., *Improving parking availability prediction in smart cities with IoT andensemble-based model*, Elsevier, 2020
3. Chen Xiqun, Zahiri Majid, Zhang Shuaichao, *Understanding ridesplitting behavior of on-demand rideservices: An ensemble learning approach*, Elsevier, 2017
4. Silva Prasanna, *Riconoscimento di flussi di mobilità urbana con approccio ensemble learning ibrido,* 2020
5. Morucci Davide, *Analisi delle dinamiche di riempimento di hotspot cittadini attraverso classificatori basati su Random Forest*, 2020