Progetto di Sistemi Context Aware:
Smart Parking System
A.A. 2021-2022

Luca Genova 1038843
Marco Benito Tomasone 1038815
Simone Boldrini 1038792

2022-2023

Indice

Capitolo 1

Introduzione

Il parcheggio è una delle sfide più comuni che i conducenti affrontano quotidianamente nelle città. Trovare un posto auto libero in una zona affollata può richiedere molto tempo e aumentare il traffico, causando stress e frustrazione per i conducenti. Per risolvere questi problemi, è stato proposto un sistema di crowdsensing dei parcheggi che monitora la disponibilità dei posti auto in una zona specifica della città di Bologna. Il sistema fornisce informazioni sul numero di posti auto liberi, aiutando i conducenti a risparmiare una quantità notevole di tempo.
Il sistema si basa sul riconoscimento dell'attività dell'utente e della posizione per determinare lo stato di guida o cammino dell'utente e fornire informazioni sullo stato dei parcheggi disponibili. Questo è il cuore del progetto e consente di tenere conto degli eventi di ingresso (WALKING → DRIVING) e uscita (DRIVING → WALKING) dall'auto.
L'esecuzione di questo progetto è stato suddiviso in due fasi:

- La prima fase consiste nel realizzare il sistema di parcheggio intelligente che monitora la disponibilità dei posti auto in una zona specifica della città di Bologna. Esso è diviso in tre parti:
  - **Un'app mobile**: l'applicazione mobile che permette ai conducenti di visualizzare le informazioni sullo stato dei posti auto in una zona specifica. La funzionalità dell'app è basata sul riconoscimento della posizione e dell'attività dell'utente.
  - **Un frontend**: il sito web che permette agli amministratori di visualizzare le informazioni sulle richieste di parcheggio in una zona specifica, di utilizzare l'algoritmo K-means per visualizzare gli eventi di parcheggio in cluster e anche di visualizzare la heatmap di essi.
  - **Un backend**: il server che gestisce le richieste dell'applicazione e fornisce le informazioni sullo stato dei posti auto interrogando un database PostgreSQL (con estensione PostGIS) tramite query spaziali.

- La seconda fase consiste nell'analizzare le prestazioni di tre diversi algoritmi di clustering: K-means, Random Forest e Gaussian Naive Bayes. Questi algoritmi vengono, poi, confrontati con le prestazioni del sistema integrato all'interno dell'applicazione, per determinare quale sia il più adatto per essere implementato all'interno di essa. In questo modo, è possibile ottenere una soluzione più precisa e affidabile per fornire informazioni sullo stato dei posti auto disponibili.

Il presente report, quindi, fornisce una descrizione dettagliata del funzionamento del sistema di parcheggio intelligente e analizza le sue prestazioni e la sua fattibilità. Verrà presentato il design del sistema, le tecnologie utilizzate e i risultati delle prove effettuate. Inoltre, verrà fornita una valutazione complessiva del sistema e verranno proposte possibili soluzioni per eventuali problemi riscontrati.

Capitolo 2

Progettazione

La prima fase di questo progetto ha riguardato lo studio dei requisiti al fine di scegliere le migliori tecnologie da utilizzare. Nello specifico:
L'app mobile deve prevedere:

- Il riconoscimento dell'attività dell'utente
- Il riconoscimento della variazione dell'attività di un utente (WALKING-> DRIVING e viceversa)
- La possibilità di visualizzare una mappa con le varie aree poligonali di Bologna
- La possibilità di chiedere al backend la disponibilità di parcheggi all'interno di una certa area poligonale
- Il riconoscimento della posizione dell'utente
- L'invio al server degli eventi di entrata e uscita di un parcheggio

Il server deve prevedere:

- La possibilità di ricevere e rispondere alle richieste di disponibilità di parcheggi da parte dell'app
- La possibilità di ricevere le richieste di entrata e uscita di parcheggi da parte dell'app
- La possibilità di inviare le aree poligonali di Bologna all'app
- In caso di zone prive di dati (ad esempio per mancanza di utilizzo dell'app da parte di utenti in quella zona) il backend deve prevedere un meccanismo di interpolazione dei dat, per stimare il numero di parcheggi disponibili

Il frontend deve prevedere:

- Visualizzare, per ogni zona della città, il numero totale di richieste di parcheggio provenienti da quella zona.
- Utilizzare il clustering K-Means per visualizzare gli eventi di parcheggio.
- Representare gli eventi di parcheggio mediante una heatmap.

