Capitolo 1

Algoritmi di classificazione e Risultati

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.

1.1 Dataset e tecnologie utilizzate

Il dataset utilizzato per l'analisi ci è stato fornito dal prof. 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.

1.1.1 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 fornitoci era già di ottima qualità. Per la rimozione dei valori mancanti, abbiamo rimosso solo le righe per le quali mancavano almeno due attributi per ogni classe di sensori. Questo ci ha portato a perdere circa 5000 tuple. Per quelle righe in cui era presente solamente un valore mancante questo è stato sostituito con la media. 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 due 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%.

1.2 Gli algoritmi

Come già detto precedentemente, in questa fase di valutazione degli algoritmi di classificazione, sono stati considerati tre differenti algoritmi: 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.

1.2.1 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	14
weights	distance
leaf_size	1
p	2

Con questi parametri siamo riusciti ad ottenere un'accuratezza finale dell'89%.
Facendo training di questo modello con il dataset su cui abbiamo applicato il MinMax-Scaling, abbiamo ottenuto un'accuratezza del 90%.
I parametri risultanti dal tuning sono invece:

	Best value
n_neighbors	12
weights	distance
leaf_size	1
p	2

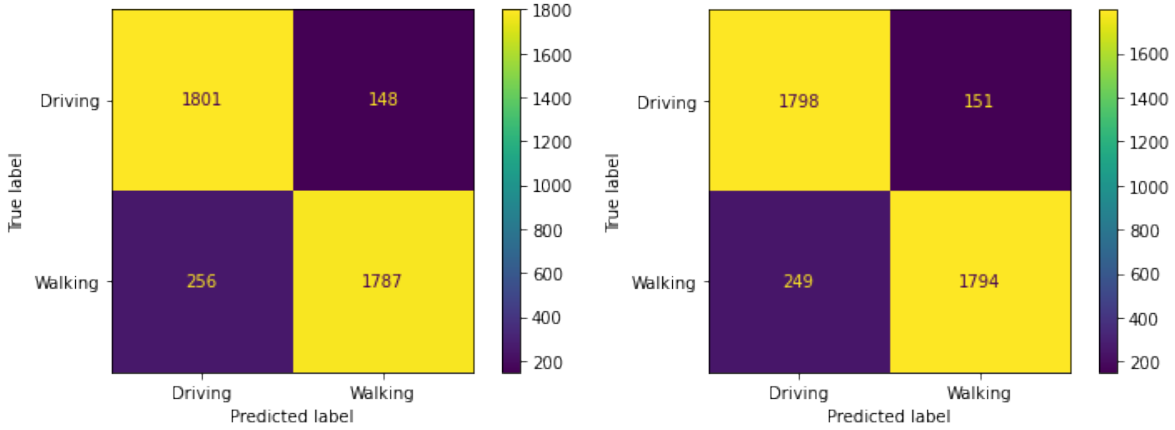

Figura 1.1: Matrice di confusione del modello senza scaling (sx) e con scaling (dx)

Si può notare come questo modello commetta più facilmente errori nel riconoscere la classe DRIVING.

1.2.2 Random Forest

Come per il KNN, anche per il Random Forest è stato effettuato un tuning degli iperparametri 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 fit. In questo caso è stata preferita una RandomizedSearchCV rispetto ad una GridSearchCV poichè il numero di combinazioni di iperparametri da testare è molto elevato e testarli tutti avrebbe richiesto un tempo di esecuzione molto elevato rispetto alla potenza di calcolo disponibile.

Risultati

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

	Best value
n_estimators	800
max_depth	90
min_samples_split	5
min_samples_leaf	1
max_features	sqrt
bootstrap	False

Con questi parametri siamo riusciti ad ottenere un'accuratezza finale del 95.24% utilizzando il dataset su cui non è stato applicato il MinMaxScaling.
Facendo tuning sul modello Random Forest utilizzando il modello con il MinMaxScaling, si è ottenuto un'accuratezza del 89,95%.
I parametri ottenuti dal tuning sono:

	Best value
n_estimators	1400
max_depth	80
min_samples_split	10
min_samples_leaf	2
max_features	sqrt
bootstrap	True

Mostriamo adesso i grafici delle Matrici di confusione ottenute per i due modelli Random Forest, uno con il dataset su cui non è stato applicato il MinMaxScaling e l'altro con il dataset su cui è stato applicato il MinMaxScaling.

