UNIVERSITÀ DI PISA

Riconoscimento delle attività umane tramite smartphone

Progetto di Data Mining 2 a.a. 2021/2022

A cura di:
Ismaele Gorgoglione
Aura Petrucci

Università di Pisa

25 luglio 2022

Indice

1 Introduzione 1

2 Data Understanding 1

3 Basic Classifiers 2

3.1 KNN 2

3.2 Decision Tree 3

3.3 Conclusioni 3

4 Outlier Detection 3

4.1 Local Outlier Factor (LOF) . 3

4.2 Isolation Forest 4

4.3 KNN 5

4.4 Conclusioni 5

5 Sbilanciamento del dataset 5

5.1 Decision Tree sul dataset sbilanciato 5

5.2 KNN sul dataset sbilanciato . 6

5.3 Conclusioni 7

6 Riduzione della dimensionalità 7

6.1 Pricipal Component Analysis (PCA) 7

6.1.1 KNN PCA 7

6.1.2 Decision Tree PCA . . 8

6.2 IsoMap 8

6.2.1 Decision Tree e KNN con IsoMap 8

6.3 T-distributed stochastic neighbor embedding (t-SNE) . . 9

7 Conclusioni generali 9

8 Metodi di classificazione avanzata 10

8.1 Naive Bayes 10

8.2 Regressione Logistica 10

8.3 Support Vector Machines (SVM) 10

8.3.1 SVM lineare 11

8.3.2 SVM non lineare . . . 11

9 Ensemble classifiers 11

9.1 Random Forest 11

9.2 Bagging e Boosting 13

10 Reti neurali 14

10.1 Multilayer Perceptron Classifier (MLP) 14

10.2 Deep neural network 14

11 Regressione Lineare 16

11.1 Regressione con variabili continue 16

11.1.1 Lasso e Ridge Regression 17

11.2 Regressione Lineare Multipla . 17

12 Serie Temporali 17

12.1 Motifs e anomalie 18

12.2 Shapelets 18

12.3 Clustering 19

12.3.1 Shape Based Clustering 19

12.3.2 Feature Based Clustering 20

12.3.3 Approximation Based Clustering 20

12.3.4 Conclusioni 20
12.4 Classification 20
12.4.1 Shapelet Models . . . 21
12.4.2 Time Series Classifier . 21
12.4.3 1-NN Classifier 22
12.4.4 ROCKET e MiniROCKET 22
12.4.5 CIF - Canonical Interval Forest 22
12.4.6 Multivaraite Time Series 22
12.4.7 Conclusioni 23

13 Sequential Pattern Mining e Advanced Clustering 23
13.1 Sequential Pattern Mining . . 23
13.2 Advanced Clustering 25
13.2.1 K-Means e X-Means . 25
13.2.2 OPTICS 25
13.2.3 K-Mode e Rock 26

14 Explainability 27
14.1 SHAP 27
14.2 LIME 28
14.3 Conclusioni 29

1 Introduzione

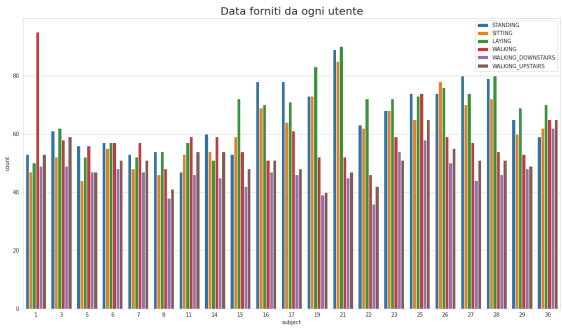
L'obiettivo del presente lavoro è stato quello di realizzare un'analisi di un dataset riguardo il riconoscimento dell'attività umana. L'esperimento che ha portato alla nascita del dataset è stato effettuato da un campione di 30 volontari di età compresa tra i 19 e i 48 anni, i quali hanno svolto sei differenti attività monitorate attraverso l'utilizzo di uno smartphone. Tra le attività eseguite abbiamo:

1. **Walking**;
2. **Walking upstairs**;
3. **Walking downstairs**;
4. **Sitting**;
5. **Standing**;
6. **Laying**.

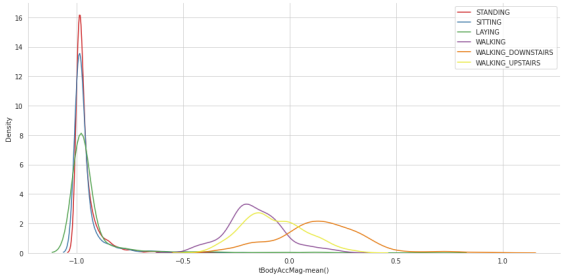
2 Data Understanding

Il dataset si compone di un totale di 10.299 records suddivisi in 561 features: esse sono tutte di tipo numerico (int e float), alle quali sono state aggiunte 3 variabili di tipo intero e stringa corrispondenti ai diversi tipi di attività svolte. In particolare quella di tipo categorico proviene dal file activity_labels all'interno del quale sono specificati i corrispondenti termini indicanti le 6 attività svolte dai soggetti esaminati.
Le features numeriche sono normalizzate in un range che va da un minimo di -1 ad un massimo di 1. Il dataset è stato suddiviso in 4 dataset minori i quali rappresentano la suddivisione in training e test set. In primo luogo è stata realizzata un'analisi descrittiva dei dati, prendendo come riferimento l'intero dataset; a tale scopo è stata effettuata un'indagine sulla distribuzione dei dati riguardati

le differenti attività svolte dai 30 partecipanti allo studio.



(a) Distribuzione dei dati forniti dagli utenti



(b) Densità in base all'accellerazione del corpo per le attività svolte

Figura 1: Valori delle distribuzioni

Come è possibile osservare dalla figura 1 (a) vi è una ripartizione abbastanza omogenea nella distribuzione delle varie attività, con l'eccezione di alcuni dati più elevati per determinati utenti, i quali riguardano principalmente le attività sedentarie; mentre quelle in movimento risultano distribuite omogeneamente, ad esclusione dell'utente 1, il quale mostra dei valori per walking significativamente più elevati rispetto alla media. Nella figura 1 (b), invece, è possibile osservare dei picchi nel livello di densità in relazione all'accellerazione del corpo per le attività di sitting, standing e laying, in corrispondenza del valore -1; mentre per le attività in movimento, l'accellerazione raggiunge valori che variano tra -0.5 e 0.5.

3 Basic Classifiers

Nella presente seconda fase di analisi, è stata realizzata la classificazione attraverso i metodi *K-Nearest Neighbor e Decision Tree*, implementati per osservare la popolarità delle varie attività.

3.1 KNN

In primo luogo è stata realizzata una GridSearch per trovare i migliori parametri da applicare al metodo, in modo tale da aumentare il livello di *Accuracy*. Il miglior valore ottenuto per n_neighbors risulta essere 7; impostandolo sono stati ottenuti i risultati riportati nella tabella sottostante (tab. 1), mostrando una performance del modello particolarmente precisa, con un'*Accuracy* pari a 0.89. In particolare per la classe 6, si osservano dei valori di *Precision, Recall ed F1-score* fin troppo precisi, non a caso la ROC AUC ha un valore di 0.99, questo sta ad indicare che il modello classifica perfettamente tutti i valori; ciò è dovuto probabilmente ad un buon bilanciamento nella distribuzione dei dati.

A conferma della validità del parametro utilizzato, sono stati testati uteriori valori per n_neighbors (5, 10, 15) e mettendo in relazione i risultati ottenuti è stato possibile osservare che l'esito ottimale si ottiene con K=7.

	Precision	**Recall**	**F1score**	**ROC auc**
1	0,70	0,86	0,77	0,95
2	0,84	0,72	0,78	0,90
3	0,83	0,74	0,78	0.94
4	0,78	0,67	0,72	0,95
5	0,74	0,87	0,80	0,97
6	1,00	0,94	0,97	0,99

Tabella 1: Valori del modello KNN

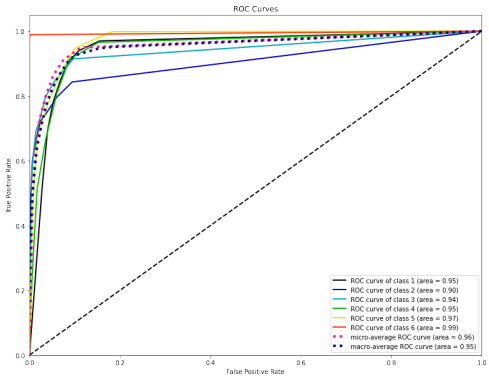


Figura 2: ROC kNN

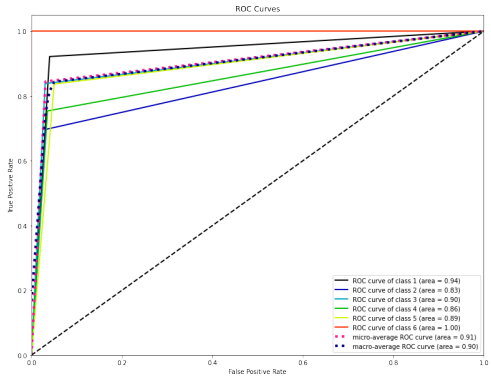


Figura 3: ROC Decision Tree

3.2 Decision Tree

Attraverso la GridSearch sono stati ottenuti come risultati ottimali da utilizzare per il modello:

1. **Depth: None**;
2. **min_sample_leaf: 1**;
3. **min_sample_split: 2**.

Nella tabella (tab. 2) è riportata la performance del modello scegliendo come criterio *entropy*, mostrando dei valori più accurati rispetto a quelli ottenuti attraverso l'applicazione del *KNN*. Da notare la classe 6 (laying), la quale mostra tutti i parametri uguali a 1.00, mentre risultati meno ottimali si hanno per la classe 2 (walking_upstairs).

	Precision	**Recall**	**F1score**	**ROC auc**
1	0,82	0,92	0,87	0,94
2	0,81	0,70	0,75	0,83
3	0,82	0,84	0,83	0.90
4	0,81	0,75	0,78	0,86
5	0,79	0,84	0,81	0,89
6	1,00	1,00	1,00	1,00

Tabella 2: Valori del modello DT

3.3 Conclusioni

Entrambi i modelli per la Classificazione danno dei risultati accurati, sintomo di un dataset ben bilanciato; in ogni caso il *Decision Tree* sembra mostrare dei risultati più precisi, per tal motivo è da preferire rispetto al *KNN*.

4 Outlier Detection

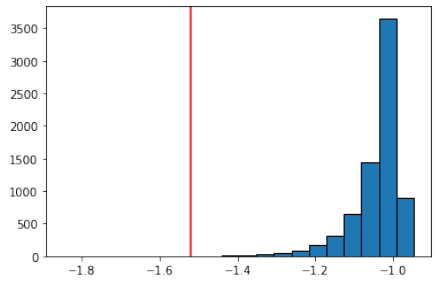
Per la presente fase di analisi, sono stati utilizzati i metodi *LOF, Isolation Forest e KNN* per analizzare la presenza di eventuali outliers nelle features del dataset.

4.1 Local Outlier Factor (LOF)

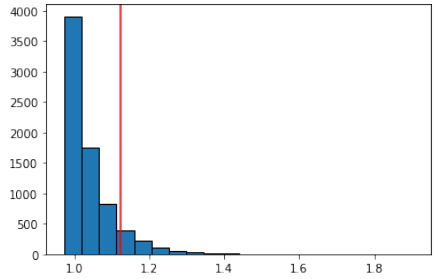
Il metodo *LOF* permette di calcolare la media del rapporto tra la densità locale di un sample e i suoi vicini (k) più contigui. Per l'implementazione di tale metodo sono state utilizzate due librerie differenti (PyOD e scikit-learn), le quali hanno permesso di giungere all'individuazione di outilers, ma riportando risultati notevolmente differenti. Nel caso di scikit-learn è stata implementata una GridSearch per trovare il parametro migliore

per k, ottendendo 5 come risultato ottimale. Applicando n_neighbors = 5, sono stati individuati solamente 10 valori anomali. Attaverso l'implementazione del *LOF* con la libreria pyod, si ottengono invece risulato differenti, contando 658 outliers.
Per quanto riguarda il top 1% outliers, attraverso la libreria pyod, sono stati riscontrati sei valori:

1. **1.905**;
2. **1.896**;
3. **1.829**;
4. **1.814**;
5. **1.740**;
6. **1.694**.