Dopo aver studiato i requisiti siamo passati alla scelta dei linguaggi e delle librerie da utilizzare.
Per offrire un'app mobile che possa essere utilizzata sia su sistema operativo Android che iOS abbiamo deciso di utilizzare il framework *Flutter*, un framework open-source sviluppato di Google che si poggia su *Dart*. Il progetto è stato però testato solo su android poichè nessun componente del gruppo possedeva un dispositivo con MacOS. Per la gestione della mappa abbiamo utilizzato il pacchetto *flutter_map* che permette di integrare comodamente delle mappe Leaflet all'interno delle propria app, mentre per quanto riguarda la libreria per fare Activity Recognition abbiamo utilizzato un pacchetto chiamato *flutter_Activity_Recognition*, un wrapper alle librerie di sistema Android e iOs per fare riconoscimento dell'attività dell'utente.
Per il backend abbiamo deciso di utilizzare *Node.js*, un runtime Javascript che permette di creare applicazioni web veloci e scalabili, il Database invece, è stato creato utilizzando *PostgreSQL* con estensione *PostGIS* per la gestione dei dati spaziali.
Per motivi di riusabilità del codice e dei componenti abbiamo deciso di utilizzare il framework React per il frontend, un framework *Javascript* open-source sviluppato da Facebook che permette di creare componenti riutilizzabili. Per la gestione delle mappe sia nel frontend abbiamo utilizzato la libreria *Leaflet*, un framework Javascript open-source che permette di creare mappe interattive, dal quale abbiamo preso anche due pacchetti per fare Clustering e per visualizzare la Heatmap dei dati. Infine per la parte grafica è stato utilizzato il framework *Material-UI*, un framework Javascript open-source che permette di creare componenti grafici in stile *Material Design*.

Capitolo 3

Valutazione degli Algoritmi di Classificazione

Nella presente fase del progetto, è stata effettuata la valutazione di tre algoritmi di classificazione: **KNN**, **Random Forest** e **Gaussian Bayes**. L'obiettivo era quello di eseguire un confronto tra i tre algoritmi e di determinare quale di essi fosse più preciso nel classificare i dati utilizzando il dataset HAR fornito.
La valutazione è stata effettuata attraverso la misurazione dell'accuratezza degli algoritmi e il confronto dei risultati ottenuti. Questo ha permesso di determinare quale algoritmo fosse il più adatto per il problema di riconoscimento delle attività svolte dall'utente.

3.1 Dataset, data mining e tecnologie utilizzate

Il dataset utilizzato per l'analisi ci è stato fornito dal professor Marco Di Felice. Il dataset è composto da 62.584 osservazioni e 13 colonne riguardanti il valore dei sensori di uno smartphone android e l'attività dell'utente, come seguente:

accelerometer#mean	la media delle osservazioni dell'accelerometro.
accelerometer#min	l'osservazione minima dell'accelerometro.
accelerometer#max	l'osservazione massima dell'accelerometro.
accelerometer#std	la deviazione standard delle osservazioni dell'accelerometro.
gyroscope#mean	la media delle osservazioni del giroscopio.
gyroscope#min	l'osservazione minima del giroscopio.
gyroscope#max	l'osservazione massima del giroscopio.
gyroscope#std	la deviazione standard delle osservazioni del giroscopio.
gyroscopeuncalibrated#mean	la media delle osservazioni del giroscopio non calibrato.
gyroscopeuncalibrated#min	l'osservazione minima del giroscopio non calibrato.
gyroscopeuncalibrated#max	l'osservazione massima del giroscopio non calibrato.
gyroscopeuncalibrated#std	la deviazione standard delle osservazioni del giroscopio non calibrato.
target	l'attività svolta dall'utente.

Le etichette di questo dataset assumono 5 valori differenti:

- **STILL**: l'utente non si muove.
- **WALKING**: l'utente cammina.
- **CAR**: l'utente è in auto.
- **BUS**: l'utente è in bus.
- **TRAIN**: l'utente è in treno.

3.1.1 Data mining

Per il data mining è stato utilizzato il linguaggio di programmazione *Python* e per il training e prediction dei modelli la libreria *Scikit-learn*. Inoltre, durante la fase di preprocessing e per la gestione dei dati è stata utilizzata la libreria di Python *Pandas*.
Non è stato fatto un grande preprocessing dei dati, in quanto il dataset è stato fornito abbbastanza pulito. I dati sono stati lavorati al fine di eliminare i valori NaN e di normalizzare i dati, scalandoli utilizzando la funzione *MinMaxScaling*. Inoltre sono state eliminate tutte le tuple che non riguardavano le attività di interesse (WALKING e DRIVING).
Per tutti e tre gli algoritmi il dataset è stato suddiviso in tre parti:

- **Training**: è stato utilizzato per il training del modello.
- **Testing**: è stato utilizzato per il testing del modello.

Per il training è stato utilizzato l'80% dei dati e per il testing il 20%.

3.2 Gli algoritmi

Come già detto precedentemente, in questa fase di valutazione degli algoritmi di classificazione, sono stati considerati tre differenti approcci: il **KNN**, il **Random Forest** e il **Gaussian Naive Bayes**.
Questi algoritmi rappresentano tre approcci distinti alla risoluzione del problema di classificazione, ciascuno con i suoi punti di forza e limiti. Il **KNN** utilizza una logica basata sulla vicinanza dei dati, il **Random Forest** è un algoritmo di ensemble che utilizza molteplici alberi di decisione e il **Gaussian Naive Bayes** si basa sull'assunzione che le feature sono distribuite normalmente.
In questa sezione, verranno descritti in dettaglio ciascuno di questi algoritmi e verrà presentato un confronto tra i loro risultati e le loro prestazioni.