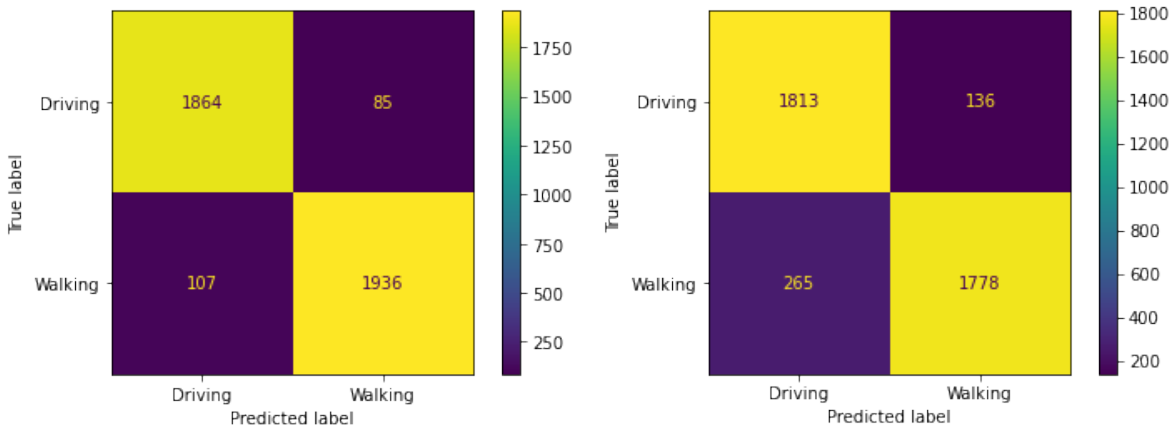

Figura 1.2: Matrice di confusione del modello senza scaling (sx) e con scaling (dx)

Queste matrici di confusione mostrano come il modello Random Forest sia ottimo per la classificazione di questo tipo dati. Sia il modello senza scaling che quello con scaling sono in grado di classificare correttamente la maggior parte dei dati, ma è possibile notare cme riescano a riconoscere più facilmente l'etichetta WALKING, rispetto a quella DRIVING, nella quale è presente un errore maggiore.

1.2.3 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.
Per questo modello il tuning dei parametri è stato effettuato utilizzando la funzione *GridSearchCV* di *Scikit-learn* solo su un'unico parametro *var_smoothing*, che rappresenta la varianza della distribuzione normale.

	Unscaled	**Scaled**
var_smoothing	0.0012328467394420659	0.0003511191734215131

Nonostante ciò le accuracy sono risultate essere identiche per entrambi i modelli. Mostriamo quindi solo una delle due matrici di confunsione ottenute, essendo esse identiche.

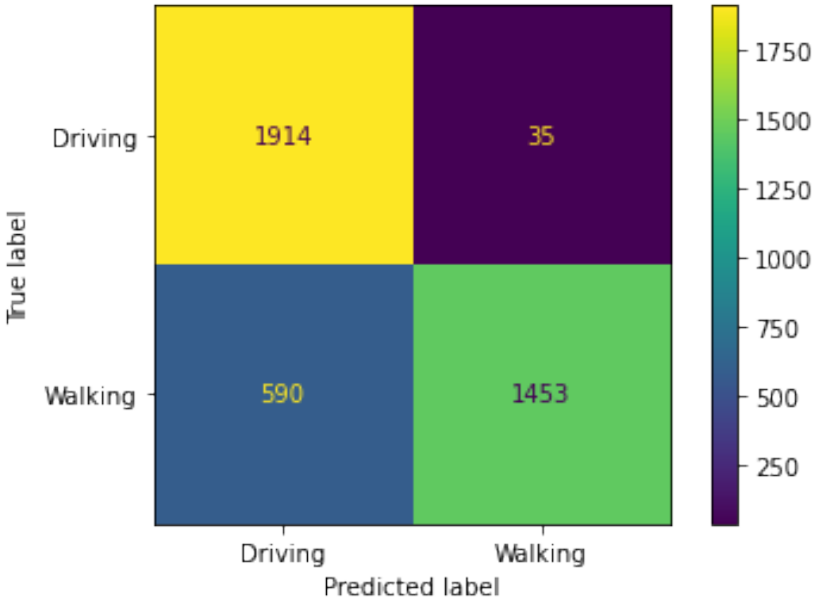

Figura 1.3: Matrice di confusione

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.