(a) Metodo LOF implementato con la libreria scikit-learn



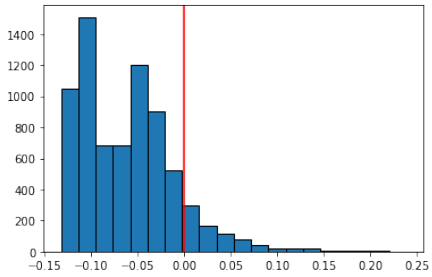
(b) Metodo LOF implementato con la libreria pyod

Figura 4: Outlier detection attraverso il metodo LOF

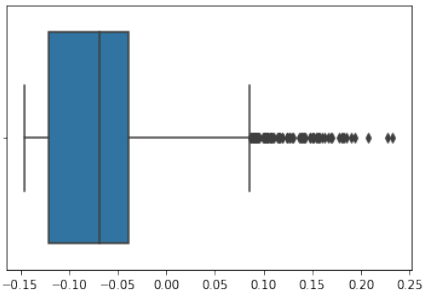
4.2 Isolation Forest

Come secondo metodo è stato implementato l'*Isolation Forest*, tale algoritmo rileva le anomalie basandosi esclusivamente sul concetto di isolamento, senza utilizzare alcuna misura di distanza o densità.

Ugualmente, per il presente metodo sono state utilizzate le due librerie sopracitate. Attraverso l'implementazione dell'algoritmo con scikit-learn si ottengono come risultato 633 valori anomali, d'altra parte invece, con pyod, si conta un totale di 736 outliers; si nota quindi che anche con questo metodo, si ottengono dei risultati più elevati, seppur moderati, rispetto a quelli dell'algoritmo precedente calcolato con la seconda libreria.
Attraverso l'*Isolation Forest* è stato individuato il top 1% di outliers nel range che va da 0.232 a 0.185.



(a) Outlers detection attraverso l'Isolation Forest



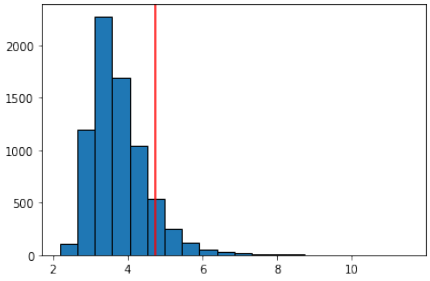
(b) Boxplot degli outliers con l'Isolation Forest

Figura 5: Outliers detection implementando l'Isolation Forest

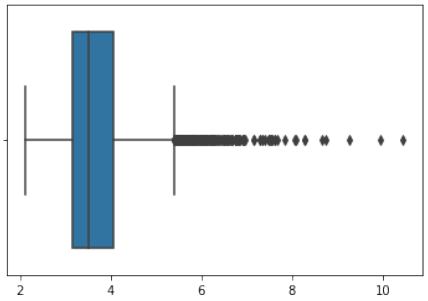
4.3 KNN

L'ultimo metodo implementato per l'outlier detection è un density-based approach, ovvero il *KNN*, il quale si basa sulle distanze dei punti dai loro vicini per trovare eventuali valori anomali. Tale algoritmo è stato implementato attraverso la libreria pyod, ottenendo 622 outliers in totale. Per il top 1% outliers sono stati individuati i sei valori seguenti:

1. **10.442**;
2. **9.937**;
3. **9.255**;
4. **8.732**;
5. **8.651**;
6. **8.286**.



(a) Outlers detection attraverso il KNN



(b) Boxplot degli outliers con KNN

Figura 6: Outlier detection implementando il KNN

4.4 Conclusioni

In base ai risultati ottenuti si è optato per l'eliminazione di alcuni outlier. Tale rimozione è avvenuta attraverso un'intersezione dei risultati ricavati dai differenti metodi di outlier detection (*LOF, Isolation Forest, KNN*). Si è deciso di non utilizzare i risultati ottenuti attraverso il metodo *LOF* con la libreria scikit-learn in quanto si otteneva un numero notevolmente più basso di valori anomali rispetto agli altri. Dunque, in totale sono stati eliminati 91 outliers comuni ai metodi utilizzati.

5 Sbilanciamento del dataset

Nella fase di sblianciamento è stato preso in considerazione un subset del dataset contenente le classi relative alle attività in movimento (walking, walking_downstairs e walking_upstairs); di queste, la classe 1 (walking) è stata sbilanciata passando da 1226 a 200 valori ed ugualmente è stato effettuato lo sbilanciamento della classe 2 (walking_downstairs) passando da 1073 a 9 valori.

5.1 Decision Tree sul dataset sbilanciato

A seguito dello sbilanciamento è stato utilizzato nuovamente il *Decision Tree* per osservare eventuali cambiamenti nella performance del modello di classificazione.

Come è possibile osservare, la classificazione mostra una performance con valori notevolmente meno accurati rispetto al dataset

	Precision	Recall	F1score
1	0,38	0,89	0,53
2	0,54	0,14	0,22
3	0,93	0,93	0,93

Tabella 3: Valori del modello DT sul dataset sbilanciato

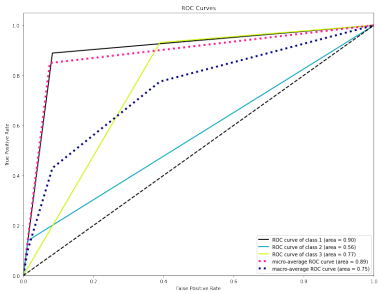


Figura 7: ROC Decision Tree su dataset sbilanciato

precedentemente analizzato, ad eccezione della classe 3 che presenta dei valori per *Precision*, *Recall* e *F1-score* pari a 0.93. Non a caso, tale classe risulta essere l'unica non sottoposta allo sbilanciamento.

Per le classi 1 e 2 si ha una performance più bassa: la seconda ha un valore di *Recall* pari a 0.14. Ciò è chiaramente dovuto al numero notevolmente basso di valori ad essa appartenenti a seguito dello sbilanciamento.

La differenza nelle performance del *Decision Tree* per le tre classi si osserva nella figura 7, dove viene mostrato l'andamento delle ROC per ognuna di esse. Se per la classe 1 si ha un'area sotto la curva pari a 0.90, per la classe 2 tale valore equivale a 0.56. Mentre per la numero 3, la quale presenta i valori di *Precision*, *Recall* e *F1-score* più elevanti, la ROC AUC è di 0.77. Ciò indica che per la seconda classe si hanno delle prestazioni poco significative nella predizione degli elementi ad essa appartenenti.

5.2 KNN sul dataset sbilanciato

Oltre al *Decision Tree*, si è deciso di implementare nuovamente il *KNN* sul dataset sbilanciato. Dai risultati ottenuti e riportati nella tabella sottostante (tab. 4) è evidente, anche in questo caso, un decremento nelle prestazioni del modello di classificazione, principalmente per quel che riguarda le classi sulle quali è avvenuto lo sbilanciamento (1 e 2). Significativi sono i valori di *Precision*, *Recall* e *F1-score* della classe 2, tutti pari a 0.00. Inoltre la classe 3, come con il *Decision Tree*, mostra essere la classe con dei valori di performance più elevati, raggiungendo lo 0.95 con la *Recall*.

	Precision	Recall	F1-score
1	0,37	0,78	0,50
2	0,00	0,00	0,00
3	0,90	0,95	0,93

Tabella 4: Valori del modello KNN sul dataset sbilanciato

Le ROC Curves implementate mostrano dei risultati simili (seppur leggermente più accurati) a quelli ottenuti dal classificatore precedente, in quanto anche in questo caso la classe con le prestazioni più elevate è la numero 1 (ROC AUC = 0.94), mentre nella classe 2 (ROC AUC = 0.63) la relazione tra TPR e FPR è inferiore rispetto alle altre.

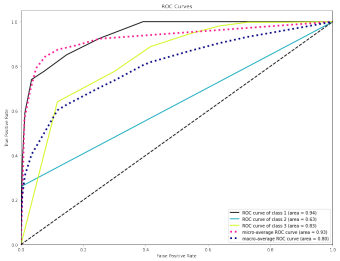


Figura 8: ROC KNN su dataset sbilanciato

5.3 Conclusioni

Mettendo a confronto i due classificatori, anche nel caso del dataset sbilanciato, si ottengono risultati migliori per *Precision*, *Recall* e *F1-score* con il *Decision Tree*; mentre per le ROC Curves si ha una performance migliore implementando il *KNN*. In entrambi i casi si è osservata una somiglianza nell'andamento delle ROC per la classe 1, difatti i valori ottenuti sono più elevati rispetto a quelli delle altre classi. Ciò potrebbe essere dovuto al fatto che le classi 2 e 3 (walking_upstairs e walking_downstairs) risultino essere più simili portando quindi a dei valori di TPR meno elevati rispetto alla classe 1.
Ad ogni modo, è attraverso un dataset bilanciato che si ottengono delle prestazioni più significative da parte di entrambi i modelli di classificazione.

6 Riduzione della dimensionalità

Nella fase successiva del lavoro si è proseguito con la riduzione della dimensionalità, ovvero la trasformazione dei dati da uno spazio ad alta dimensione in uno di dimensione minore, in maniera tale che essa mantenga alcune proprietà significative dei dati originali. Nell'analisi sono state utilizzate 5 diverse tecniche di riduzione della dimensionalità: *PCA*, *t-SNE*, *Random Subspace Projection*, *Multi Dimensional Scaling* e *IsoMap*, delle quali solamente alcune (riportate di seguito) hanno mostrato delle performance interessanti.

6.1 Pricipal Component Analysis (PCA)

L'*Analisi delle Componenti Principali* risulta essere la tecnica di riduzione più interessante; essa funziona derivando, a partire da un set di variabili numeriche correlate, un insieme più ridotto di variabili ortogonali "artificiali". A differenza di *IsoMAp*, si tratta di un metodo che applica una traformazione di tipo lineare. Per la sua applicazione, per mantenere un livello di varianza pari al 95%, si è deciso di utilizzare 65 come numero di componenti. Inoltre, per valutare la performance della *PCA*, sono stati implementati diversi metodi di classificazione.

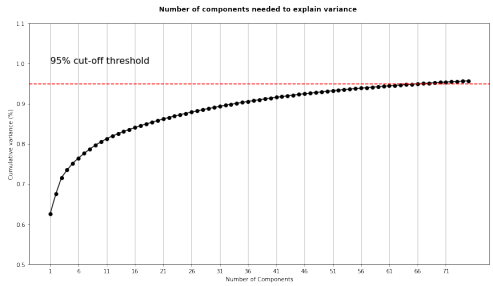


Figura 9: Varianza al variare dei numero dei componenti

6.1.1 KNN PCA

Il *KNN* è il primo metodo di classificazione implementato. Come primo step si è mantenuto il numero ottimale di neighbors iniziale ottenuto attraverso l'applicazione della GridSearch, il quale risulta essere 7. I valori ottenuti attraverso l'attuazione del *KNN* risultano essere ancora precisi, discostandosi leggermente dai valori iniziali, ciò si osserva inoltre attraverso il valore dell'*Accuracy* (0.9), il quale risulta essere equivalente al suo valore originario.

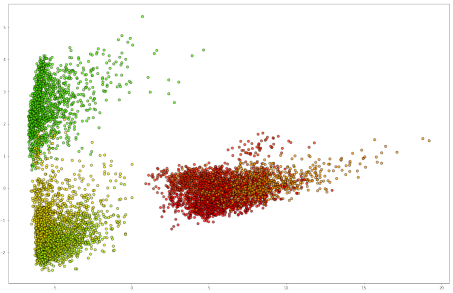


Figura 10: Visualizzazione in 2D della PCA

Per quanto concerne la performance degli altri valori, nella tabella sottostante (tab. 5) si nota che tutti i parametri delle 6 classi, quali *Precision*, *Recall* e *F1-Score*, sono equiparabili ai precedenti, inoltre tale similarità nei risultati si ritrova anche nelle ROC curve.