3.2.1 KNN

Per quanto riguarda il KNN è stato deciso di effettuare un tuning degli iperparametri per il KNN, con lo scopo di ottenere una prestazione ottimale del modello.
I parametri che sono stati modificati sono stati:

- **n_neighbors**: rappresenta il numero di vicini considerati nella classificazione
- **weights**: determina il peso dei vicini nella classificazione
- **leaf-size**: rappresenta la dimensione massima di una foglia dell'albero di ricerca
- **p**: determina la potenza della metrica utilizzata nella calcoli della distanza tra i vicini

Il tuning degli iperparametri è stato effettuato utilizzando la funzione *GridSearchCV* di *Scikit-learn*. Questa funzione permette di eseguire una ricerca su una griglia di valori per gli iperparametri, testando tutte le possibili combinazioni e selezionando quella che produce la migliore prestazione in termini di accuratezza.
L'obiettivo di questa ottimizzazione era quello di migliorare la precisione del modello KNN e di fornire una soluzione ottimale per la classificazione dei dati. Il risultato ottenuto ha dimostrato che l'ottimizzazione degli iperparametri è stata efficace e ha contribuito a migliorare la performance del modello.

Risultati

I risultati del tuning degli iperparametri utilizzando il dataset precedentemente descritto sono riportati nella tabella seguente:

	Best value
n_neighbors	8
weights	distance
leaf_size	1
p	2

Con questi parametri siamo riusciti ad ottenere un'accuratezza finale del 89%.

3.2.2 Random Forest

Come per il KNN, anche per il Random Forest è stato effettuato un tuning degli iperparametri, sempre con *GridSearchCV*, per ottenere una prestazione ottimale del modello. I parametri che sono stati modificati sono stati:

- **n_estimators**: che rappresenta il numero di alberi decisionali nella foresta
- **max_features**: che rappresenta il numero di features da considerare quando si cerca la migliore divisione
- **max_depth**: che rappresenta la profondità massima degli alberi decisionali
- **min_samples_split**: che rappresenta il numero minimo di campioni richiesti per dividere un nodo interno
- **min_samples_leaf**: che rappresenta il numero minimo di campioni richiesti per costruire una foglia dell'albero
- **bootstrap**: che rappresenta la modalità di campionamento dei dati

Il tuning degli iperparametri è stato effettuato utilizzando la funzione *RandomizedSearchCV* di *Scikit-learn*.
L'obiettivo di questa ottimizzazione era quello di migliorare la precisione del modello Random Forest e di fornire una soluzione ottimale per la classificazione dei dati. Il risultato ottenuto ha dimostrato che l'ottimizzazione degli iperparametri è stata efficace e ha contribuito a migliorare la performance del modello. La RandomizedSearchCV è stata eseguita per un numero di 100 iterazioni, usando una 5-fold cross validation. Per un totale di 500 test.

Risultati

I risultati del tuning degli iperparametri utilizzando il dataset precedentemente descritto sono riportati nella tabella seguente:

	Best value
n_estimators	1800
max_depth	50
min_samples_split	2
min_samples_leaf	4
max_features	auto
bootstrap	False

Con questi parametri siamo riusciti ad ottenere un'accuratezza finale del 93.5% utilizzando il dataset su cui non è stato applicato il MinMaxScaling.

3.2.3 Gaussian Naive Bayes

Per quanto riguarda il Gaussian Naive Bayes, abbiamo deciso di utilizzare questo algoritmo per la classificazione dei dati in quanto si tratta di un metodo semplice e veloce per la classificazione basato sull'assunzione che le feature siano distribuite secondo una distribuzione normale.
In questo caso, non era necessario effettuare un tuning degli iperparametri, in quanto il Gaussian Naive Bayes non ha parametri che possono essere ottimizzati per migliorare le sue prestazioni.

Risultati

Il modello Gaussian Naive Bayes ha fornito un'accuratezza del 81.82%, che risulta essere un risultato inferiore rispetto ai modelli KNN e Random Forest.

3.3 Confronto dei risultati

Illustriamo di seguito i risultati completi dei tre modelli precedenti, ottenute utilizzando il dataset precedentemente descritto.

Evaluating algorithms

	KNN			RF			GNB			
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**Support**
0	0.91	0.99	0.95	0.92	0.94	0.93				1220
1	0.23	0.02	0.04	0.95	0.93	0.94				1275
Accuracy	0.8927855711422845			0.9350701402805611						3992
Macro AVG	0.89	0.89	0.89	0.94	0.94	0.94				2495
Weighted AVG	0.89	0.89	0.89	0.94	0.94	0.94				2495

Tabella 3.1: P = Precision, R = Recall e F1 = F1-score

Bibliografia

[1] Flutter official page. `https://flutter.dev/`.

[2] flutter$_{a}ctivity_{r}ecognitionpackage.$.

flutter$_{m}appackage.$.

Node.js official page. `https://nodejs.org/en/`.

React leaflet official page. `https://react-leaflet.js.org/`.

React official page. `https://it.reactjs.org/`.