1.3 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	0.88	0.92	0.90	0.95	0.95	0.95	0.76	0.98	0.86	1949
1	0.92	0.87	0.90	0.95	0.95	0.95	0.98	0.71	0.82	2043
Accuracy	0.8987975951903807			0.9524048096192385			0.843436873747495			3992
Macro AVG	0.90	0.90	0.90	0.95	0.95	0.95	0.87	0.85	0.84	3992
Weighted AVG	0.90	0.90	0.90	0.95	0.95	0.95	0.87	0.84	0.84	3992

Tabella 1.1: P = Precision, R = Recall e F1 = F1-score

Utilizzando il dataset su cui è stato applicato il MinMaxScaling i risultati sono:

Evaluating algorithms

	KNN			RF			GNB			
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**Support**
0	0.88	0.88	0.92	0.87	0.93	0.90	0.76	0.98	0.86	1949
1	0.92	0.88	0.90	0.93	0.87	0.90	0.98	0.71	0.82	2043
Accuracy	0.8997995991983968			0.8995490981963928			0.843436873747495			3992
Macro AVG	0.90	0.90	0.90	0.90	0.90	0.90	0.87	0.85	0.84	3992
Weighted AVG	0.90	0.90	0.90	0.90	0.90	0.90	0.87	0.84	0.84	3992

Tabella 1.2: P = Precision, R = Recall e F1 = F1-score

Prima di confrontare i tre modelli tra loro, confrontiamo le differenze nell'applicazione per ogni modello dello scaling dei valori. Per quanto riguarda il Gaussian Naive Bayes i risultati sono praticamente identici, mentre per il modello KNN c'è un miglioramento di circa un 0.1% dell'accuratezza. Per il modello Random Forest invece c'è un peggioramento dell'accuratezza di circa il 6%, un risultato molto significativo. Dati i risultati ottenuti abbiamo tralasciato quindi questa tecnica. Riguardo invece al confronto tra i

tre modelli, con ben un 6% di accuratezza in più il modello Random Forest è il modello migliore.

1.4 Accuratezza sistema Har integrato

Dati i precedenti risultati, abbiamo valutato l'accuratezza del sistema di HAR integrato nell'applicazione e offerto dalla libreria *flutter_activity_recognition*. Per farlo, data l'impossibilità di testare la libreria con il dataset utilizzato per testare i modelli precedentemente presentati, abbiamo raccolto un piccolo testbed tramite esperimenti reali. Abbiamo raccolto 1147 record, formati dai dati dei sensori e due colonne target: il target riconosciuto dall'utente (la ground truth) e il target riconosciuto dalla libreria. Proprio il confronto tra questi target ci ha permesso di valutare l'accuratezza del sistema. Il risultato ottenuto è un'accuratezza del **91,02**%. Dato che il dataset è stato raccolto da noi stessi e che abbiamo previsto l'apparizione di un toast sull'app mobile quando la libreria riconosceva un'attività, abbiamo potuto verificare sul campo quali fossero le situazioni che portavano in errore la libreria. Dobbiamo dire che sul riconscimento di un'attività di WALKING o STILL la libreria si presenta solida. Qualche errore è stato commesso durante il riconoscimento di attività IN_VEICHLE, in caso di grandi frenate o curve particolari portavano la libreria per qualche secondo a riconoscere UNKNOWN. Nel grafico mostriamo un istogramma che mette a confronto le tuple per ogni classe target e il numero di predizioni corrette fatte. Come si può notare la maggior parte degli errori è stata commessa nella classe STILL, questo però è dipeso anche dal nostro metodo di raccolta dei dati, in cui raccoglievamo tuple ad intervalli regolari di tempo e non solo quando la libreria riconosceva una nuova attività.