	Precision	**Recall**	**F1score**	**ROC auc**
1	0,86	0.97	0,91	0,99
2	0,89	0,90	0,90	0,99
3	0,94	0,80	0,86	0.97
4	0,90	0,78	0,83	0,98
5	0,82	0,92	0,87	0,99
6	1,00	1,00	1,00	1,00

Tabella 5: Valori del modello KNN sul dataset ridotto attraverso la PCA

6.1.2 Decision Tree PCA

Implementando il *Decision Tree* sul dataset ridotto attraverso la *Principal Components Analysis* si ottiene una performance meno accurata, seppur comunque buona. Se nel dataset originario si è ottenuta un'*Accuracy* pari a 0.86, in questo caso essa è diminuita fino a raggiungerere lo 0.81, inoltre in molti casi i valori per *Precision*, *Recall* e *F1-Score* risultano essere minori dello 0.80.

	Precision	**Recall**	**F1score**	**ROC auc**
1	0,79	0.91	0,94	0,99
2	0,83	0,75	0,89	0,99
3	0,78	0,72	0,75	0.86
4	0,74	0,73	0,73	0,85
5	0,76	0,80	0,78	0,89
6	0,99	0,95	0,97	0,99

Tabella 6: Valori del modello DT sul dataset ridotto attraverso la PCA

6.2 IsoMap

Successivamente è stato implementato il metodo *IsoMap*, il quale utilizza una riduzione della dimensionalità non lineare. In questo specifico metodo di riduzione della dimensionalità si è optato per ridurre il numero di componenti, ponendolo pari a 3, in quanto si desiderava ottenere una riduzione ancora più marcata rispetto a quella precedente conseguita con la *PCA*. Nella figura che segue (fig.11) è rappresentata una proiezione bidimensionale delle attività a seguito della riduzione attraverso *IsoMap*.

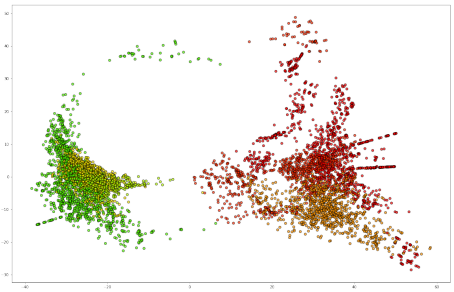
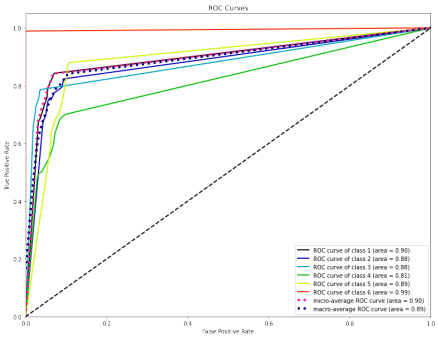


Figura 11: Visualizzazione in 2D di IsoMap

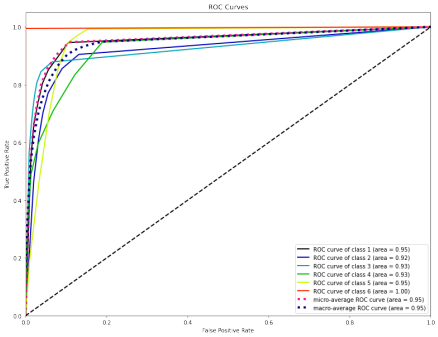
6.2.1 Decision Tree e KNN con IsoMap

Anche in questo caso abbiamo eseguito nuovamente il classification report per vedere le differenze nei risultati: si possono osservare

delle differenze notevoli a seconda del classificatore utilizzato. Se con il *Decision Tree* la classificazione risulta essere non molto precisa con valori di *F1-score* anche inferiori a 0.7 e con un'*Accuracy* appena superiore a 0.80, con il *KNN* invece le cose cambiano ed anche con il dataset ridotto si ottiene una classificazione con dei risultati interessanti, in quanto anche se i valori ottenuti risultano essere inferiori rispetto a quelli della sezione 3, abbiamo comunque degli score superiori a 0.80, con un' *Accuracy* del modello pari a 0.87.
Nella figura 12 la rappresentazione delle ROC dei due classificatori sul dataset a seguito della riduzione della dimensionalità con IsoMap.



(a) ROC DT su dataset ridotto



(b) ROC KNN su dataset ridotto

Figura 12: Visualizzazione delle ROC AUC sul dataset ridotto con IsoMap

6.3 T-distributed stochastic neighbor embedding (t-SNE)

Il *t-SNE*, come l'*IsoMap*, è una tecnica della riduzione della dimensionalità di tipo non lineare. Attraverso questo metodo non è possibile apprendere una trasformazione e riutilizzarla su dati diversi (Train e Test), poiché *t-SNE* non apprende una funzione di mappatura su uno spazio di dimensioni inferiori, ma esegue una procedura iterativa su un sotto spazio per trovare un equilibrio che minimizza una distanza su alcuni dati. Di conseguenza, il metodo non è constatato come adatto per il nostro studio, in ogni caso esso risulta essere interessante a livello di rappresentazione grafica.

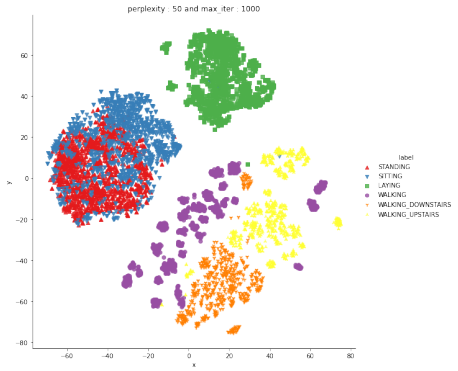


Figura 13: Visualizzazione in 2D della tSNE

7 Conclusioni generali

A seguito di questa prima fase di analisi risulta chiaro un buon bilanciamento del dataset, dimostrabile dai vari risultati ottenuti attraverso l'implementazione dei metodi di classificazione, in particolare del *kNN e del Decision Tree*. Inoltre, a seguito dell'outlier detection, si nota come la quantità di valori

anomali riscontrati attraverso l'implementazione dei vari metodi (*LOF, Isolation Forest e KNN*), sia in realtà bassa, portando difatti all'eliminazione di solo 91 outliers. Infine, la PCA risulta essere il metodo più interessante ai fini del nostro studio per quel che riguarda la riduzione della dimensionalità, in quanto applicando il *kNN ed il Decision Tree* sul dataset ridotto attraverso tale algoritmo, si ottiene la performance migliore.

8 Metodi di classificazione avanzata

L'analisi è proseguita implementando alcuni metodi di classificazione avanzata, tra i quali *Naive Bayes*, *Regressione Logistica, Support Vector Machines, Neural Networks, Ensemble Classifier*. Per ogni classificatore sono stati calcolati i parametri principali di valutazione ed analizzati i cambiamenti al variare dei parametri di ogni modello. Infine, si analizzerà un modello di *Regressione Linear*e e *Multipla* attraverso l'utilizzo di due attributi continui, dove per l'ultimo citato si implementeranno modelli di regressione lineare regolarizzata.

8.1 Naive Bayes

Il primo algoritmo di classificazione implementato è il *Naive Bayes*: si tratta di un classificatore di tipo probabilistico. Considerando la natura del dataset che non contiene alcun attributo categorico, è stato possibile implementare unicamente il *Gaussian Naive Bayes Classifier*. Applicando tale modello al dataset in esame non si ottiene una performance ottimale, ciò si può osservare dai diversi valori di *Recall* ed *F1-score* stimati, in quanto per alcune classi in particolare (3 e 4), raggiungono dei livelli pari a 0.61 per la prima e 0.65 per la seconda.

	Precision	**Recall**	**F1score**
1	0,82	0.84	0,83
2	0,76	0,96	0,84
3	0,83	0,61	0,70
4	0,58	0,75	0,65
5	0,80	0,86	0,83
6	0,96	0,60	0.74

Tabella 7: Valori del classificatore Naive Bayes

8.2 Regressione Logistica

Successivamente, l'esecuzione del metodo della *Regressione Logistica* ha portato all'individuazione dei seguenti risultati:

	Precision	**Recall**	**F1score**
1	0,95	0.99	0,97
2	0,96	0,94	0,95
3	0,99	0,96	0,97
4	0,96	0,88	0,92
5	0,90	0,97	0,93
6	1,00	1,00	1,00

Tabella 8: Valori del classificatore Regressione Logistica

Come è possibile osservare, il modello in questione presenta una performance notevolmente migliore rispetto all'algoritmo di classificazione precedente. L'*Accuracy* raggiunge lo 0.96 e la classe 6 risulta essere classificata con un'accuratezza pari al 100%.

8.3 Support Vector Machines (SVM)

L'obiettivo dell'algoritmo della *Macchina Vettoriale di Supporto* è trovare un iperpiano in uno spazio N-dimensionale (N è il numero di caratteristiche) che classifichi distintamente i punti dati. Per separare le classi

di punti dati, ci sono innumerevoli possibili iperpiani che potrebbero essere scelti. L'obiettivo è stato quello di trovare un piano che abbia il margine massimo, ovvero la distanza massima tra i punti delle classi.

8.3.1 SVM lineare

Per il metodo *SVM lineare* un'interessante analisi riguarda l'andamento dei risultati al variare del parametro C, ovvero la penalità nell'errata classificazione delle istanze. Per determinare il valore ottimale per tale parametro è stata utilizzata una GridSearch, facendo variare i valori di C da 0 a 100. In tal modo è stato ottenuto 66 come miglior valore da implementare, seppur anche con altri valori si ottenevano dei risultati notevolmente accurati.

	Precision	**Recall**	**F1score**
1	0,97	1,00	0,99
2	0,98	0,97	0,97
3	1,00	0,98	0,99
4	0,96	0,88	0,92
5	0,91	0,97	0,94
6	1,00	1,00	1,00

Tabella 9: Valori del classificatore SVM lineare

In questo caso i risutati ottenuti per *Precision*, *Recall* e *F1-score* sono ancor più precisi ed accurati rispetto ai precedenti ottenuti con la *Regressione Logistica*.

8.3.2 SVM non lineare

Anche per l'*SVM non lineare* è stata eseguita una GridSearch per trovare i parametri migliori da implementare. Per quanto riguarda il parametro indicante la funzione non lineare utilizzata *(kernel)*, sono state implementate le tre differenti tipologie (*sigmoid, rbf e poly)*, ottenendo i risultati migliori attraverso la funzione polinomiale. Per quel che riguarda il parametro C i valori più accurati si ottengono ponendo il valore uguale a 100.

	Precision	**Recall**	**F1score**
1	0,95	0,98	0,97
2	0,96	0,96	0,96
3	0,99	0,96	0,98
4	0,98	0,91	0,94
5	0,92	0,98	0,95
6	1,00	1,00	1,00

Tabella 10: Valori del classificatore SVM lineare

9 Ensemble classifiers

Rispetto ai classificatori *ensemble*, sono stati testati i metodi *Random Forest, Bagging e Boosting*. Il *Random Forest* non è altro che una tecnica di apprendimento automatico utilizzata per risolvere problemi di regressione e classificazione; ciò avviene attraverso l'*Ensemble Learning*, ovvero una tecnica che combina più classificatori per fornire soluzioni a problemi complessi. Il *bagging*, o *bootstrap aggregation*, invece, è una tecnica per ridurre la varianza di una funzione di previsione stimata, il quale sembra funzionare particolarmente bene con procedure ad alta varianza e bassa distorsione, come gli alberi.

9.1 Random Forest

Per questo classificatore è stata impostata una GridSearch utilizzando sia il criterio *entropy* che *gini* e facendo variare:

- *n_estimators* per valori compresi fra 1 e 100;
- *max depth* fra 2, 20 e la profondità massima dell'albero (depth = None);
- *min samples split* con valori fra 2 e 100;

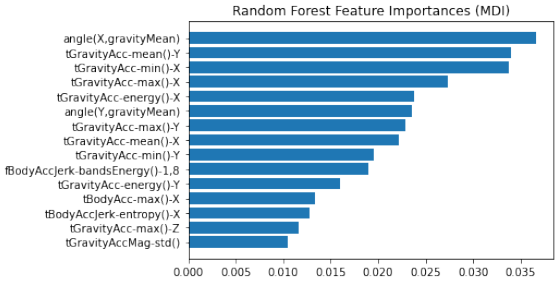
- *min samples leaf* fra 1 e 100 valori.

Come risultati migliori sono stati ottenuti: *n_estimator* = 100, *criterio*="gini", *max depth*=None, *min samples split*=2 e *min samples leaf*=1; lasciando invariati *min weight fraction leaf* a 0 e *max features* uguale ad "auto" si osserva come le performance di tutte le classi risultano ampiamente migliori rispetto al singolo DT (cfr. sezione 3.2).

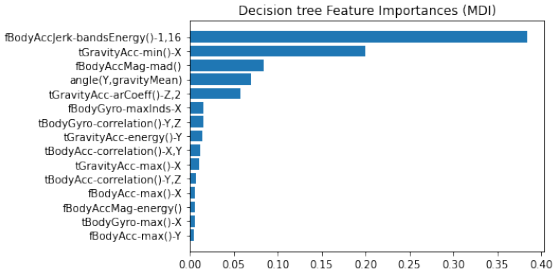
	Precision	**Recall**	**F1score**	**ROC auc**
1	0,89	0,98	0,93	1,00
2	0,90	0,90	0,90	0,99
3	0,96	0,84	0,90	0,99
4	0,90	0,90	0,90	1,00
5	0,90	0,91	0,91	1,00
6	1,00	1,00	1,00	1,00

Tabella 11: Valori del modello RF

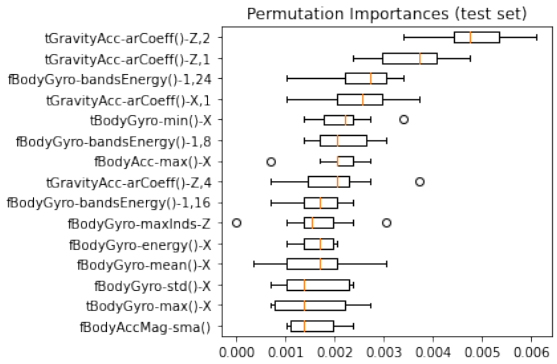
Le figure sottostanti (fig. 15) mostrano la Features Importance: tecnica che assegna un punteggio alle features in input, basandosi sull'importanza che hanno nel predire la variabile target. Comparando i risultati ottenuti per quel che concerne la features importance del Random Forest e quella del Decision Tree, è possibile notare come l'unica feature che abbia un'importanza elevata in entrambi i classificatori risulti essere *"tGravityAcc-min()-X"* (segnale di accelerazione del sensore circa le componenti gravitazionali e i movimento del corpo); sebbene nel Decision Tree quest'ultima abbia un'importanza di circa il 10% maggiore rispetto al Random Forest, in esso aumenta l'importanza di tutte le altre features che nel DTC risultano molto basse. L'unica eccezione si riscontra per la variabile con maggiore importanza nel DT (*tBodyAccjerk-bandsEnergy()1,16*), la quale nel RF risulta avere un'importanza notevolmente bassa.



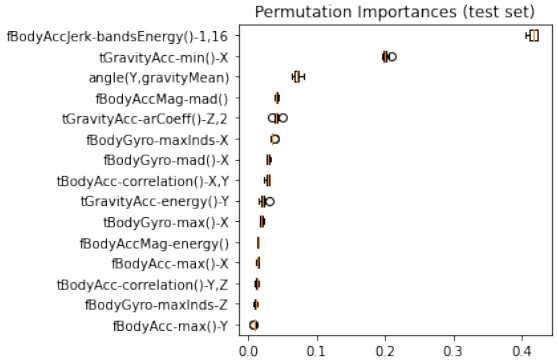
(a) Feature importance RF



(b) Feature importance DT



(a) Permutation importance RF



(b) Permutation importance DT

Figura 15: Features Importance

9.2 Bagging e Boosting

Tramite una GridSearch è stato trovato 100 come valore ottimale per n_estimator e modificando il *base estimator*, è stato possibile confrontare i risultati per osservare quanto questi siano accurati nella classificazione.

Come primo base estimator è stata implementata una *SVM lineare*, tramite la quale abbiamo ottenuto un'*accuracy* di 0.95. I risultati per *Precision*, *Recall* ed *F1-score* sono riportati nella tabella 12.

	Precision	**Recall**	**F1score**
1	0,95	0,98	0,96
2	0,94	0,96	0,95
3	0,99	0,92	0,95
4	0,94	0,89	0,92
5	0,91	0,95	0,93
6	1,00	1,00	1,00

Tabella 12: Valori del classificatore SVM lineare

Il secondo base estimator utilizzato è il *Random Forest*: questo ha ottenuto un'*accuracy* pari a 0.93. Si possono nuovamente osservare i risultati di *Precision*, *Recall* ed *F1-score* nella tabella sottostante.

	Precision	**Recall**	**F1score**
1	0,90	0,98	0,94
2	0,90	0,93	0,92
3	0,96	0,84	0,89
4	0,90	0,88	0,89
5	0,89	0,91	0,90
6	1,00	1,00	1,00

Tabella 13: Valori del classificatore RF

L'ultima analisi è stata eseguita attraverso l'implementazione del *Decision Tree* come stimatore, risultando come il peggiore dei tre classificatori utilizzati, seppur presentando un valore di accuracy elevato (0.90). Anche in questo caso sono stati riportati in forma tabulare i valori indicanti la qualità del modello.

	Precision	**Recall**	**F1score**
1	0,87	0,96	0,91
2	0,88	0,83	0,86
3	0,93	0,87	0,90
4	0,88	0,80	0,84
5	0,83	0,89	0,86
6	1,00	1,00	1,00

Tabella 14: Valori del classificatore DT

Come è possibile osservare dai vari risultati ottenuti ai fini della classificazione del dataset attraverso il *bagging*, il *base estimator* che restituisce le migliori performance è l'*SVM* di tipo lineare.

Per quanto riguarda il *Boosting*, l'algoritmo di base scelto è il *Random Forest*, in quanto si presta particolarmente bene ad essere utilizzato da AdaBoost. Gli iperparametri utilizzati per la foresta sono quelli individuati dalla GridSearch realizzata nella sezione precedente.
Nella tabella sottostante (tab. 15) sono riassunti i risultati ottenuti, i quali risultano essere in minima parte inferiori rispetto all'implementazione della sola *Random Forest* (si perde lo 0.01 di *accuracy*).

	Precision	**Recall**	**F1score**
1	0,8	0,96	0,92
2	0,90	0,90	0,90
3	0,95	0,85	0,90
4	0,92	0,88	0,90
5	0,89	0,93	0,91
6	1,00	1,00	1,00

Tabella 15: Valori di Boosting con classificatore RF

Successivamente, si è deciso di testare questo algoritmo sul dataset sbilanciato (cfr. sezione 5), ottenendo delle performance migliori rispetto ai base classifiers utilizzati in precedenza. Tuttavia, tali miglioramenti non risultano tanto rilevanti da essere riportati più nel dettaglio.

10 Reti neurali

Le *reti neurali* sono sistemi di calcolo con nodi interconnessi, che funzionano in modo molto simile ai neuroni del cervello umano. Utilizzando gli algoritmi, esse possono riconoscere modelli e correlazioni nascoste nei dati non strutturati, raggrupparli e classificarli e, con il tempo, apprendere e migliorare continuamente. Nello specifico caso del nostro studio è stato implementato il *percettore multistrato*, un modello di rete neurale artificiale che mappa insiemi di dati in ingresso in un insieme di dati in uscita appropriati.

10.1 Multilayer Perceptron Classifier (MLP)

In primo luogo, il *Multilayer Perceptron Classifier* (MLP) necessita del settaggio di una grande quantità di iperparametri. Quindi, per individuare la combinazione più adatta al nostro dataset, è stata implementa una GridSearch considerando diverse combinazioni di iperparametri. In particolare, per hidden_layer_size è stato utilizzato un solo hidden layer contenente un numero di neuroni compresi tra 2 e 6. Anche per il batch_size sono stati considerati diversi valori (6, 12, 24, 32, 64 e 128). I restanti iperparametri sono stati inseriti nella grid considerando tutte le varie possibilità disponibili, ovvero *activation*: ['tanh', 'relu', 'logistic'], *solver*: ['sgd', 'adam', 'lbfgs'] e *learning_rate*: ['constant','adaptive'].

Ai fini dell'indagine, i valori restituiti dalla GridSearch che danno risultati più alti (sia per gli score che per l'accuracy) si hanno per *activation function = 'tanh', learning rate = 'adaptive' e solver = 'adam'*; per quanto riguarda i valori di hidden layer e batch size, i Classification Report risultano essere uguali impostando gli hidden layer su 4 e 5 e variando il numero dei batch nel range di riferimento. Si ha una differenza minima per quanto riguarda i valori di *accuracy* soltanto in caso di batch pari a 12, dove risulta essere migliore un numero di layer pari a 4, al contrario con una dimensione del batch pari a 32 il numero di layer ideale risulta essere 5.

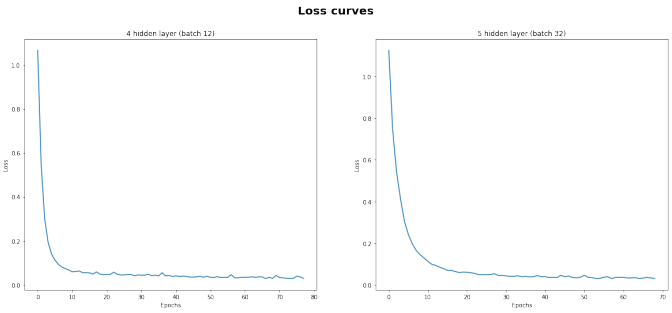


Figura 16: Variazione della loss al variare del numero di epochs

In figura (fig.16) sono riportate le variazioni della loss al crescere delle epochs; si è optato per mostrare graficamente i due differenti risultati in quanto per gli altri valori di layer e batch si hanno delle curve ancor più simili di quelle presenti. Nella figura è possibile osservare come la rete apprende molto velocemente, effettivamente dopo appena 10 epoche la curva raggiunge valori molto bassi.

10.2 Deep neural network

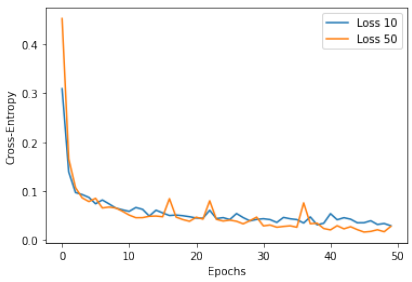


Figura 17: Livello di perdita durante le epochs

In questa fase dello studio in primo luogo sono stati creati due modelli, nei quali è stato mantenuto lo stesso numero di epochs (50), variando il valore di batch_size, ponendolo nel primo pari a 50 e nel secondo pari a 10. Nella figura 17 si può notare come la perdita durante la progettazione e la configurazione dei modelli risulti non mostrare grandi differenze tra i due. Difatti si calcola un livello di loss per il primo modello pari a 0.20, mentre per il secondo la loss è di 0.24.

Inoltre, anche l'*accuracy* risulta essere leggermente più elevata nel primo caso (0.95). Un paragone della performance dei modelli si può osservare dalla tabella sottostante (tab. 16), dove è evidente come il secondo modello performi in modo meno accurato. In particolar modo ciò si può osservare dai valori di *Precision, Recall e F1-score* della classe 5, tutti pari a 0.00. Ad ogni modo, i due modelli implementati, se paragonati ai precedenti, non mostrano dei risultati più interessanti ed accurati, per tal motivo non sono da preferire.

epochs=50, batch_size=10			**epochs=50, batch_size=50**		
Precision	Recall	F1-score	Precision	Recall	F1-score
1,00	0,68	0,81	1,00	0,52	0,68
1,00	0,15	0,26	0,97	0,06	0,12
1,00	0,69	0,82	1,00	0,50	0,67
1,00	0,05	0,09	1,00	0,01	0,01
1,00	0,00	0,00	0,00	0,00	0,00
1,00	0,96	0,98	1,00	0,78	0,78

Tabella 16: Risultati sui modelli al variare del batch_size

Un terzo modello è stato implementato ponendo *epochs = 700 e batch_size = 10*, ed in questo caso si ha una prestazione notevolmente migliore, i cui risultati sono riportati qui di seguito:

	Precision	**Recall**	**F1score**
1	0,99	0,93	0,96
2	0,96	0,94	0,95
3	0,99	0,84	0,97
4	0,97	0,81	0,89
5	1,00	1,00	1,00
6	1,00	1,00	1,00

Tabella 17: Valori del modello con epochs=700 e batch_size=10

Questo modello presenta inoltre un livello di *Accuracy* più rilevante rispetto ai due precedenti (0.956), tuttavia la loss è ampiamente più elevata, arrivando a 0.65. Quindi si migliora nella performance del modello al prezzo di una perdita maggiore (circa 0.4).

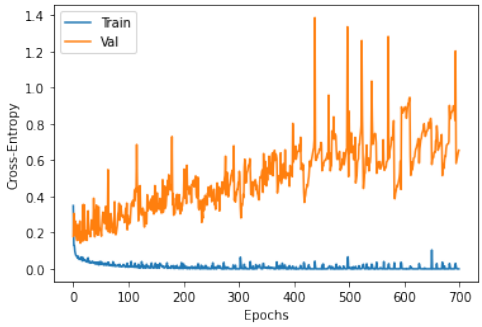


Figura 18: Overfitting della rete

A questo punto si è deciso di dividere il Training Set in Training e Validation set (80-20%) per capire se la rete stesse andando in overfitting. Come possiamo vedere dalla figura 18, dopo alcune iterazioni, l'errore del Validation Set inzia ad aumentare mentre l'errore del Training continua a diminuire.
Questo suggerisce che il modello sia in overfitting, per valicare il problema sono state utilizzate delle tecniche di *Early Stopping*.

Le tecniche utilizzate sono: *Dropout Regolarization*, *Layer Weight Regularization* e *NOREG Regularization*, ottenendo con quest'ultima i risultati migliori (Tab. 18).

	NOREG	DROPOUT	LAYERWEIGHT
Accuracy	0,98	0,83	0,83
Loss	0,03	2,57	0,45

Tabella 18: Valori di Accuracy e Loss per Regularization

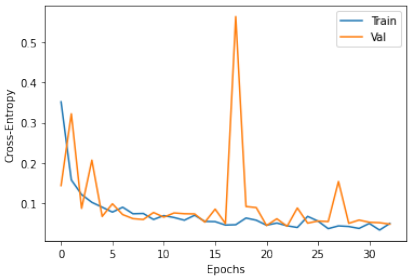


Figura 19: Overfitting dopo NOREG Regularization

Per ottenere tali risultati con la *NOREG Regularization* si è proceduto implementando un modello con *epochs=100 e batch_size=10*, inserendo *Early Stopping* con *patience=10*. In questo caso pur avendo un'*Accuracy* pari a 0.98, quindi ancor più elevata, si ha una loss di 0.03, ovvero la più bassa ottenuta fino a questo momento; con questo modello si ottiene quindi la performance migliore.

Andando nuovamente a controllare l'overfitting dopo l'applicazione dello stop con i risultati migliori, si può osservare come il modello continui a non essere eccezionalmente preciso, ma esso ottiene comunque un miglioramente significativo. Ciò è facilmente osservabile dal grafico della figura 19.

11 Regressione Lineare

La *Regressione Lineare* è una tecnica usata per studiare una serie di dati che consistono in una variabile dipendente ed una o più variabili indipendenti. L'obiettivo ultimo è stimare un'eventuale relazione funzionale esistente tra tali variabili.

11.1 Regressione con variabili continue

In questa fase sono stati analizzati due attributi continui con cui definire un problema di regressione lineare. Il modello è stato realizzato utilizzando come variabile indipendente *tBodyAcc-max()-X*, ovvero il segnale massimo registrato dall'accelerometro (soltanto su uno degli assi), ed il segnale minimo, *tBodyAcc-min()-X*, come variabile dipendente. Tramite la *Regressione Lineare* si ottiene una funzione avente coefficiente pari a -0.62 e un'intercetta di 0.28.

È stata analizzata la validità del modello di regressione attraverso *MAE*, *MSE* ed *R2*. Il primo rappresenta la differenza tra i valori originali e previsti estratti dalla media della differenza assoluta sul set di dati; il secondo la differenza tra i valori originali e previsti estratti dal quadrato della differenza media sul set di dati. Mentre R-quadrato rappresenta il coefficiente di adattamento dei valori rispetto ai valori originali (può variare da 0 a 1 e più alto è il valore, migliore è il modello). Nel nostro caso i valori ottenuti sono:

1. **R2: 0.889**;
2. **MSE: 0.014**;
3. **MAE: 0.075**;

Come controprova dei risultati, la figura 20 mostra il confronto tra i valori reali della variabile dipendete e quelli predetti dal modello: come si può vedere, la predizione risulta accurata nella maggior parte dei casi.

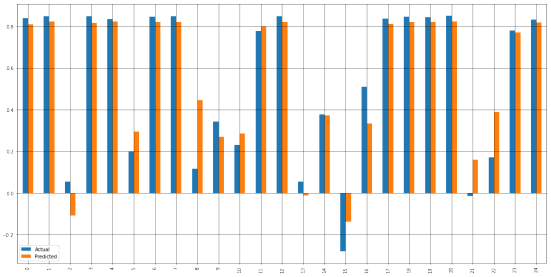


Figura 20: Confronto tra valori reali e predetti della DV

A riprova della buona performance del modello, il risultato in 2D della regressione (fig. 21) mostra come quest'ultimo sia adatto alla descrizione della distribuzione dei dati in quanto questi appaiono distribuiti linearmente.

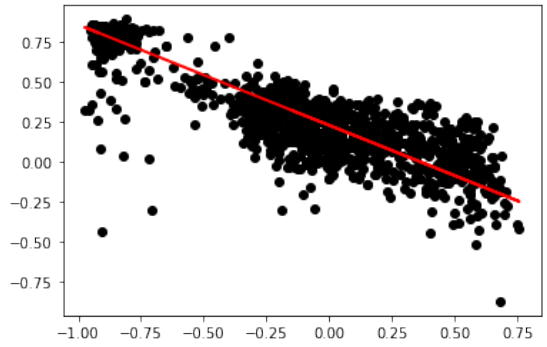


Figura 21: Visualizzazione in 2D della Regressione Lineare

11.1.1 Lasso e Ridge Regression

In aggiunta, sono state utilizzate la *Lasso Regression* e la *Ridge Regression*, per osservare eventuali cambiamenti nella performance. Se con la *Ridge* i risultati finali non cambiano, con la *Lasso* i risultati non hanno subìto alcun miglioramento, anzi peggiorano notevolmente.

11.2 Regressione Lineare Multipla

Infine, è stato implementato un modello di *Regressione Lineare Multipla*, con due variabili indipendenti (*tBodyAcc-max()-X* e *tBodyAcc-mean()-X*) e la precedente variabile dipendente *tBodyAcc-min()-X*. Come si può vedere, andando ad aumentare il numero di variabili utili per la predizione della variabile dipendente, i risultati migliorano:

1. **R2: 0.902**;
2. **MSE: 0.013**;
3. **MAE: 0.071**;

Tale miglioramento nella prestazione, seppur lieve, si può notare inoltre attraverso la visualizzazione del confronto tra valori reali e predetti della variabile dipendente (fig.22).

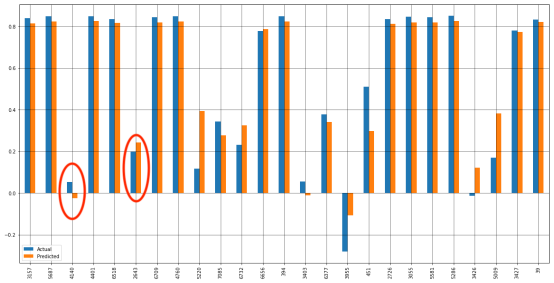


Figura 22: Confronto tra valori reali e predetti della DV nella MLR

Anche in questo caso, sia con la *Lasso* che con la *Ridge Regression*, si ottengono le medesime performance conseguite in precedenza.

12 Serie Temporali

Per l'analisi delle serie temporali è stato preso in considerazione il dataset *body_gyro_x* indicante il vettore di velocità angolare misurato dal giroscopio con una sliding window di 2.56

sec. Tale dataset è stato scelto tra i 9 possibili attraverso il calcolo della varianza e della media della deviazione standard di ognuno di essi; in tal modo è emerso quindi che *body_gyro_x* sia il dataset con tali valori più elevati, contando una STD di 0.41 ed una VAR di 7.78. Tale scelta è stata quindi dettata dal voler analizzare un tipo di serie temporale che fosse il meno omogenea possibile.

12.1 Motifs e anomalie

Per l'individuazione di motif è stata utilizzata la TS risultante dalle 7352 rilevazioni nel file sopracitato. L'analisi è stata effettuata sia sulla TS originale, sia su quella sottoposta a trasformazioni quali eliminazione del trend, della media e riduzione del noise attraverso il metodo smooting, tuttavia sono stati riportati solamente i risultati ottenuti sull'originale essendo più significativi. Nelle figure sottostanti sono stati analizzati i motif relativi a 2 differenti attività, ovvero *walking* per la figura 23 e *walking_downstairs* per la numero 24. Le anomalie riscontrate (fig. 25 e 26) permettono di individuare dei punti che si discostano maggiormente dai motifs dello spettro; esse emergono sia nel caso di *Walking* che in quello di *WalkingDownstairs* in corrispondenza dei medesimi time stamps. Inoltre, alcune anomalie vengono individuate in entrambe le TS verso la fine delle stesse.

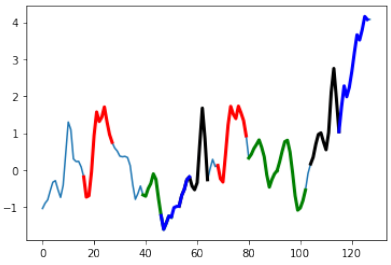


Figura 23: Rappresentazione dei motif per Walking

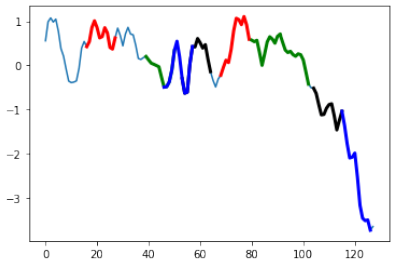


Figura 24: Rappresentazione dei motif per WalkingDownstairs

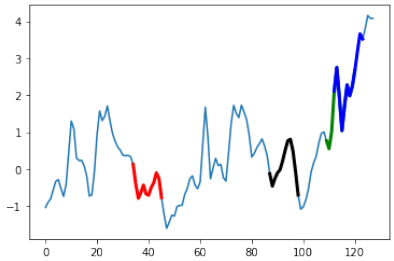


Figura 25: Rappresentazione delle anomalie per Walking

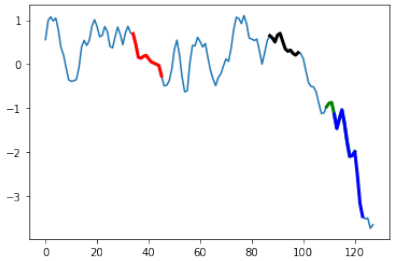


Figura 26: Rappresentazione delle anomalie per WalkingDownstairs

12.2 Shapelets

In una seconda fase di analisi, è stato nuovamente preso in considerazione il dataset *body_gyro_x*. Per l'implementazione si è optato per l'utilizzo di 6 shapelets di lunghezza pari a 12. Nelle figure sottostanti sono state prese in considerazione le attività sopracitate (*walking e walking_downstairs*); per *walking* (fig. 27) le shaplets tendono a sovrapporsi, sintomo di un andamento più irregolare rispetto a quelle ottenute, invece, con *walking_downstairs* (fig.28).

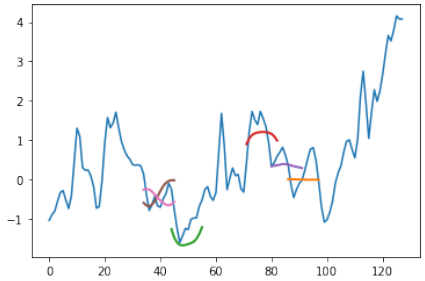


Figura 27: Rappresentazione delle shapelets per Walking

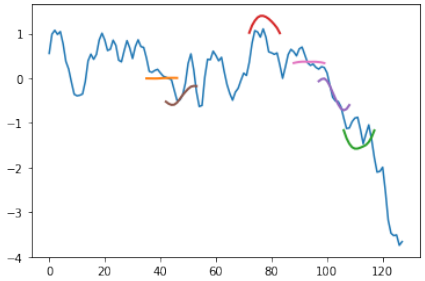


Figura 28: Rappresentazione delle shapelets per WalkingDownstairs

12.3 Clustering

Per la fase di clustering si è deciso di concentrarsi solamente sulle attività in movimento (*walking, walking_downstairs e walking_upstairs*), ottenendo quindi un dataset con un totale di 3285 time series. In particolare, i tipi di clustering su cui è stata effettuata l'analisi sono: *Shape Based, Feature Based, Compression Based* ed *Approximation Based*. I risultati riferiti al *Compression Based Clustering* non sono stati riportati nel report, in quanto poco significativi: il DBSCAN classifica tutti i punti all'interno di un unico cluster.

12.3.1 Shape Based Clustering

Nel caso dello *Shape Based* è stato ricercato il numero di K ottimale attraverso il Silouette Score e l'Inerzia, ottenendo come risultato 6 con la distanza Eucidea come metrica, mentre con DTW e softDTW il numero di cluster aumenta a 11; la tabella 19 riassume i risultati ottenuti.

Metrica	*Inerzia*	*Cluster*	*TS per Cluster*
Euclidean	31.77	6	(786, 536, 561, 291, 788, 323)
DTW	6.03	11	(389, 205, 181, 124, 550, 401, 76, 181, 597, 353, 228)
softDTW	34375.46	11	(485, 340, 161, 121, 259, 252, 279, 212, 478, 496, 202)

Tabella 19: Risultati per lo Shape Based Clustering

Qui di seguito sono riportate le figure inerenti alla distribuzione dei centroidi dello *Shape-based* con metrica Euclidean e DTW (si è optato per non riportare anche la softDTW in quanto presentava una rappresentazione simile a quella della fig. 30).

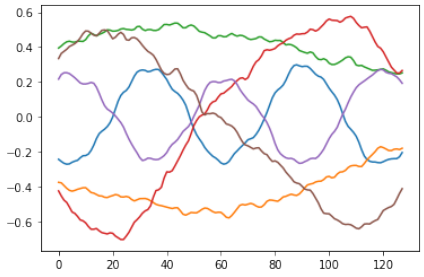


Figura 29: Rappresentazione dei cerntroidi con la distanza Euclidea

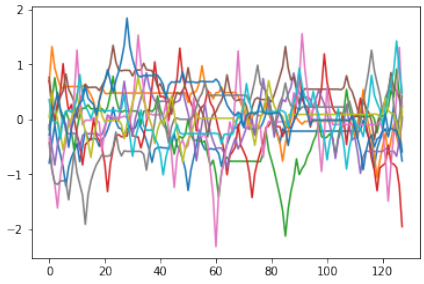


Figura 30: Rappresentazione dei centroidi con la DTW

Attraverso l'utilizzo della distanza Euclidea (fig. 29) i centroidi si distinguono in modo più evidente, è infatti possibile osservare facilmente trend ascendenti e discendenti.

D'altra parte, prendendo in considerazione i valori di Inerzia delle varie metriche, si può vedere come attraverso il DTW si ottengano dei cluster migliori in quanto più piccolo risulta il valore di Inerzia più coerenti sono i diversi cluster.

12.3.2 Feature Based Clustering

Per il calcolo del *Feature Based Clustering* sono stati utilizzati differenti indici statistici, tra cui: media, deviazione standard, Kurtosis, Skewness; calcolati tutti sul dataset senza normalizzazione. Anche in questo caso la scelta del numero dei cluster è avvenuta attraverso il calcolo del Silouette Score e dell'Inerzia, risultando nuovamente 6 come numero di K ideale. In tal modo si ottengono 6 cluster il cui contenuto non appare tanto bilanciato quanto quello conseguito con lo *Shape Based* con la distanza Euclidea, difatti quattro di questi in particolare tendono a sovrapporsi, causa di una media intracluster simile tra i diversi cluster. Pur aumentando il numero di K, tale risultato non cambia.

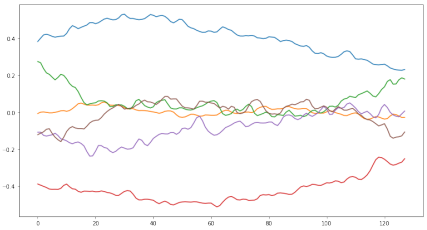


Figura 31: Rappresentazione dei centroidi per Feature Based clustering

12.3.3 Approximation Based Clustering

Il presente metodo di clustering è stato l'ultimo testato. Per fare ciò è stata utilizzata un'approssimazione PAA variando il numero di segmenti, e mantenendo k=6. Partendo con un numero di segmenti elevato (20), si ottengono dei centroidi simili tra loro; per ottenere una differenziazione tra questi è stato necessario diminuire tale valore a 5 (fig. 32).

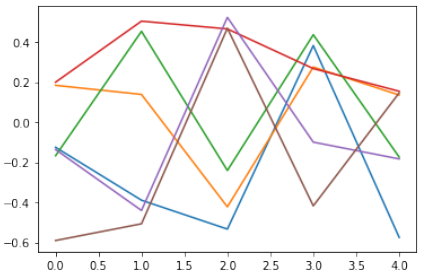


Figura 32: Rappresentazione dei centroidi per numero di segmenti=5

12.3.4 Conclusioni

A conclusione di questa parte di analisi lo *Shaped Based Clustering* risulta il metodo migliore in quanto nessuno degli altri analizzati presenta dei risultati simili. In particolare, le migliori metriche da implementare con tale algoritmo sono Euclidean e DTW: esse presentano un valore di Inerzia basso, sintomo di una buona differenziazione dei cluster. Come possiamo osservare dalla tabella 19, il valore di Inerzia minore è quello del DTW, risulta difatti la metrica da preferire per lo *Shape Based*.

12.4 Classification

In questa fase di analisi sono state utilizzate come variabili target le varie attività studiate anche in precedenza, in modo da osservare se i classificatori fossero in grado di classificare correttamente le azioni svolte. Per tale motivo è stato utilizzato lo stesso dataset delle fasi precedenti, il quale presenta già una suddivisione in train e test set.

Come modelli di classificazione sono stai utilizzati sia in Univariate che in Multivariate Time Series: *Shapelet models*, *Feature Based*, *TS classifier*, *CNN*, *LSTM*, *ROCKET*, *miniRocket*, *1NN DTW* e *CIF*. Di questi, *Feature Based*, *CNN* e *LSTM*, seppur testati, non hanno mostrato risultati interessanti, per questo di seguito verranno riportati solamente quelli con le performance migliori.

12.4.1 Shapelet Models

Per lo *Shapelet Distance Model* in primo luogo è stato trovato il numero ideale di shapelet_sizes, il quale risulta essere 12; inoltre sono stati ricercati il numero ideale di K e la *weight_option* attraverso una GridSearch, ottenendo nel primo caso 30 come valore ideale e nel secondo "distance". Lo *shapelet-based classifier* è stato applicato utilizzando sia il *KNN* sia il *Decision Tree*; nel primo caso i risultati ottenuti dal modello non risultano classificare al meglio le nostre classi, difatti il valore di *Accuracy* appare uguale a 0.56 ed i valori per *Precision, Recall ed F1Score* delle 6 classi non risultano mai maggiori di 0.67. Riguardo il DT, è stata implementata una GridSearch solamente per la profondità dell'albero, ottenendo 7 come valore più adatto; con il presente classificatore si sono potuti osservare dei risultati similari ai precedenti, con un lieve peggioramento delle performance, non a caso il valore dell'*Accuracy* è pari a 0.55.

12.4.2 Time Series Classifier

A seguito di una GridSearch per il KNN è stato impostato K=5 e weight="distance", mentre per il DT max_depth=15. Dalla tabella sottostante si possono osservare i risultati ottenuti:

	KNN			DT		
	Precision	Recall	F1	Precision	Recall	F1
1	0.92	0.71	0.80	0.70	0.72	0.71
2	0.68	0.61	0.64	0.55	0.52	0.54
3	0.91	0.48	0.63	0.54	0.51	0.52
4	0.47	0.67	0.56	0.48	0.62	0.54
5	0.49	0.52	0.50	0.48	0.50	0.49
6	0.43	0.53	0.48	0.48	0.37	0.42

Tabella 20: Performance dei modelli KNN e DT

Entrambi i modelli mostrano delle performance abbastanza buone, in particolare il KNN classifica meglio le classi rispetto al DT, a conferma di ciò esso presenta inoltre un valore di *Accuracy* più elevato (0.59).

In un secondo momento per il KNN si è deciso di impostare due diverse metriche di distanza: *dtw_sakoechiba* e *dtw_itakura*. La prima produce una banda centrata intorno alla diagonale principale e qualsiasi cella la cui distanza è inferiore rispetto a quella data, risulta valida. Diversamente l'itakura produce un parallelogramma: contrariamente alla banda sakoechiba, la cui larghezza è costante, il parallelogramma itakura ha una larghezza variabile, consentendo spostamenti di tempo più grandi nel centro rispetto ai punti iniziali e finali.

	sakoechiba			itakura		
	Precision	Recall	F1	Precision	Recall	F1
1	0.79	0.88	0.83	0.81	0.84	0.83
2	0.73	0.73	0.73	0.71	0.74	0.73
3	0.78	0.64	0.70	0.74	0.64	0.69
4	0.50	0.50	0.50	0.50	0.48	0.49
5	0.58	0.58	0.58	0.57	0.62	0.59
6	0.51	0.53	0.52	0.48	0.46	0.47

Tabella 21: Performance per sakoechiba e itakura

È possibile notare un visibile miglioramento nelle prestazioni del KNN, il quale, grazie a queste metriche implementate, classifica con più facilità le 6 differenti attività; in aggiunta anche il valore dell'accuratezza del modello risulta più elevato.

12.4.3 1-NN Classifier

L'*1-NN* è un classificatore KNN che utilizza 1 come valore di K; per la sua implementazione è stata impostata come metrica per la distanza DTW. Rispetto ai metodi utilizzati in precedenza, si ottengono delle performance migliori seppur non troppo rilevanti; le classi vengono classificate con un buon livello di accuratezza, difatti l'*Accuracy* equivale a 0.63.

	Precision	Recall	F1
1	0.81	0.83	0.82
2	0.71	0.74	0.73
3	0.74	0.65	0.69
4	0.51	0.51	0.51
5	0.55	0.58	0.56
6	0.48	0.47	0.48

Tabella 22: Performance dell'1-NN

12.4.4 ROCKET e MiniROCKET

A seguito di una prima fase di reshape si è proseguito implementando l'algoritmo *ROCKET* con RidgeClassifierCV come classificatore. Esso presenta delle prestazioni notevolmente più elevate rispetto ai modelli analizzati in precedenza; in particolare, come si può osservare dalla tabella 23, si ha un miglioramento generale nella classificazione di tutte le classi (nello specifico si hanno delle prestazioni accurate per la classe 1). Un ulteriore perfezionamento si ottiene con il *miniROCKET*, dove il livello di *Accuracy* è pari a 0.72; inoltre esso risulta computazionalmente più efficiente impiegando poco più di 30 secondi nell'esecuzione contro i quasi 5 minuti del ROCKET.

	ROCKET			miniROCKET		
	Precision	Recall	F1	Precision	Recall	F1
1	0.88	0.82	0.85	0.91	0.81	0.86
2	0.72	0.74	0.73	0.74	0.83	0.78
3	0.65	0.70	0.67	0.77	0.77	0.77
4	0.50	0.51	0.50	0.61	0.55	0.58
5	0.62	0.66	0.64	0.66	0.80	0.72
6	0.55	0.50	0.52	0.64	0.56	0.60

Tabella 23: Performance per ROCKET e miniROCKET

12.4.5 CIF - Canonical Interval Forest

Un differente modello di classificazione utilizzato è il *CIF*, la tabella sottostante ne riporta le prestazioni:

	Precision	Recall	F1
1	0.84	0.87	0.85
2	0.70	0.77	0.74
3	0.75	0.64	0.69
4	0.51	0.51	0.51
5	0.65	0.78	0.71
6	0.52	0.42	0.46

Tabella 24: Performance del CIF

Come è possibile osservare per *Precisoin*, *Recall* ed *F1-score*, i valori risultano simili a quelli ottenuti tramire il *ROCKET*, ma comunque meno precisi rispetto al *miniROCKET*; l'*Accuracy* in questo caso è pari a 0.66.

12.4.6 Multivaraite Time Series

In quest'ultima fase della nostra analisi si è proseguito attraverso l'implementazione di tre diversi algoritmi di classificazione multivariati. Per tale indagine, oltre al precedentemente citato dataset (*body_gyro_x*), ne è stato selezionato un secondo: *body_gyro_y*. Tale scelta è avvenuta in quanto si è preferito circorscivere l'analisi al solo giroscopio, per comprendere se avvenisse una buona classificazione delle misure calcolate dal dispositivo. Il primo algoritmo eseguito è il *ROC-*

KET, il quale rispetto all'univariate classifica in modo migliore tutte le classi, effettivamente l'*accuracy* aumenta di circa il 20%. Per quel che riguarda il *miniROCKET*, si hanno dei risultati ancora più interessanti: esso risulta, anche in questo caso, più accurato rispetto al *miniROCKET* univariato ed allo stesso *ROCKET* multivariato appena analizzato. Inoltre anche in questo caso il *miniROCKET* ha un costo computazionale inferiore (circa 2 minuti contro quasi 7).

	ROCKET			miniROCKET		
	Precision	Recall	F1	Precision	Recall	F1
1	0.98	0.75	0.85	0.95	0.78	0.86
2	0.86	0.92	0.89	0.83	0.99	0.90
3	0.77	0.85	0.81	0.85	0.85	0.85
4	0.60	0.64	0.62	0.69	0.71	0.70
5	0.67	0.75	0.71	0.76	0.88	0.81
6	0.66	0.60	0.63	0.76	0.62	0.68

Tabella 25: Performance per ROCKET e miniROCKET multivariate

L'ultimo algoritmo preso in considerazione è stato il *1-NN*, tramite il quale è stata ottenuta una performance migliore rispetto a quella con l'*1-NN* univariato, tuttavia risulta non essere il miglior classificatore; come mostrato dalla tabella sottostante (tab. 26) i risultati del classification report non riportano prestazioni più elevate rispetto al *ROCKET* o al *miniROCKET*.

	Precision	Recall	F1
1	0.83	0.86	0.84
2	0.80	0.83	0.82
3	0.89	0.66	0.76
4	0.57	0.66	0.61
5	0.68	0.62	0.65
6	0.52	0.57	0.54

Tabella 26: Performance dell'1-NN

Anche l'accuratezza del modello è inferiore rispetto agli altri due algoritmi, in questo caso raggiunge lo 0.70, mentre nei due precedenti essa è pari rispettivamente a 0.74 e 0.80.

12.4.7 Conclusioni

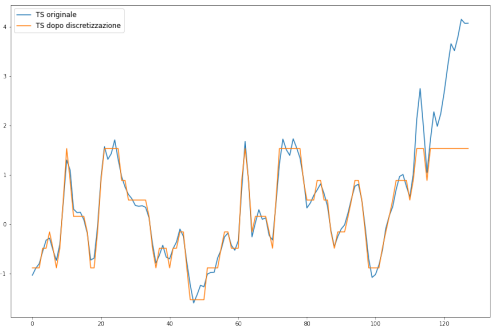
Concludendo, si può chiaramente affermare che non tutti gli algoritmi di classificazione implementati hanno portato agli stessi risultati. Sia nel caso di classificatori Univariate che Multivariate Time Series si hanno le performance più interessanti attraverso il *ROCKET* ed il *miniROCKET*, in particolare con quest'ultimo.

13 Sequential Pattern Mining e Advanced Clustering

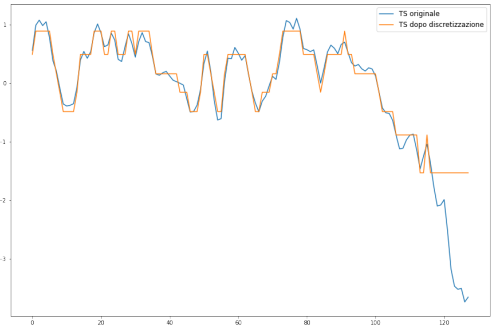
13.1 Sequential Pattern Mining

In questa sezione è stata effettuata un'analisi del *Sequential Pattern Mining* utilizzando il dataset precedentemente assunto nell'analisi delle Time Series (*body_gyro_x_train*).

Si è proceduto attraverso una riduzione delle dimensioni del dataset e, come in precedenza, si è optato per analizzare soltanto le classi relative alle attività in movimento (*walking, walking_downstairs e walking_upstair*s). É stata in seguito realizzata una discretizzazione del dataset utilizzando la SAX con n_paa_segments = 20 e n_sax_symbols = 8. La scelta di un numero ridotto di simboli (8) permette un'individuazione di un maggior numero di sequential pattern significativi.



(a) Rappresentazione di *walking* con e senza approssimazione



(b) Rappresentazione di *walking upstairs* con e senza approssimazione

Figura 33: Approssimazione SAX

Nella figura 33 possiamo vedere il confronto a livello grafico di due serie temporali con le relative approssimazioni.

Tramite l'algoritmo *PrefixSpan* è stato possibile estrarre i pattern più frequenti. Si è optato per utilizzare come support un valore pari al 15% della grandezza del dataset, ovvero 500 (il valore preciso sarebbe, in realtà, 492, ma si è preferito inserire l'approssimazione alle centinaia più vicine).
Andando ad osservare i support dei vari simboli all'interno del dataset possiamo notare come questi aumentino fino al raggiungimento del simbolo [4] e diminuiscano successivamente; inoltre considerando il valore di support di tale simbolo ed il numero totale di frequent pattern estratti (3473), è possibile affermare che questo appare in quasi tutte le estrazioni.

Support	Simbolo
639	[0]
1800	[1]
2812	[2]
3147	[3]
3162	[4]
2846	[5]
1828	[6]
579	[7]

Tabella 27: Support Singleton

In seguito, prendendo in considerazione le top 6 sequenze di lunghezza 5 ed i loro relativi support (tab. 28) è stato osservato in quante Time Series esse fossero presenti, ciò per comprendere se alcuni pattern fossero prevalenti in determinate rilevazioni del giroscopio.

Support	Sequenza
1624	[4, 4, 4, 4, 4]
1587	[3, 3, 3, 3, 3]
1496	[4, 4, 4, 4, 3]
1494	[4, 4, 4, 3, 3]
1472	[4, 4, 3, 4, 4]
1463	[4, 3, 3, 3, 3]

Tabella 28: Top 6 5-sequence

Dal grafico a barre (fig. 34) è possibile notare come la distribuzione dei vari pattern sia pressoché la medesima. L'unica classe che ha una distribuzione meno regolare rispetto alle altre risulta essere *walking*, ciò si può affermare in particolare andando ad osservare i pattern 3 e 5.

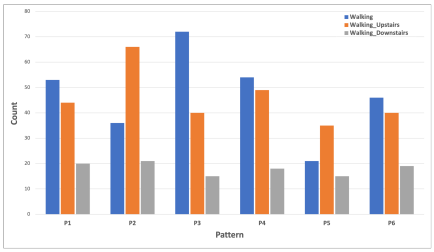


Figura 34: Distribuzione delle classi nei pattern

13.2 Advanced Clustering

Per la parte inerente all'Advanced Clustering è stato utilizzato nuovamente il dataset dei primi due moduli, sul quale sono stai testati diversi algoritmi, come: *K-Means, X-Means, OPTICS* ed altri due di tipo Transactional, ovvero *K-Mode e Rock*.

13.2.1 K-Means e X-Means

In questa fase è stato testato in primo luogo il metodo del *K-Means*; per fare ciò è stato ricercato il numero ideale di K attraverso l'implementazione dell'*SSE* e della *Silhouette Score* con valori di K in un range da 2 a 30. Nella figura 35 sono riportati gli andamenti delle due misurazioni di errore: la Silhouette presenta valori molto bassi e tende a decrescere all'aumentare di K. Il valore ottimale da impostare risulta essere K=3.

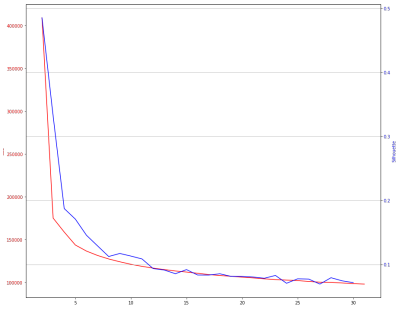


Figura 35: Andamento Silhouette Score e SSE

I tre cluster che si ottengono da tale implementazione presentano dimensioni differenti, il primo conta in totale 1288 elementi, il secondo 4044 ed il terzo 2020. Si è voluto quindi osservare la distribuzione delle differenti features all'interno di essi; ne risulta, come si può osservare dalla figura 36, che in quest'ultima appare evidente una sovrapposizione dei differenti elementi presenti all'interno dei cluster.

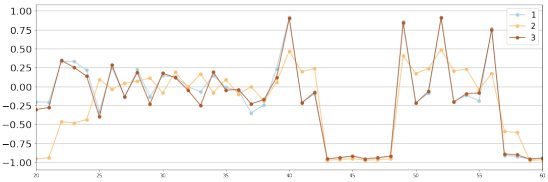


Figura 36: Distribuzione di 40 features nei 3 cluster

Dei risultati migliori si sono ricercati attraverso l'*X-Means*; in primo luogo è avvenuta l'implementazione del *WCE Score*, il quale indica che il numero ideale di cluster da prendere in considerazione equivale a 20 (fig. 37).

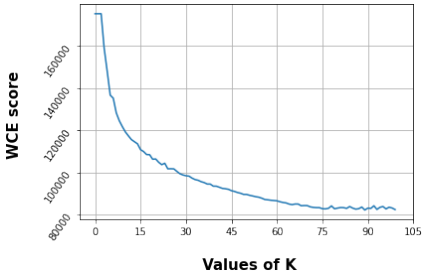


Figura 37: Andamento WCE Score

Anche in questo caso si ottengono dei risultati simili rispetto al modello *K-Means*, si è optato difatti per non riportare la visualizzazione grafica dei cluster ottenuti in entrambi i casi, in quanto la maggior parte degli elementi in essi risultavano sovrapposti tra loro.

13.2.2 OPTICS

Un ulteriore metodo di clustering avanzato implementato è *OPTICS* dove per trovare il numero ideale di clusters è stato utilizzato il grafico della *Reachability Distance* (fig. 38) dalla quale, come si può vedere, si ottengono due clusters. A riprova di tale risultato è stata realizzata una GridSearch facendo variare il numero di samples in un range da 5 a 20 e come risultato ottimale di cluster si ottiene anche in questo caso 2.

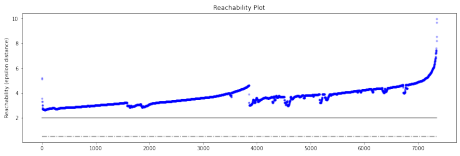


Figura 38: Reachability Distance OPTICS

Risulta quindi evidente che tale algoritmo di clustering non porta a risultati interessanti ai fini della nostra analisi, difatti la visualizzazione grafica ne mostra una sovrapposizione come quella ottenuta con gli algoritmi precedentemente analizzati, inoltre in questo caso vi è una non equa distribuzione degli elementi, dove il cluster 1 ha al suo interno gran parte di questi ultimi.

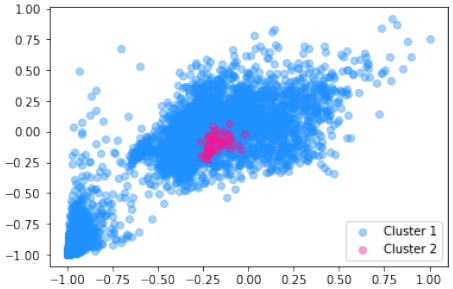


Figura 39: Visualizzazione dei cluster con OPTICS

13.2.3 K-Mode e Rock

Due algoritmi di clustering di tipo transazionale sono stati testati nell'analisi, i risultati ottenuti non sono interessanti ai fini dello studio in quanto *K-Mode e Rock* sono algoritmi che funzionano al meglio delle proprie prestazioni con attributi categorici e booleani, quindi non nel caso specifico del datset analizzato; seppur il primo presenta dei risultati più chiari rispetto al secondo. In primo luogo è stato implementato il *K-Mode*, per il quale, impostando come funzione di inizializzazione *Cao* e sfruttando una GridSearch, è stato ritrovato come numero ideale di cluster 5; è possibile osservare ciò attraverso l'andamento della funzione d'errore riportata in figura 40.

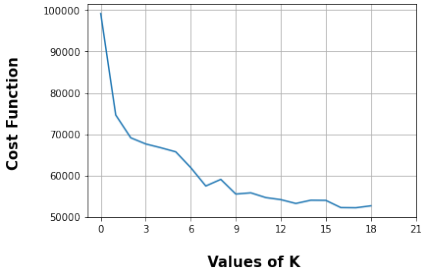


Figura 40: Andamento della funzione d'errore

Dalla visualizzazione in figura 41 si possono osservare 2 dei 5 cluster facilmente distinguibili in quanto ben separati rispetto agli altri che appaiono sovrapposti tra di loro. Difatti analizzando la distribuzione di alcune features nei centroidi (fig. 42) si ottiene un risultato simile a quello realizzato in precedenza con il *K-Means* (cfr. fig. 36), ovvero, ad eccezione di alcuni punti della distribuzione, vi è una chiara sovrapposizione tra i diversi elementi nei relativi cluster.

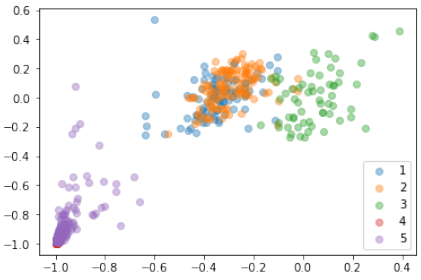


Figura 41: Visualizzazione dei cluster con K-Mode

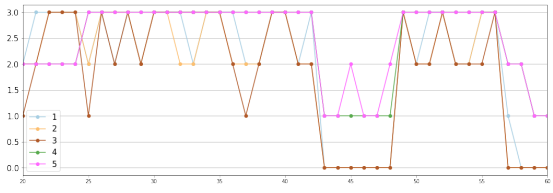


Figura 42: Distribuzione di 40 features nei 5 cluster

Per quanto concerne il *Rock*, anche in questo caso è stato utilizzato 5 come numero di cluster, ma gli esiti ottenuti non risultano interessanti ai fini dell'analisi, per tal motivo si è optato per non riportarli.

14 Explainability

Come ultima fase dello studio sono stati implementati due diversi explanation methods: *LIME e SHAP*, i cui risultati sono riportati qui di seguito.

14.1 SHAP

SHAP (SHapley Additive exPlanations) è un metodo basato sulla teoria dei giochi cooperativo che aiuta a predire l'importanza di ogni specifica feature in un dato modello.
Per la sua implementazione è stato scelto come modello di classificazione il *Random Forest*. Uno dei punti di forza del RF, e la ragione principale per la quale è stato scelto, è la sua capacità di calcolare varie misure di feature importance che possono costruire la base per migliorare la sua spiegabilità. Nelle figure seguenti è possibile osservare un confronto tra la feature importance calcolata da valori SHAP (fig. 44) e l'importanza delle caratteristiche calcolata dal classificatore sopracitato (fig. 43). Come si può vedere, hanno un aspetto molto simile, ma con alcune differenze nelle features, le quali inoltre nel caso dello SHAP hanno un impatto medio sull'output del modello più elevato.

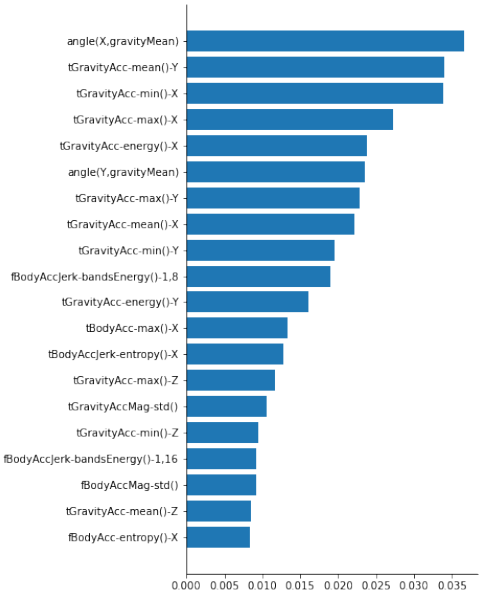


Figura 43: Feature importance del Random Forest

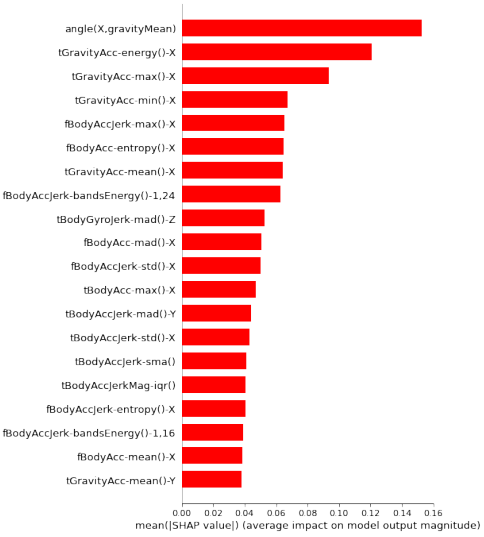


Figura 44: Feature importance dello SHAP

Il summary plot (fig. 45) combina l'importanza delle funzionalità con gli effetti delle stesse. Ogni punto del grafico di riepilogo è un valore Shapley di un'istanza per funzione.

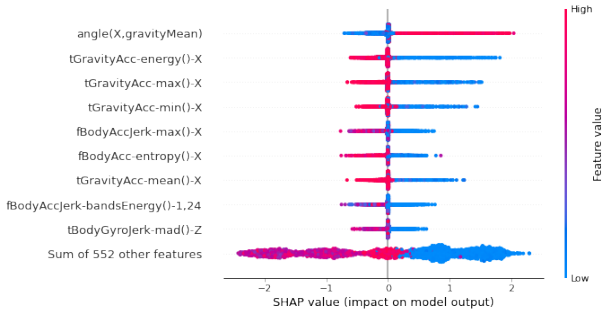


Figura 45: Summary Plot

Come è possibile osservare, le differenti features riportate in figura (quali *tGravityAcc-energy()-X, tGravityAcc-max()-X, tGravityAcc-min()-X*...) hanno un forte impatto nell'output del modello se negative, mentre *angle(X,gravityMean)* ha un impatto elevato se positiva.

Più nel dettaglio, qui di seguito sono stati riportati due waterfall plots indicanti nel primo caso un'attività statica (standing), mentre nel secondo una in movimento (walking).

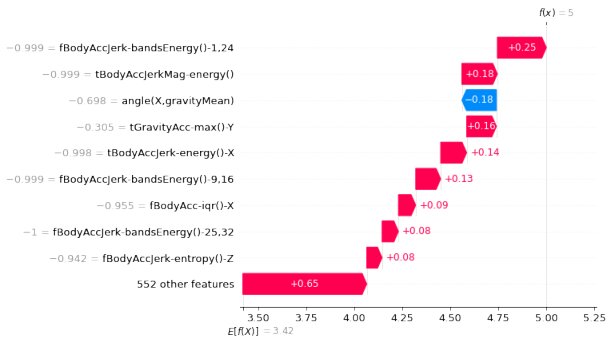


Figura 46: Waterfall Plot per standing

Se nella figura 46 la sola feature con un'influenza negativa nella predizione della classe risulta essere *angle(X,gravityMean)*, nella figura 47, invece, tale influenza è data da 5 features. Inoltre, se nel primo caso *fBodyAccjerk-bandsEnergy()-1,24* è la feature con un'influenza maggiore nella spiegazione della classe (+0.25), nel secondo tale ruolo è svolto dalla variabile *tGravityAcc-energy()-X*, la quale però ha un influsso di tipo negativo (−0.27).

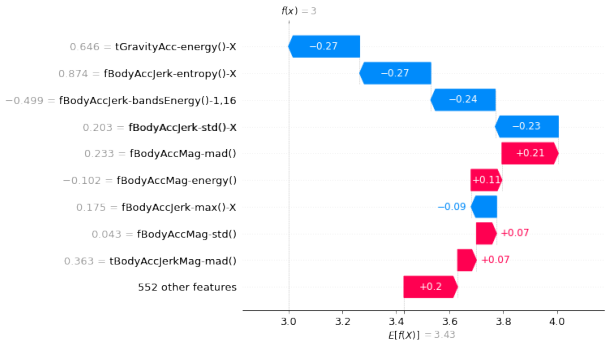


Figura 47: Waterfall Plot per walking

14.2 LIME

LIME (Local Interpretable Model-agnostic Explanations) è una nuova tecnica di explainability che spiega la previsione di qualsiasi classificatore in modo interpretabile e fedele apprendendo un modello interpretabile localmente. Anche in questo caso si è optato per utilizzare il Random Forest come classificatore, ma a differenza del modello utilizzato in precedenza, le classi rappresentanti le differenti azioni svolte sono state discretizzate in due classi indicanti attività statiche ed in movimento. Così facendo è stato possibile osservare le prediction probabilities di alcune righe del dataset; in particolare, qui di seguito, sono stati riportati i risultati ottenuti da due attività opposte per osservarne le varie influenze delle features. Seppur i risultati si mostrano come meno intuitivi rispetto ai precedenti, le prediction probabilities classficano ogni classe con un'accuratezza quasi sempre equivalente al 100%.

In un primo momento è stato implementato il modello predittivo prendendo in considerazione un'attività in movimento: wal-

king. Come si evince dalla figura 48 le features che influenzano positivamente la giusta predizione di tale attività risultano essere: *tBodyAccJerk-std()-X, tBodyAccJerk-max()-X e fBodyAccJerk-bandsEnergy()-1,24*, mentre variabili come *tGravityAccMag-arCoeff()4 e fBodyAccMag-min()* hanno un'influenza negativa nella predizione finale della classe.

Figura 48: Modello di predizione per walking

Prendendo in considerazione altri elementi riguardanti attività in movimento (come ad esempio walking_downstairs) si ottengono anche in questo caso dei risultati simili: emergono infatti come features positivamente influenti *tBodyAccJerk-std()-X e fBodyAccJerk-bandsEnergy()-1,24*, presenti nell'esempio precedentemente analizzato (cfr. fig. 48); ciò indica quindi che tali variabili condizionano, più in generale, le predizioni delle classi in movimento.

Per quel che riguarda le classi statiche, anch'esse vengono predette con un elevato livello di accuratezza, ma in questo caso le features influenti per la predizione riguardano principalmente differenti tipologie di accellerazione con valori sempre vicini a -1 (fig. 49).

Figura 49: Modello di predizione per standing

14.3 Conclusioni

In conclusione, confrontando i due metodi di Explainability implementati, entrambi mostrano dei risultati interessanti. Inoltre, più in particolare, in entrambi per la classe walking emergono come features influenti *fBodyAccJerk-max()-X* e *fBodyAccjerk-std()-X*.
Chiaramente, in base agli obiettivi che si desidera raggiungere con tale analisi, è da preferire *SHAP* se si aspira ad una spiegazione a livello globale, altrimenti la soluzione migliore risulta essere l'implementazione di *LIME*, difatti nello specifico caso di questo progetto il primo appare come la soluzione migliore.