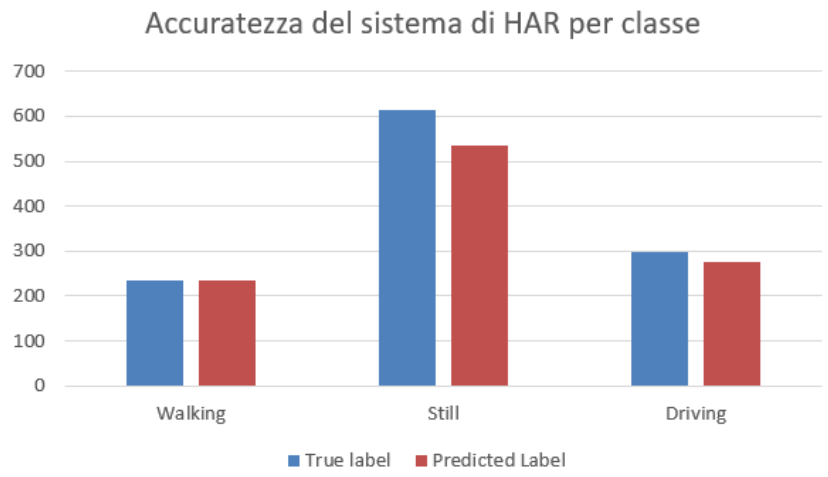

Figura 1.4: Istogramma che mette a confronto le tuple per ogni classe target e il numero di predizioni corrette fatte

1.5 Implementazione del Random Forest nell'app

Dati i risultati ottenuti nella sezione 1.2.2, comparati con i risultati ottenuti dagli altri due modelli, abbiamo scelto il Random Forest come algoritmo di classificazione da implementare direttamente nell'applicazione. Nonostante nella fase precedente è stato allenato un classificatore binario, implementando direttamente nell'applicazione questo tipo di classificatore fa si che l'applicazione non sia in grado di riconoscere quando l'utente è fermo, riconoscendo comunque un'attività. Per motivi di completezza quindi, abbiamo deciso di trasformare il classificatore da binario a ternario riconoscendo WALKING, DRIVING e STILL. Data la poca disponibilità di dati raccolti da noi stessi, questo è stato allenato su una combinazione dei dati raccolti direttamente tramite l'applicazione e dei dati del dataset HAR, discusso nelle sezioni precedenti. Anche il successivo testing è stato fatto su una combinazione dei dati provenienti da queste due diverse fonti.
Data la variazione del dataset, abbiamo deciso di effettuare nuovamente il tuning degli iperparametri del modello.

	Best value
n_estimators	1600
max_depth	None
min_samples_split	2
min_samples_leaf	1
max_features	log2
bootstrap	False

Nella tabella precedente possiamo vedere i migliori valori trovati per ogni iperparametro, mentre nella successiva mostriamo i risultati ottenuti dalla fase di testing del nostro modello. La tabella presenterà i risultati ottenuti sul test set del dataset HAR ottenuto dal prof e sul test set dei dati reali raccolti direttamente da noi tramite l'applicazione. Ovviamente questo test set si presenta molto piccolo data la nostra impossibilità a raccogliere una grande quantità di dati, quindi la valenza di questi risultati è molto limitata.

Evaluating algorithms

	Dataset HAR				Dataset Real			
	P	**R**	**F1**	**Support**	**P**	**R**	**F1**	**Support**
0	0.90	0.91	0.90	982	0.70	0.81	0.75	26
1	0.93	0.93	0.93	1004	0.81	0.96	0.88	23
2	0.94	0.93	0.94	1125	0.93	0.82	0.87	66
Accuracy	0.9247830279652844			3111	0.8434782608695652			115
Macro AVG	0.92	0.92	0.92	3111	0.82	0.86	0.83	115
Weighted AVG	0.92	0.92	0.92	3111	0.86	0.84	0.85	115

Tabella 1.3: P = Precision, R = Recall e F1 = F1-score

Mostriamo ora per entrambi questi test set le matrici di confusione: