HUMAN ACTIVITY RECOGNITION USING SMARTPHONES

DATA MINING II
PROJECT WORK

MARIA GRAZIA ANTICO 645005

ELEONORA BARACCO 616403

DAVIDE INNOCENTI 640090

UNIVERSITÀ DI PISA

2021/2022

Sommario

1. INTRODUZIONE

Il dataset da cui si sviluppa il nostro lavoro è nato da una collaborazione tra l'Università di Genova e il Politecnico di Barcellona. Attraverso l'ausilio dei sensori di movimento incorporati negli smartphone (Samsung Galaxy S II), nello specifico giroscopio e accelerometro, sono stati annotati i segnali di movimento di 30 volontari, i quali performano sei diverse attività. Da qui le sei classi di movimento che etichettano il dataset : WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING.

Il dataset è composto 561 features e per quanto riguarda la fase di pre-processing non molto è stato applicato, per diversi motivi.

Grazie alla documentazione dello stesso ci siamo innanzitutto accertati che i valori non avessero bisogno di alcuna normalizzazione in quanto già compresi nel range [-1,1], di conseguenza il dataset è stato nella maggior parte dei casi mantenuto tale, se non in determinate sezioni, verso cui si rimanda, dove è stata applicato uno StandardScaler, tecnica utile a genere una distribuzione con media 0 e deviazione standard 1.

Per quanto riguarda invece la qualità del dato ci si è accertati che non ci fossero incongruenze, nel dataset non sono infatti presenti missing values e le distribuzioni non mostrano elevate quantità di valori non inliers.

Diverse tecniche di Data Mining e ML sono state trattate e di conseguenza diversi approcci di volta in volta sono stati applicati sul dato per meglio preparare il dataset ad un determinato task, verrà quindi segnalato in ogni sezione dove e se un diverso pre-processing è stato applicato.

2. DIMENSIONALITY REDUCTION

Con riduzione delle dimensionalità si fa riferimento alla riduzione delle variabili in input di cui si compone un dataset (tecnica utile non solo per la data visualization) al fine di trasformare uno spazio multi-dimensionale in uno spazio a due o tre dimensioni, ma anche per l'implementazione di migliori modelli predittivi di classificazione. Per quanto riguarda il nostro dataset, vengono definiti dei task di classificazione che vengono applicati dapprima sul dataset non ridotto, composto quindi da 560 variabili, i cui risultati vengono successivamente confrontati con dimensionalità più contenute.

2.1 Classificazione base

I training di classificazione sono stati realizzati sia sul df_train_multi, con target variable avente valori che vanno da 1 a 6 e che rappresentano diversi dinamiche del movimento umano, sia sullo stesso dataset ma con una divisone binaria della suddetta variabile, df_train_binary, con valori inerenti alla staticità e al movimento dell'individuo, rispettivamente 0 ed 1. Una volta trainati i modelli, questi sono stati applicati su due Test set a loro volta multi-classe e binari, di cui vengono successivamente riportati i risultati. Due modelli supervisionati sono stati implementati: Decision Tree e KNN.

Infine, di ognuno di essi se ne riporta la combinazione di parametri selezionati, previa Randomized search, e risultati ottenuti su Test set.

2.1.1 Decision Tree - TEST_SET MULTICLASS:

Parametri ottimali:

{'criterion': 'entropy',
'max_depth': None,
'max_features': None,
'min_samples_leaf':6 ,'min_samples_split': 9,
'splitter': 'random'}

CLASSIFICATION REPORT DECISION TREE - TEST

	precision	recall	f1-score	support
1	0.70	0.83	0.76	496
2	0.73	0.69	0.71	471
3	0.91	0.76	0.83	420
4	0.83	0.78	0.80	491
5	0.81	0.85	0.83	532
6	1.00	1.00	1.00	537
accuracy			0.82	2947
macro avg	0.83	0.82	0.82	2947
weighted avg	0.83	0.82	0.83	2947

Figure 3. Classification report DT -Test

Grazie all'attributo feature_importances_ è stato possibile ottenere la lista ordinata delle variabili che l'albero di decisione riporta come più importanti. La sola *'tBodyAccJerk-sma()'*, ovvero la prima variabile trovata, viene riportata con una rilevanza del 40% rispetto alla totalità di features disponibili, ad implicare che quasi metà dell'informazione che il Decision Tree ottiene proviene da un'unica feature.

Il modello risponde in maniera ottimale al Test set riportando score elevati, non solo per quanto riguarda l'accuracy ma anche per metriche quali Precision, Recall ed F1-Score. Le classi con cui il modello fa, relativamente, più fatica sono la classe 2 e 4 (WALKING_UPSTAIRS) e 4 (SITTING), fattore che di certo non deriva da uno sbilanciamento delle classi, vista la perfetta omogeneità delle stesse. Le performances vengono confermate dalla ROC curve, di cui qui di seguito, dove la classe 6 (LAYING), colore arancione, prevale sulle altre venendo riconosciuta nella totalità dei casi. I risultati vengono infine confermati da una confusion matrix, di cui si riportano i dati normalizzati e da un Cross_Validation_Score dello 86,1%.

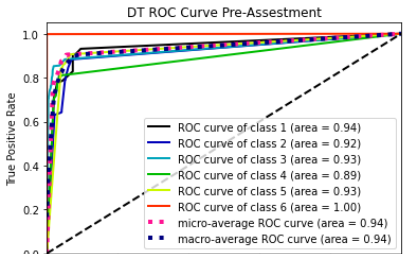


Figure 1. DT ROC Curve

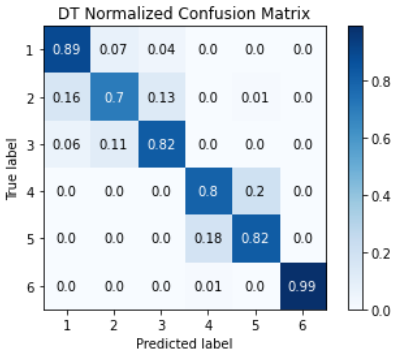


Figure 2. Confusion matrix DT normalized

2.1.2 Decision Tree - TEST_SET BINARY CLASS

Per realizzare questa classificazione la variabile target è stata mappata e trasformata in due soli valori, dove le classi 1,2, e 3 (movimento) vengono ora rappresentate da uno 1, mentre le rimanenti 4,5,6 (staticità) da uno 0.

Dopo aver impostato una Grid_search apposita ed aver avviato il modello, i risultati su Test set appaiono indiscutibili, raggiungendo picchi di 100% di precisione nel riconoscimento delle due classi: lo stesso si riflette sull' Accuracy e su tutte le altre metriche. Un Validation set è stato quindi generato dai records di traning per

testare nuovi scenari, senza però alcun risultato differente.

L'unico modo per ottenere risultati più utili al nostro scopo di comparare diversi modelli di classificazione è stato quello, come vedremo in una sezione successiva, di sbilanciare pesantemente il dataset. Per ora invece, conseguentemente ai risultati, si è deciso di continuare con il solo sviluppo di modelli multi-classe.

2.1.3 KNN - TEST_SET_MULTICLASS

Si è proceduto usando una Grid_Search di cui i parametri comprendono:

1. Il numero di vicini (**Nearest Neighbors**) **K:**
   - Un valore ridotto induce ad una suscettività del modello verso anomalie ed outliers.
   - Un valore elevato rende i "confini" della classificazione meno definiti.

Abbiamo quindi analizzato il numero di NN con un range di 30 valori partendo dal singolo più vicino.

2. Le **metriche di prossimità** usate per calcolare la distanza tra i punti. In generale il risultato non viene rivoluzionato al cambiare della metrica, e quella **Euclidea** è di certo quella più comune, ma abbiamo comunque voluto ampliare il discorso aggiungendo ai parametri la distanza di **Manhattan.**
3. Le **weight functions** utili per classificare punti nello spazio di cui non si conosce la classe utilizzando quella dei punti più vicini. Ciò viene realizzato basandosi su di una questione di maggioranza da parte dei NN, oppure "ponderando" il voto in base alla distanza dagli stessi NN. Le metriche scelte sono appunto "**Uniform**" e "**Distance**".
4. **Algoritmi** di base del KNN:
   - **Brute**: una soluzione che, come suggerisce il nome, utilizza una potenza computazione che mira a calcolare ogni singola distanza punto-punto del dataset. Risulta ingiustificata per classificazioni su dataset molto complessi, ma molto competitiva se ci si riferisce a dataset ridotti.
   - **K-Dimensional Tree**, che prova a ridurre la complessità di calcolo andando ad aggregare diversi punti (molto utile per spazi a poche dimensioni.
   - **Ball Tree** che, infine, ottiene risultati comparabili al KD_Tree ma su spazi a più dimensioni.

Saranno questi i parametri su cui verranno basati questo ed i successivi modelli di KNN.

Con poco stupore, anche in questo caso, un elevato valore di Accuracy viene raggiunto (metrica che prenderemo spesso in considerazione per comodità di confronto), sfiorando il **92%,** confermato da una Cross_validation del **91.8%.**

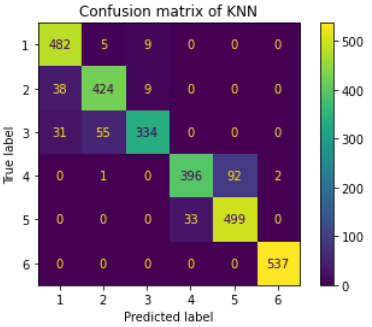


Figure 4. Confusion matrix KNN

3. FEATURE REDUCTION

In questo paragrafo vengono analizzate alcune delle più famose tecniche di riduzione delle dimensionalità. Bisogna innanzitutto distinguere tra due diversi approcci:

- **Feature selection:** le variabili che il modello genera come "ottimali" sono un sub-set di quelle originali; non viene modificata la struttura né il nome delle stesse.
- **Feature Projection:** tecnica per lo più utilizzata nella visual rapresentation, dove, generalmente, si passa da high-dimensional spaces a low-dimensional, creando un'enorme semplificazione della visualizzazione dei dati. Le features originali non vengono mantenute ma se ne creano di nuove, dove, in alcuni casi, è possibile ripristinare lo stato iniziale con una perdita minima e non sostanziale di informazione.
- **"Manuale",** per cui si cerca quelle sono le variabili più correlate tra di loro e senza l'ausilio di alcun particolare algoritmo si fa una cernita delle features che possono esser scartate per alleggerire il dataset.

Applicando in primis proprio quest'ultimo metodo è stata realizzata una Correlation_Matrix che indica la percentuale di relazione per ogni coppia di variabile secondo alcune metriche statistiche: Pearson è quella da noi utilizzata.

Per ovvi motivi di complessità (560 dimensioni) è impossibile riportare una rappresentazione grafica, ma basti pensare che due variabili

strettamente correlate (a partire quindi da un valore di circa 75%) contengono lo stesso valore informativo ed una di queste può esser scartata.

Nella nostra analisi diverse soglie sono state impostate e non c'è bisogno di arrivare a quote particolarmente basse per avere una sostanziale riduzione di variabili. Con un treshold del 95% (mantenendo ancora quindi molte variabili ad alta correlazione all'interno del dataset), la metà dello spazio dimensionale viene cancellato, rimanendo infatti con sole 258 features. Con una soglia del 75%, così come proposto in precedenza, si scende a sole 124, lasciando riflettere su quanto una feature reduction sia più che utile con un dataset del genere.

3.1 Variance Treshold

Approccio non supervisionato (non richiede in input la target variable), appartenente alla famiglia delle "**filter strategy**": in maniera molto simile al metodo appena presentato, non vengono usati particolari algoritmi e le variabili vengono scelte in base una soglia che viene impostata in input.

Nel caso del Variance Treshold vengono rimosse tutte quelle variabili la cui varianza non rispetta i livelli desiderati. Questa tecnica è molto utile per eleminare features particolarmente "piatte", da cui non è possibile ricavare grandi livelli di informazione e che quindi possono essere eliminate. Nel nostro caso, con un treshold pari a **0.19** otteniamo un ridimensionamento che ci porta a 67 dimensioni (quantitativo preso da esempio perché come si vedrà più avanti, la stessa PCA cattura il 95% di varianza proprio su 67 features). Applicando lo stesso Decision Tree di cui sopra le performance diminuiscono lievemente, raggiungendo il **79%** di **Accuratezza**. Diverse soglie sono state impostate, ottenendo risultati pressoché identici.

3.2 Univariate Feature Selection

L' Univariate Feature Selection è un approccio di tipo supervisionato e rimuove un quantitativo di features pari a quelle che non si vuole mantenere, il tutto basandosi su test statistici, nel nostro caso `CHI2`. Il parametro **K** (numero di features da mantenere) è stato provato in un range che va da 10 a 50, ottenendo risultati di classificazione ancor meno performanti del metodo precedente, rimanendo non oltre la soglia del 75% di accuratezza, motivo per cui si è pensato di scartare il suddetto per quanto riguarda il nostro dataset.

3.3 Recursive Feature Elimination

Probabilmente l'algoritmo migliore per la riduzione dimensionale per il nostro dataset. Appartiene alla famiglia "Wrapped", applica quindi algoritmi (input dell'utente), su cui man mano è applicata una selezione delle variabili.

Nello specifico, ricorsivamente, testa vari subset di features (nel nostro caso con un Decision Tree) e man mano seleziona quelle che sono le feature più importanti. È un greedy algorithm, ciò significa che non è deterministico. Grazie ad esso abbiamo raggiunto, applicato al Test set, un **Accuracy** pari all' **85%** sul **Decision Tree**, e del **91%** sul **KNN,** il tutto su un totale di **29** variabili selezionate. Di conseguenza si è pensato di estrapolare le features selezionate dall' **RFE** per lavorare su altri modelli di Machine Learning che beneficiano di tale riduzione. I dataset `df_RFE_train` e `df_RFE_test` sono stati creati e costituiscono la base dei successivi task del progetto.

3.4 PCA

La Principal Component Analysis è una tecnica che calcola una proiezione del dataset in uno spazio dimensionale fortemente ridotto, molto utile per la visualizzazione 2D-3D dei dati.

Due soluzioni sono state realizzate: una prima che implementasse il modello in due dimensioni, ed un'altra che rispettasse un determinato threshold di informazione. Nel primo caso (rappresentazione 2D) il risultato grafico è il seguente in Fig. 5. È stato deciso di implementare la PCA in due dimensioni, con il parametro n_components=2, ed un risultato ed un risultato di questo tipo qui accanto.

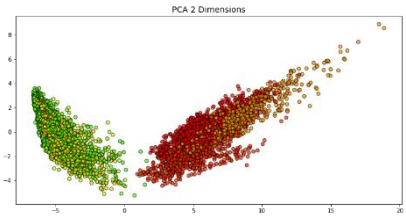


Figure 5. PCA 2-dimensions

Applicando però, i risultati, e quindi due sole features, al DT e KNN di base si ottengono risultati scarsi, rispettivamente 70% e 75 di Accuracy. Si è quindi pensato di sfruttare la PCA per catturare il maggior quantitativo di varianza utile, e di conseguenza sfruttare il numero di features da esso scelto. Per far ciò è stato plottato il seguente grafico che ci aiuta a comprendere come, in base al numero di

features, la varianza venga catturata dal modello:

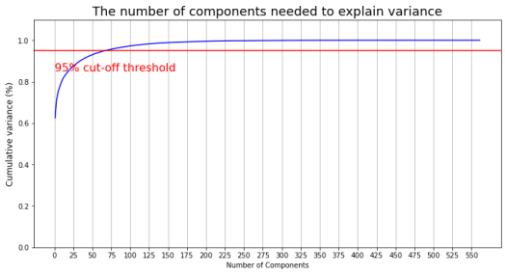


Figure 6. Components needed to explain variance

È stato selezionato un threshold del 95% di varianza, ed il numero di variabili che vengono selezionate per tale quantitativo è 67.

È stato poi creato un grafico a barre per visualizzare come la varianza si comportasse a partire dai vari components (Fig. 7).

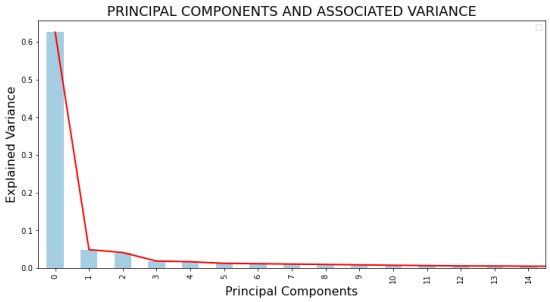


Figure 7. Principal Components and associated variance

Il numero di features visualizzate non comprende il totale (67), bensì solo una porzione di esse (top 15) poiché come è semplice vedere è la prima principal component a contenere oltre il 60% della varianza totale del dataset, mentre il discorso cambia radicalmente per le successive.

Anche in questo caso, la PCA con il 95% di varianza viene trainata sugli stessi modelli di DT e KNN, con risultati di poco inferiori ai modelli a 560 dimensioni, lasciandoci ancora una volta preferire l'RFE.

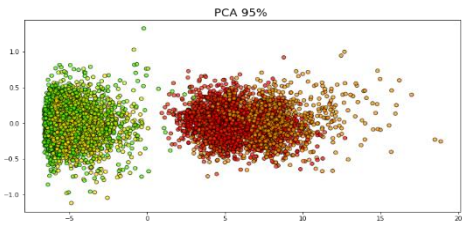


Figure 8. PCA 95%

3.5 T-SNE

Un ultimo modello è stato impementato, ovvero il *t- stochastic neighbor embedding*, che, come suggerisce il nome, basa le sue computazioni non sulla distanza ma sulle probabilità.

Fa parte dei Multi Dimensional Scaling, i quali si basano sull'assunto che due punti che sono vicini nello spazio multi-dimensionale lo saranno anche in quello ridotto, conservando di fatto la struttura del dataset. Partendo da questo concetto, grazie alla rappresentazione grafica del dataset in sole 2D (Fig. 10), ci si aspetta che la forma basilare dello stesso sia stata in qualche modo preservata e che questa rappresenti una situazione più o meno reale dei punti nello spazio. Riguardo alle performance si raggiungono score non molto soddisfacenti, sia su DT che KNN, sottolineando definitivamente l'RFE come soluzione migliore per il nostro lavoro.

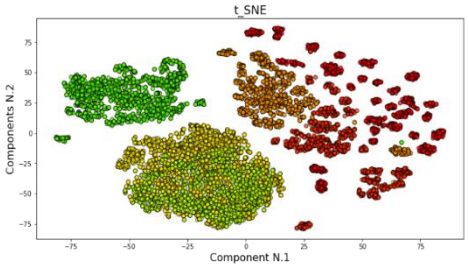


Figure 10. t-SNE 2D

4. UNBALANCING

Come precedentemente sottolineato, il nostro dataset è particolarmente omogeneo a livello di target variable, il che, in parte, fa sì che le performance di apprendimento sullo stesso portino a risultati spesso esagerati.

Per questo motivo è stato deciso di provare a sbilanciare un po' la situazione per vedere non solo come DT e KNN si comportano, ma anche per metter mano su alcune tecniche di Imbalanced Learning e testarne le funzionalità.

Si è operato quindi sulla nostra target variable, sia sotto forma binaria che multiclasse, sbilanciando in maniera decisa la quantità di classi all'interno del df_RFE_train. Non verrà qui riportata la digressione riguardo la class_lable binaria in quanto, nonostante lo sbilanciamento, non si è riusciti ad avere un punto di partenza abbastanza tale da poterne migliorare la situazione, essendo di per sé ottima sin dall'inizio. Di seguito la situazione pre e post sbilanciamento:

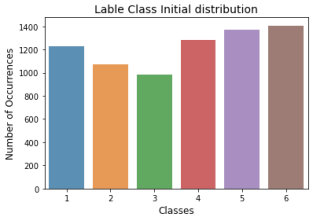


Figure 11. Labels Class initial distribution

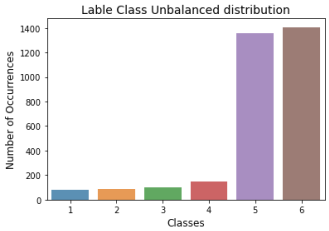


Figure 12. Lable class Unbalanced distribution

Come è possibile notare si è passati ad una situazione di questo tipo: `Counter({6: 1407, 5: 1374, 4: 1286, 1: 1226, 2: 1073, 3: 986})`

Ad una palesemente più distorta, ovvero:

```
Counter({6: 1407, 5: 1357, 4: 150, 3:
        100, 2: 90, 1: 80})
```

Per uno shape totale di: `(3184, 29)`

Il dataset è stato profondamente sbilanciato, con un rapporto di quasi 1:14 nel peggiore dei casi, a svantaggio non solo delle tre categorie riguardanti il movimento, ma anche una che si riferisse alla stazionarietà, al fine di mischiare meglio le carte in tavola.

Inoltre, si è pensato di non utilizzare alcuna tecnica "smart" nello sbilanciamento, sfruttando la libreria `imblearn` e settando manualmente il quantitativo desiderato, il tutto al fine di rendere tutto il più "sporco" possibile per gli algoritmi che invece andranno a ribilanciarlo a posteriori. A lato la situazione grafica del nostro dataset.

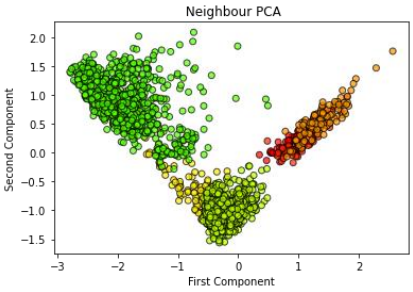


Figure 9. Neighbour PCA

Come da prassi un DT è stato implementato per avere un punto di partenza e studiare situazioni per migliorare la situazione. Sappiamo bene che il DT non è un algorimo deterministico bensì greedy, e dopo la solita grid_search, dopo alcuni tentativi, il risultato di accuracy si stabilizza intorno al 78% di Accuratezza con un F1-score sulla classe 4 non molto convincente, con 0.65.

Tre approcci di bilanciamento vengono proposti: *Class weight adjustment, Condensed Nearest Neighbor, SMOTE.* Nel primo caso abbiamo inserito un peso arbitrario alle classi della target feature, il tutto cercando di mantenere livelli di proporzioni adeguati. Classi fortemente sbilanciate (1,2,3,4) hanno un'importanza 10 volte superiore a quella delle rimanenti, così da bilanciare l'apprendimento del modello di classification. Buoni risultati sono stati ottenuti (82% di accuracy) ma ci si è basati sugli altri due metodi, più caratteristici della letteratura scientifica.

4.1 CondensedNearestNeighbour (CNN)

Questo primo metodo si riferisce a tecniche di Undersampling. Ciò che produce è un dataset più compatto dove la/le classi maggioritarie vengono riportate a livelli pressoché paritari a livello di proporzione con le classi minoritarie. Dopo diversi aggiustamenti sui parametri del CNN, si raggiunge la seguente situazione:

```
Counter ({1: 80, 3: 41, 5: 38, 2: 36, 6:
        31, 4: 25})
```

con uno shape di questo tipo: `(251, 29)`, andando a sottolineare come la classe 5 e 6 siano vertiginosamente calate di numero per pareggiarsi con le altre. A differenza di un Random Undersampling, questo algoritmo risulta molto più funzionale in quanto zone apparentemente più dense tendono a mantenere questa loro densità.

Questo grafico mostra come infatti diversi punti siano sparsi in maniera più o meno omogenea nello spazio, e come, sul lato sinistro, classi più dense, presumibilmente 5 e 6, mantengano tale livello di consistenza. Un DT è stato successivamente trainato ed applicato, ma vista la scarsità di record a disposizione raggiunge livelli non sufficienti di qualità, sfiorando a malapena il 70% di accuracy.

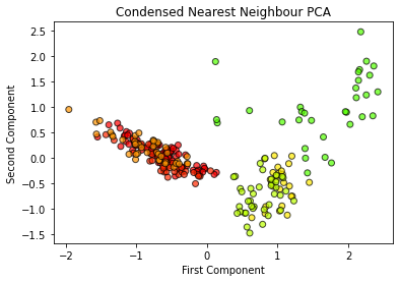


Figure 10. CNN PCA

4.2 SMOTE

Lo smote, diminutivo di Synthetic Minority Oversampling Technique, viene utilizzato per l'Oversampling su classe/i minotirarie. Quello che produce è quindi una situazione di proporzione tra classi alzando il numero di records appartenenti a classi scarsamente presenti.

La class lable viene quindi trasformata in:

```
Counter({1: 1407, 2: 1407, 3: 1407, 4:
        1407, 5: 1407, 6: 1407}
```

Con uno shape di tipo: `(8442, 29.`

Peculiarità dello SMOTE, a differenza di un Random Oversampling, è che non vengono ricreati a ripetizione punti nello spazio con le medesime caratteristiche, e quindi sovrapposti, ma si creano invece records fittizzi con attributi simili a quelli già esistenti, ma non uguali, il tutto nelle vicinanze più prossime di ogni classe.

Come si evince dal grafico, osservando ad esempio sul lato destro la classe di colore arancione, vengono creati punti che uniscono le distanze tra punti della stessa classe. Risulta facile osservare infatti come questa classe appaia ora più compatta. Tornando infine a parlare di performance, il DT che viene costruito sopra questo dataset ottiene risultati, qui di seguito, ottimali, raggiungendo un accuracy dell'87%,

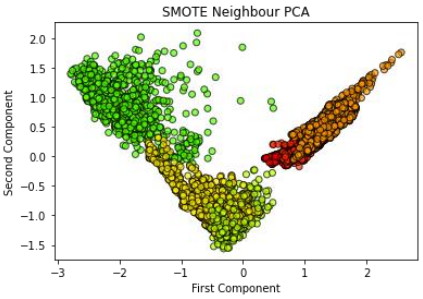


Figure 10. SMOTE PCA

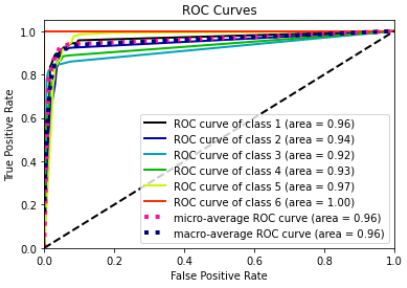


Figure 13. ROC curve DT

andando a migliorare sensibilmente la situazione (sbilanciata) di partenza.

Conclusioni

Per le sezioni che seguiranno del report, si terrà dunque considerazione del dataset ridotto ottenuto con RFE e composto da 29 variabili, da cui, come spiegato precedentemente, sono stati ricavati il set per il traning e per il test sui quali verterà il lavoro delle sezioni successive, ossia `df_RFE_train` composto da 7532 records e `df_RFE_test`, composto da 2947 records.

5. OUTLIER DETECTION

In tale sezione verranno applicati differenti approcci utili a rilevare l'eventuale presenza di outliers nel dataset. Inizialmente, attraverso un'analisi locale, è stato possibile evidenziare un'elevato numero di outliers in alcune variabili, come evidenziato dal boxplot sottostante.

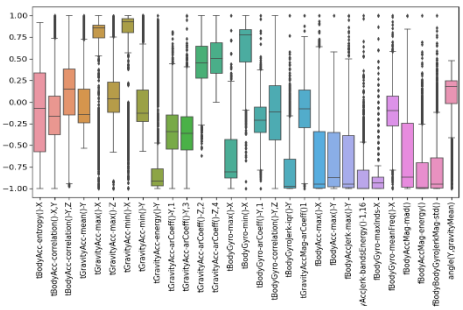


Figure 16. Boxplot Variables

Anziché procedere ad una eliminazione degli outliers attraverso la definizione del range interquartile, si è deciso, per ottenere un'analisi più completa, di analizzare ulteriori approcci per la rilevazione delle anomalie anche a livello globale. In particolare, sono stati selezionati alcuni approcci tra loro differenti (*Likelihood, Local outlier factor (LOF), ABOD* e *Isolation forest*) e, per alcuni di questi, si è scelto di osservare il dataset tramite PCA per meglio evidenziare, a livello visivo, la presenza di outliers.

Il primo approccio globale analizzato consiste nel *Likelihood approach* con una solida base matematica che risente del tipo di distribuzione delle variabili soprattutto quando il numero di queste è elevato. Utilizzando l'intero dataset (dunque 29 dimensioni), questo approccio si è rilevato poco utile nel nostro caso in quanto bisognerebbe applicare l'approccio per ogni singola coppia di variabili, e quindi risulta essere un procedimento più lento rispetto a quelli analizzati a breve, e pertanto è stato scartato dall'analisi in questione.

5.1 Model Based Approach

Uno degli approcci globali utilizzati è **l'Isolation Forest,** un modello che attraverso la generazione di più alberi isola le anomalie. Uno dei parametri più influenti di questo algoritmo è la contamination che definisce la percentuale di punti anomali del dataset. Al fine di individuare il miglior valore per questo parametro si è deciso di visualizzare il dataset tramite PCA, marcando i punti individuati come anomali. Così facendo si è compreso che la porzione di anomalie nel dataset è bassa, arrivando, dunque, a scegliere come valore ottimale di contamination 0,02 (2%).

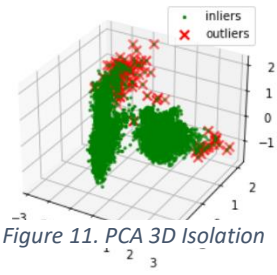


Figure 11. PCA 3D Isolation

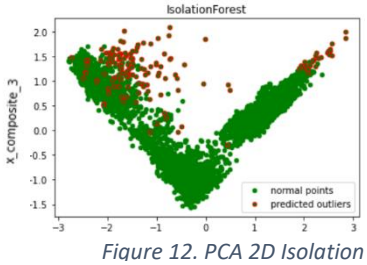


Figure 12. PCA 2D Isolation

Come si può notare dalle immagini non tutti i punti etichettati come outliers risultano effettivamente tali, inoltre alcuni di quelli definiti normali, invece, potrebbero essere

considerati anomali, soprattutto perché si trovano in posizione simile ad altri outliers. Aumentando il valore di contamination moltissimi punti anomali vengono individuati all'interno delle aree dense, mentre, diminuendo ancora il valore, un elevato numero di punti isolati non vengono selezionati.

Esiste un'estensione di questo algoritmo, la quale permette di effettuare "tagli" dello spazio non ortogonali. Si chiama **Extended Isolation Forest** e si è deciso di utilizzare l'algoritmo implementato dalla libreria H2O. Si è lavorato con alcuni parametri di default, ad esempio il numero di alberi generati (100) e il numero di sample casualmente selezionati per allenare ogni albero (256). Per quanto riguarda il grado di estensione, si può scegliere un valore che va da zero (isolation forest) fino ad il numero di feature meno 1 (estensione massima). Si è scelto di utilizzare l'estensione massima al fine di ridurre al minimo il bias dell'isolation forest. Questo algoritmo restituisce un punteggio di anomalia che va da 0 a 1, dove i valori prossimi a 1 sono da considerarsi anomali, tuttavia, se tutti i punteggi risultano vicini a 0.5, ciò significa che non ci sono vere e proprie anomalie. I risultati da noi ottenuti cadono esattamente in questo caso : i valori più bassi non scendono sotto lo 0,46 e i valori massimi non vanno sopra lo 0,65. Questo risultato è comprensibile, dato che anche con gli altri algoritmi la porzione di outliers presenti nel dataset è risultata bassa.

5.2 Density Based Approach

L'approccio tenuto conto in questa categoria è il LOF, ossia l'approcio che computa la deviazione locale di densità di un determinato punto rispetto ai suoi vicini. Utilizzando una soglia di contamination del 0.05%, è possibile osservare tramite PCA come tale valore sia sufficiente nel rilevare una porzione adatta di outliers. In questo modo, 202 outliers risultano essere rilevati. In particolare, il punto "meno outlier" risulta avere un valore di -1.41 ed il punto che risulta avere il valore più alto, e dunque considerato il "più outlier", ha un valore di -2.35. Normalmente, i punti considerati "inliers" hanno un negative outlier factor vicino a -1, dunque questi valori sembrano essere abbastanza contenuti.

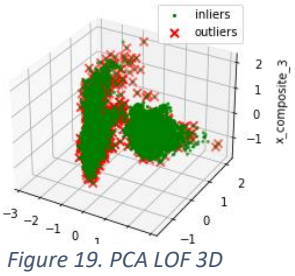


Figure 19. PCA LOF 3D

5.3 Angle-Based Approach

In questa sezione, invece, si è analizzato l'approccio ABOD per la rilevazione di outliers. Anche qui, grazie alla PCA, si è osservato che il valore che sembrerebbe ottimale si aggiri sullo 0.02% ed utilizzando k=5, con method=fast, rilevando in questo modo 145 outliers. Nonostante sembri che, con questi valori, alcuni puntini in aree meno dense e distanti da altri non siano rilevati come outliers, si è voluto appositamente non innalzare il valore di contamination in modo da poter eventualmente ricavare dei risultati interessanti da tali "anomalie".

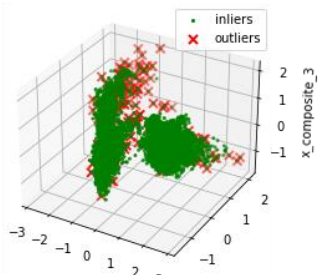


Figure 20. PCA ABOD 3D

Conclusioni

Dopo aver scartato l'approccio *Likelihood*, si è deciso di confrontare i risultati ottenuti da *LOF, ABOD e Isolation Forest.* Nel fare ciò, sono state stilate 3 nuove colonne contenenti i risultati degli algoritmi e si è deciso di rimuovere dal dataset gli outliers che risultavano essere comuni in tutti e tre gli approcci. Confrontando, dunque, le loro colonne si è ottenuto un totale di 31 outliers, i quali sono stati rimossi dal dataset seppur non ottenendo miglioramenti nella classificazione, in quanto considerato un numero pressoché irrilevante rispetto al numero totale di osservazioni e considerato, inoltre, come "rumore".

6.ADVANCED CLASSIFICATION

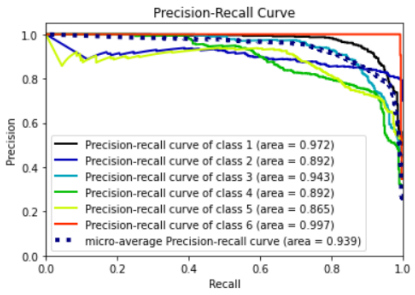
Per la sezione di classificazione, si è scelto di utilizzare, come dataset, uno pulito da outliers. In particolare, essendo gli outliers in comune rilevati nella sezione precedente non "rilevanti" al fine di miglioramenti per la classificazione base, si è dapprima utilizzato differenti dataset puliti dai diversi outliers riscontrati dagli

algoritmi in precedenza e si è riscontrato che, i migliori risultati, si ottenessero utilizzando il dataset ripulito dai soli outliers rilevati da LOF, e dunque ripulito dei 202 outliers.

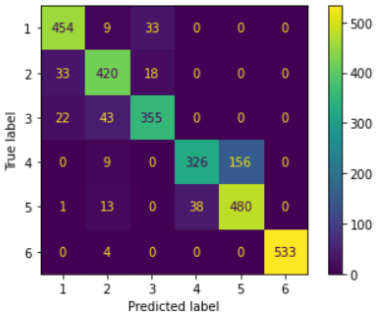
6.1 Naive Bayes

In questa sezione si analizzeranno differenti tipi di classificatori Naive Bayes, i quali si basano ognuno su differenti assunzioni. Il primo classificatore trattato è il **Gaussian Naive Bayes**, in cui vi è l'assunzione di base secondo cui i valori continui delle variabili predittive, associati ad ogni classe, siano distribuiti secondo una distribuzione Gaussiana. Si è ottenuta un'accuratezza del 89.9% nel train e del **87.5%** nel test, valori confermati anche dalla cross validation che riporta un 89%. È possibile notare come tutte le classi in media abbiano dei buoni valori di precision, recall ed automaticamente anche per f1-score, ad eccezione della classe 4 (ossia "Stare seduto") che riporta una recall del 58%, dunque vi è il rischio che vi siano più falsi positivi per tale classe (si veda il grafico qui a lato, in cui "Stare seduto" è evidenziato dalla linea verde).

	precision	recall	f1-score	support
1	0.87	0.97	0.92	496
2	0.87	0.92	0.89	471
3	0.96	0.80	0.87	420
4	0.95	0.58	0.72	491
5	0.72	0.96	0.82	532
6	1.00	1.00	1.00	537



Inoltre, per la classe 5 ("Stare in piedi"), è possibile notare una più bassa precision, in quanto, come mostrato nella seguente confusion matrix, per tale classe si riscontrano 156 (su un totale di 379) osservazioni non classificate correttamente. Inoltre, utilizzando l'attributo *.class_prior*, si osserva come, per ogni classe, la probabilità che le osservazioni appartengano a tali classi sono pressoché simili e non vi è nessuna classe con una probabilità abbastanza alta rispetto alle altre. Tra queste, la classe con maggiore probabilità, seppur di poco, risulta la classe 5 ("Stare in piedi").

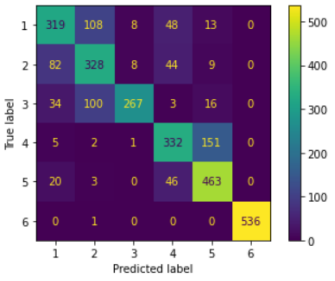



```
array([0.16796931, 0.15063237, 0.1382691
5, 0.1725167 , 0.18615887, 0.1844536 ])
```

Si è voluto, inoltre, prevedere le probabilità delle prime dieci osservazioni per ogni classe nel dataframe a lato. È possibile notare come, le prime cinque osservazioni, abbiano una probabilità del 99% e 100% e di appartenere alla classe 5("Stare in piedi").

	1	2	3	4	5	6
0	0.0	0.0	0.0	0.001	0.999	0.0
1	0.0	0.0	0.0	0.001	0.999	0.0
2	0.0	0.0	0.0	0.000	1.000	0.0
3	0.0	0.0	0.0	0.000	1.000	0.0
4	0.0	0.0	0.0	0.000	1.000	0.0

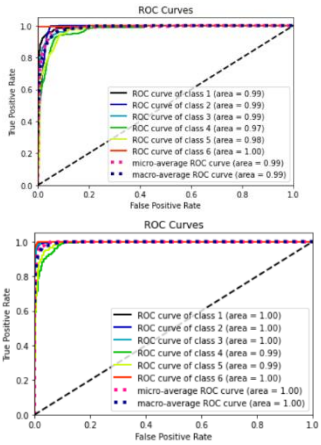
Successivamente, si è provato il classificatore **Bernoulli,** la quale assunzione di base è che le variabili predittive siano booleane. Il modulo BernoulliNB() permettere di binarizzare le variabili grazie al parametro "binarize", che si è scelto di impostare a 0, ossia alla metà del range (-1,1) delle variabili standardizzate. In questo modo, con risultati non inaspettati, abbiamo ottenuto un'accuratezza più bassa rispetto alla precedente, ossia del **76%** sul test e 80% sul train, in quanto non risulta ben adatto nel nostro caso binarizzare le variabili. Si osservano, infatti, moltissime osservazioni non classificate correttamente, questa volta con un numero elevato anche nelle classi 1("Camminare"), 2("Camminare avanti"), e 4("Stare seduto"), oltre che per la classe 5("Stare in piedi").

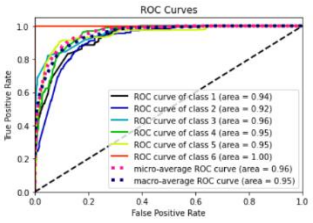


Infine, si è deciso di utilizzare anche il classificatore **Categorical**. In particolare, quest'ultimo assume una distribuzione categorica per ogni variabile predittiva. Per poterlo adattare al nostro caso, è stato necessario codificare ciascuna variabile in modo tale che ad ognuna fosse associato un numero intero ordinale. Per fare ciò, abbiamo discretizzato il nostro dataset con il modulo KBinsDiscretizer(), applicando come parametri n_bins=5, encode='ordinal' e strategy='quantile'. In questo modo, con 'quantile', i bins in ogni variabile avranno lo stesso numero di punti. In

questo modo, si è ottenuta una performance migliore rispetto ai primi due classificatori, con un'accuratezza del **91%** nel test e *90%* nel train e valori ottimali anche di precision e recall. Ance con la cross-validation si è ottenuto un 89%.

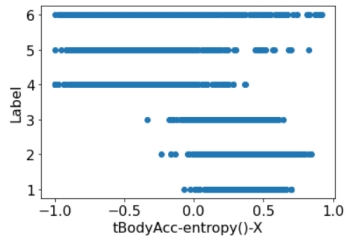
Si riportando, di seguito, le ROC curves di tutti e tre i classificatori, da cui è possibile dedurre che il primo e terzo risultano essere i migliori nel nostro caso, in particolare il terzo.



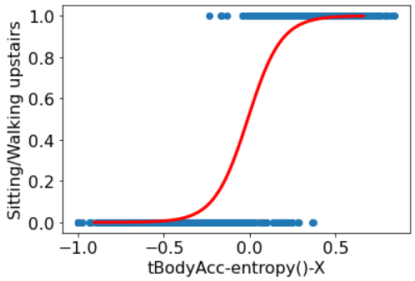


6.2 Logistic Regression

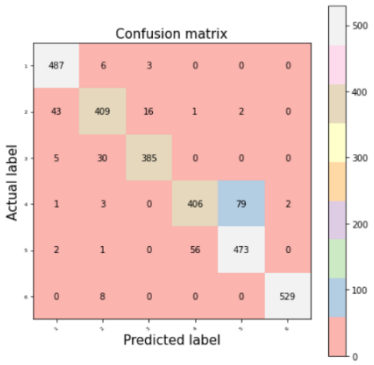
Per questa sezione si è voluto utilizzare la regressione logistica per la classificazione. Essendo che, a differenza della regressione semplice, con la regressione logistica è ottenuto un risultato e non un punteggio, si è voluto inizialmente applicarla soltanto in un contesto binario e con una sola variabile indipendente per meglio analizzarla a livello visivo. Successivamente, invece, per la classificazione multinomiale si è utilizzato l'intero dataset composto da tutte le 28 variabili. Per la classificazione binaria è stata scelta come unica variabile indipendente "entropia del segnale di accelerazione del corpo in direzione X" (ossia "tBodyAcc-entropy()-X"), in quanto rappresentata da una bassa multicollinearità. Com'è possibile notare dal grafico a lato, le classi



da cui è possibile ottenere una buona classificazione risultano essere varie, in quanto le ultime indicanti situazioni "stabili" risultano opposte a situazioni di "movimento" della persona e, per semplicità, abbiamo scelto di utilizzare in questa prima parte le classi 4 e 2, ossia le classi "Seduto" e "Camminare in avanti". Applicando, a livello grafico, la funzione sigmoidea di regressione logistica notiamo dunque come, evidentemente, con valori bassi di entropia del segnale di accelerazione del corpo la persona risulti essere seduta e come, al contrario, con l'aumentare del segnale la persona risulti camminare in avanti. Le performance, in questo caso, sono ottimali in quanto otteniamo un 98% di accuratezza nel test e con la cross validation il 100%.

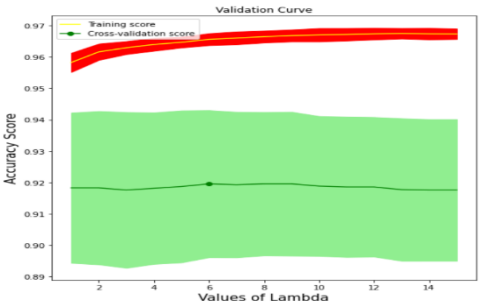


Analizzando, invece, tutte e 28 le variabili nel contesto multinomiale, tramite la gridsearch si sono trovati i seguenti iperparametri: `{'classifier__C': 6.0, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs'}`. Con questi valori, si sono ottenute performance ottimali, in particolare un 96.3% nel train e un **91.2%** nel test, valori confermati dalla cross validation che riporta un 96% di accuratezza. I valori ottenuti dalla confusion matrix sono quelli nella tabella a lato. In questo modo, grazie al penalty term Ridge, è stato possibile ridurre i valori dei coefficienti delle variabili in modo da ottenere risultati ottimali, soprattutto in vista della presenza di alcune variabili presenti nel dataset con un'alta multicollinearità (verrà approfondito in Sezione "Linear Regression"). Sono stati inoltre testati differenti valori più piccoli di C per confrontare i valori di performance ottenuti sul cross-validation, ottenendo i risultati riportati nella seguente tabella.



C	CV	Train
0.1	95.4%	95.7%
0.2	95.6%	95.9%
0.3	95.8%	96%
0.4	95.9%	96.3%
0.5	95.9%	96.3%
0.6	96%	96.3%
0.7	96%	96.3%
0.8	96%	96.3%

Per valori minori di 0.1, le performance risultano molto più basse, dallo 0.1 i valori crescono lentamente fino a raggiungere un lieve "picco" da poter notare anche nella *validation curve* a lato. Dunque, è possibile ottenere una buona classificazione con bene o male quasi tutti i valori di C ma, il valore ottimale, come ottenuto anche dalla gridsearch, risulta essere un valore C di 6.



6.3 SVM

In tale sezione sono stati mantenuti i soli records appartenenti alle classi 1 (WALKING) e 2 (WALKING UPSTAIRS) ed inoltre, nonostante i dati fossero già normalizzati su scala [-1,1], si è optato per una `StandardScaler()` che porta la distribuzione del dataset ad essere più adatta per la seguente task di classification. Una PCA è stata realizzata per avere un'idea bidimensionale del dataset.

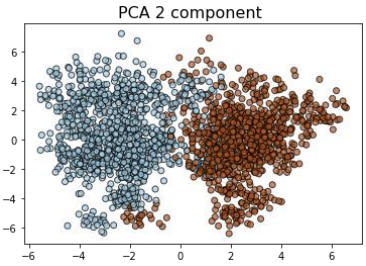


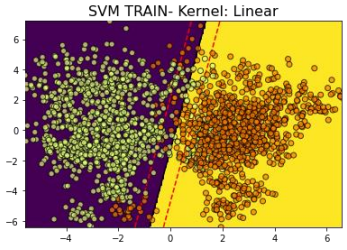
Figure 33. PCA 2 classes

Con SVM intendiamo un modello che seleziona il miglior decision boundary per separare i dati in due o più gruppi, al fine di ottenere zone dello spazio utili a classificare nuovi records. Per far ciò utilizza i così detti "Support Vector" che non sono altro che un subset del totale dei campioni utilizzati come boundary per sviluppare un hyperplane e dividere in maniera ottimale lo spazio. Diversi approcci sono stati utilizzati, ossia *LinearSVM e Non-Linear SVM*.

6.3.1 Linear Svm

Come il nome suggerisce, questa metodica è principalmente adatta a dataset dov'è possibile separare in maniera chiara i punti nello spazio da una linea retta. Ciò che l'algoritmo SVM cerca di fare è di trovare questa divisione netta cercando, allo stesso tempo, di massimizzare il margine presente dai support vector appartenenti ad ogni classe.

Prendendo ad esempio la PCA del data-preprocessing è possibile osservare come questo approccio non sia particolarmente adatto al nostro scopo. I decision boundary ed i rispettivi spazi mostrano come una divisione netta e lineare è impossibile per i nostri dati, a meno di non dare spazio a numerose missclassification. Nel nostro caso si è preferito usare la classe `SVC(kernel="linear")` anziché `LinearSVC()` per via del fatto che la seconda non ottimizza la la loss function basilare delle SVM, "Hinge", bensì una sua forma alterata. Ampliare i margini porterebbe ad un errore significativo, quindi, nonostante buoni risultati (picco di 91% di accuracy), escludiamo la soluzione lineare.



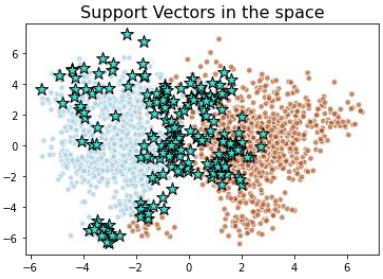
6.3.2 Non-Linear Svm

Per andare oltre la linearità abbiamo selezionato diverse tipologie di kernel: *Polynomial, RBF, Sigmoid.* Il concetto alla base di questi strumenti è di dare accesso a spazi con più dimensioni dove il problema da non lineare diventa invece lineare e quindi facilmente scomponibile come se si trattasse di un piano 2D ed una retta che lo attraversa. Oltre al kernel, si è analizzato anche l'effetto sui valori di "C", un parametro di regolarizzazione che modella i margini dell'iperpiano, e che è inversamente proporzionale alla generalizzazione del modello stesso: ad alti valori di C corrispondono dei margini più stretti, con conseguente minor numero di Support Vector, a valori più bassi i margini tendono a rilassarsi aumentando i SV ma commettendo qualche errore di classificazione in più. Dopo una Grid Search sui kernel e su altri parametri abbiamo ottenuto la

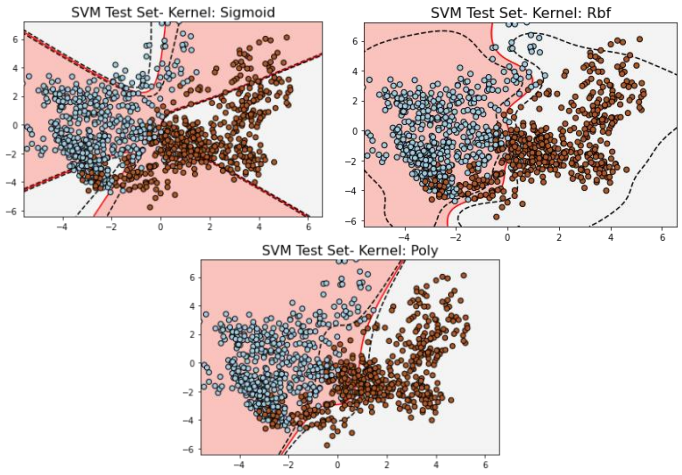
seguente configurazione (con C appartenente ad una collezione del tipo [0.001, 0.01, 0.1, 1, 10, 100]):

```
{'C': 0.1,
'class_weight': None,
'gamma': 'auto',
'kernel': 'rbf',}
```

Per ottenere alti scores c'è stato un trade-off tra ampiezza di margine e numero di Support Vectors in quanto con la configurazione di cui sopra vengono trovati circa 350 SV, con una rappresentazione grafica come segue, da cui si evince un pattern non facilmente delineabile per via della mappatura dello spazio in ulteriori n-dimensioni da parte del kernel.

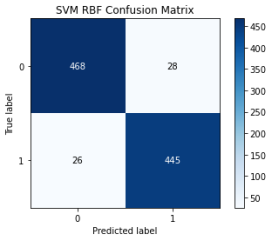


Valori più alti di regolarizzazione avrebbero portato ad un quantitativo minore di SV, ma con conseguente abbassamento di score accuracy ed F1. Di seguito i risultati grafici e numerici su Test set delle varie configurazioni:



	Accuracy	F1-Score	CV
RBF	0.944	0.942	0.945
Sigmoid	0.933	0.926	0.935
Linear	0.913	0.903	0.922
Poly	0.848	0.821	0.921

Così com'è possibile notare, il kernel Polinomiale ottiene risultati peggiori, non solo a livello numerico ma anche visivamente: i margini appaiono deboli e le classi non vengono distinte bene come negli altri casi. Secondo, di poco, il kernel Sigmoid, non molto sensibile all'aumentare della regolarizzazione dei margini che in altri casi avrebbe probabilmente prevaricato sull'RBF. Infine, appunto, il Radial Basis Function che ottiene i migliori score e di cui confermiamo la qualità con la seguente Confusion Matrix.



6.4 Single Perceptron

Per questo classificatore è stato possibile effettuare una GridSearch che comprendesse un ampio spettro di parametri, quali alpha, max_iter, tol (tolleranza) ed eta0, su di un altrettando ampio range di valori, ottenendo buoni risultati:

```
Accuracy 0.8985

F1-score 0.8988
```

Particolarità della rete neurale con singolo perceptron è che, a differenza della regressione logistica, questi non produce una probabilità d'appartenenza, bensì associa ogni record ad una classe, motivo per cui è spesso preferibile la seconda. È stato inoltre dimostrato come faccia fatica con problemi non lineari, per cui non rappresenta una soluzione ottimale per il nostro scopo.

6.4.1 Multi Layer Perceptron

Questo modello è stato implementato con l'ausilio di due differenti librerie al fine di avere una visione d'insieme più completa sulle reti neurali complesse.

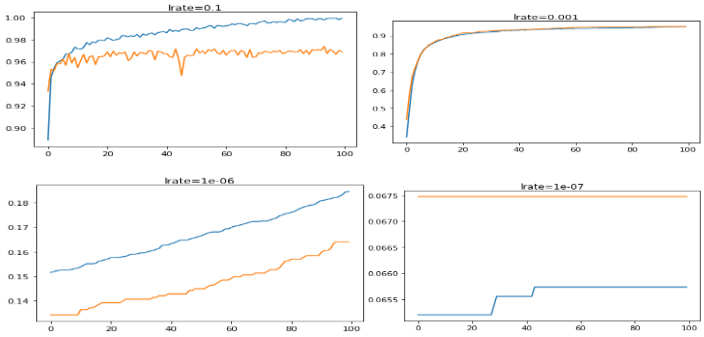
Scikit-Learn

Si è dapprima utilizzata la classe MLPClassifier() tramite Scikit-Learn la quale, con estrema facilità, ci permette di implementare MLP diverse senza particolari problemi. Diverse RandomizedSearchCV() sono state implementate, dimostrando come nel nostro caso 3 parametri rappresentino una soluzione ottimale:

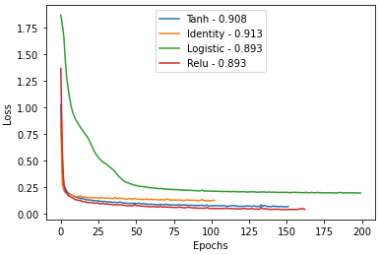
```
{'activation':'identity',

'momentum': .9,

'solver': 'adam'}
```

Con particolare attenzione verso quest'ultimo poiché riconosciuto a priori come uno dei migliori solver per dataset di grandi dimensioni. Il

numero di massime iterazioni è stato impostato a 100, così come le diminsioni di hidden-layer a (128,64,32) di conseguenza a ricerche fatte in letteratura che stabiliscono, nella maggior parte dei casi, buoni risultati con queste configurazioni negli strati nascosti. Si voluto poi rappresentare l'andamento delle performance di apprendimento su quest'implementazione al variare del Learning Rate, parametro utilizzato nell'aggiornamento dei pesi al variare delle epochs.



Come è possibile osservare troviamo uno "sweet spot" in relazione a livelli di accuratezza (asse verticale) pari a 0.001. Buona pratica è quella di mantenere livelli di LR contenuti in quanto valori troppo alti porteranno a performance discordanti tra Train error e Test error, così come quando impostato a 0.1 nel nostro caso; valori più bassi porteranno non solo a discontinuità ma anche al raggiungimento di soglie non molto elevate di scores. Fig.40 mostra i risultati ottenuti variando le diverse activation function (nel nodo di output), con un picco di performance di 0.918 di accuratezza da parte della funzione "Identity".

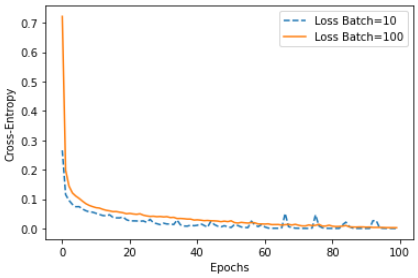


KERAS

Questa libreria, a differenza della precedente, permette di realizzare una rete neurale passo passo, potendo modificare la conformazione di ogni strato e persino l'activation function su ognuno di essi. Per questa modellazione abbiamo diviso il Training Set in Train e Validation, al fine di aver modo non solo di trainare il modello ma di poterne anche modificare al meglio gli iper-parametri. Il modello da noi creato prende forma in questo modo a lato ed è costituito da 3 strati nascosti con rispettivamente 256,128 e 64 neuroni, e funzioni di attivazione "Relu", il tutto sfruttando un modello sequenziale.

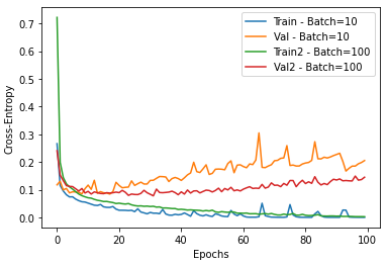
```
Model: "sequential_143"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_516 (Dense)           (None, 256)               7680

 dense_517 (Dense)           (None, 128)               32896

 dense_518 (Dense)           (None, 64)                8256

 dense_519 (Dense)           (None, 7)                 455

=================================================================
Total params: 49,287
Trainable params: 49,287
Non-trainable params: 0
_________________________________________________________________
```

Come Loss Function si è dovuta utilizzare "sparse_categorical_crossentropy" in quanto l'unica che calzasse con il nostro classification task e come metrica "accuracy". Abbiamo sfruttato un numero contenuto di epochs, al fine di ovviare (per quanto possibile) al problema dell'overfitting, impostando tale parametro a 100, in quanto con un centinaio di iterazioni vediamo che effettivamente non si raggiungono valori esagerati di accuratezza su train set. Con con l'ausilio del nostro validation set abbiamo testato la differenza tra un Batch=10 ed un Batch=100, riportandone di seguito i risultati in relazione all'aumentare delle iterazioni di Feed/Back-propagation e Cross-Entropy. I risultati, nonostante appaiano pressoché identici nascondono un vantaggio a favore di un Batch maggiore che ottiene:

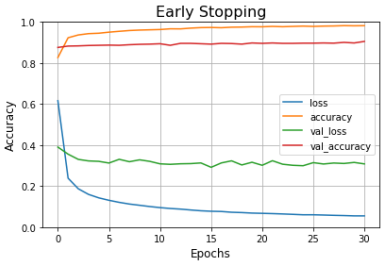


	Batch = 10	Batch = 100
Accuracy	0.892	0.901
Loss	1.278	0.593

Nel seguente schema, che mette in relazione le performance del Train set e Validation set in entrambe le combinazioni di livello di Batch, confermiamo non solo quanto detto pocanzi ma osserviamo anche come in entrambi i casi un numero di epochs intorno alle 20 sembra essere il punto di svolta per le funzioni di loss per quanto riguarda il validation set, rappresentando di fatto un



probabile threshold tale per cui il modello inizia ad andare in overfitting. Per andare più a fondo sulla questione si è testato un approccio di Early Stopping sfruttando l'omonimo modulo concentrato sui valori di "val_loss". Così come mostra il grafico sono, nonostante un valore di "Patient" impostato a 20 iterazioni, sono sufficienti non oltre le 30 epochs per convergere ad un'accuratezza/loss ottimale per il validation set.



Due ulteriori approcci di regolarizzazione sono stati infine sfruttati: "L2 Regularization" e "Drop-out". Il primo di questi interagisce con i pesi durante le iterazioni, aggiungendo un termine alla funzione utilizzata dall'algoritmo durante l'addestramento. In ogni strato della rete è stato inserito un parametro di regolarizzazione di 0.01. Il secondo invece interagisce con i neuroni presenti in ogni layer, compreso quello di input, i quali hanno una probabilità di non esser presi in considerazione. Dopo ogni layer è stato impostatato un valore di DropOut pari a 0.2. Entrambi gli approccio sono molto famosi in letteratura e spesso apportano miglioramento alle performance dei modelli. Di seguito i risultati:

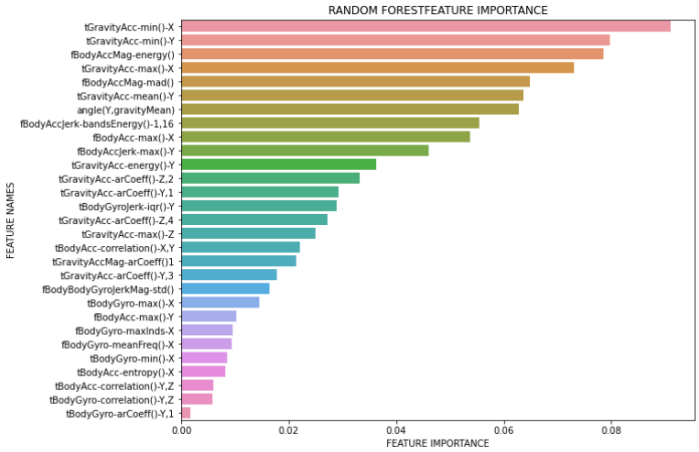
	E. Stop.	L2 Reg.	Drop Out
Accuracy	0.9162	0.6751	0.9019
Loss	0.2704	1.9510	0.4915

Nonostante le aspettative non otteniamo alla fine valori degni di nota: la regolarizzazione tramite L2 smonta la stabilità del nostro classificatore, rendendolo meno efficiente, mentre Early Stopping e Drop Out hanno sicuramente potenziale, ma che forse nel nostro caso non riesce ad esprimersi.

6.5. Ensemble Methods

In questa sezione verranno esaminati differenti Ensemble Methods che utilizzino sia tecniche di bagging che di boosting. Con il Bagging, vengono allenati in maniera indipendente differenti classificatori ognuno su un differente bootstrap sample del traning set (ossia con "rimpiazziamento", alcuni records potrebbero dunque essere ripetuti) e la predizione finale è ottenuta dall'aggregazione delle predizioni individuali di ogni classificatore. Con il Boosting, invece, vi sono dei "weak learners" allenati in maniera sequenziale, dove ognuno cerca di correggere gli errori presenti nel modello precedente ed ai records non correttamente classificati viene assegnato un "peso" più alto con una probabilità maggiore di essere selezionato più volte nei modelli successivi.

Inizialmente si è testato il **Random Forest** sul nostro dataset di 28 variabili, il quale, oltre ad utilizzare come approccio il "Bagging", seleziona per ogni sample del training un set di variabili random. Per prima cosa, si è scelto di procedere al tuning degli iperparametri "min_samples_split", "min_samples_leaf" e "max_depth. Dalla GridSearch si è ottenuto `{'min_samples_leaf': 1, 'min_samples_split': 5, 'max_depth': 9}`, e con 100 alberi si è ottenuto un 89.5% di accuratezza nel test, seguito da un 97.3% nella cross-validation e 98.9% nel train. Con questi parametri, inoltre, si è potuto osservare come la migliore performance di generalizzazione si ottenga con soli 60 alberi ottenendo un **89.9%** di accuratezza nel test e valori identici ai precedenti di cross validation e training. Si osserva, in basso, le variabili che l'algoritmo considera più importanti in termini di diminuzione dell'impurità media, ed in particolare si osserva come venga data priorità



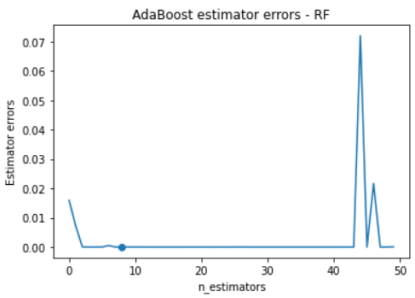
alle variabili concernenti il tempo del segnale di accelerazione della gravità mentre invece, sembrano assumere un'importanza minore le variabili relative al tempo del segnale di

accelerazione del corpo e tempo del segnale e di dominio della frequenza del corpo misurati dal giroscopio.

Passando invece al **Bagging**, si è provato ad utilizzarlo con differenti base classifier tra cui Logistic, Naive Bayes Gaussian e Decision tree. Le performance ottenute, per quanto riguarda la regressione logistica ed il Naive Bayes Gaussian utilizzati come base classifier, sono identiche a quelle che si ottengono considerando la regressione logistica ed il Naive Bayes senza utilizzare il Bagging (per il Naive, si ha un leggerissimo peggioramento). Per quanto riguarda, invece, il Decision Tree, con il Bagging si ottiene un lieve miglioramento nella performance di generalizzazione ottenendo un **88%** a fronte del 86% utilizzando il classificatore singolarmente.

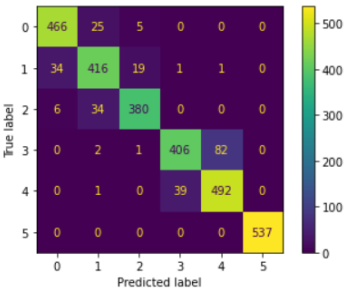
	Single classifier	With Bagging
Logistic Regression	*91.2%*	*91.2%*
Naive Bayes(Gaussian)	*87.5%*	*87.3%*
Decision Tree	*86%*	*99%*

Per le tecniche di **Boosting**, sono stati analizzati *AdaBoost, Gradient Boosting, XGboost e LIGHTboost.* Con **AdaBoost**, impostando come base classifier il Decision Tree con depth 8, numero di stimatori 100 e come algoritmo "SAMME.R", otteniamo in questo modo una performance di **91.3%** nel test che risulta essere la più alta tra il Decision Tree singolo e l'utlizzo del Bagging. Non si hanno, invece, miglioramenti della performance impostando Random Forest come base classifier, ed inoltre, come è possibile notare dal grafico degli errori di ogni stimatore, che l'errore del risulta essere minimo a partire già dai primi boosting. Ad esempio, utilizzando come n_estimators=7, ossia uno dei primi punti nel plot con il minimo errore, si ottiene un **89.9%** di performance di generalizzazione, e pur aumentando il numero di stimatori i risultati sono pressoché gli stessi, i quali coincidono con lo stesso valore trovato utilizzando singolarmente il Random Forest. Inoltre, utilizzando la Logistic Regression come base classifier, si ottengono, al contrario, dei peggioramenti rispetto all'utilizzo singolo del classificatore, in quanto si ottiene nel migliore dei casi un **89%** a fronte del 92% trovato nella sezione precedente della Logistic Regression.

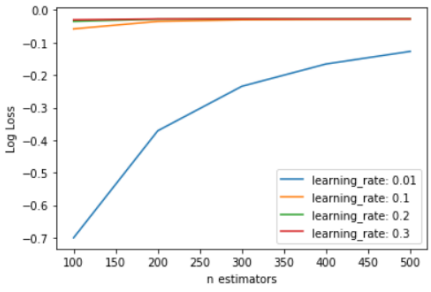


Per quanto riguarda, invece, il **Gradient Boosting**, ossia il classificatore che utilizza come base classifier dei "decision stumps", con una GridSearch si sono ottenuti i parametri `'learning_rate': 0.1, 'max_depth': 3`, e testando differenti numeri di stimatori si è ottenuta la miglior performance di generalizzazione nel caso di n_estimators=60, con un'accuratezza del **90.7%** nel test e 92.8% per la cross validation.

Utilizzando **HistGradientBoosting**, ossia utilizzando dei "bins", otteniamo un **91.5%** di accuratezza nel test, nel train 97% e 93.75% per la cross-validation. I parametri utilizzati per quest'ultimo, trovati sempre con il tuning, sono stati max_depth=3, learning_rate=0.08 e min_samples_leaf=50. I risultati della confusion matrix sono riportate sopra in tabella.

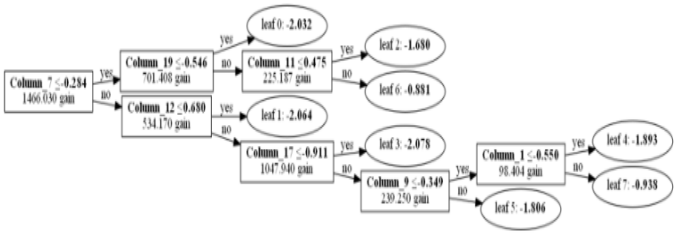


Per **XGBoost**, invece, si è plottato le differenti combinazioni di learning rate e numero di alberi utilizzando la GridSearch con scoring='neg_log_loss', dov'è possibile notare come di norma, all'aumentare del numero di alberi, le migliori performance si otterrebbero da un'aumento altresì del learning rate, anche se dal grafico si nota come, con il suddetto dataset, anche con pochi alberi ed un learning rate tra 0,1,0,2 e 0,3 le performance risultano essere già alte. Tuttavia, si è trovato che la Log Loss minore tra le varie combinazioni risultasse essere: `-0.026902`



using {'learning_rate': 0.2, 'n_estimators': 300}. Utilizzando tali parametri ed inserendo depth=5, reg_lambda=1 e min_child_weight=1 si è ottenuta un'accuratezza del test del **91.7%** a fronte di un 100% di accuratezza ottenuto nel train, e del 93.7% per la cross-validation.

Infine, con **LightGBM**, sono stati fissati alcuni iperparametri come bagging_freq': 5, feature_fraction': 0.8, 'subsample_for_bin':20000 e come parametri ottimali si sono ottenuti 'reg_alpha': 4, 'reg_lambda': 3, learning_rate': 0.2, max_depth': 7, num_leaves': 8. In questo modo l'accuratezza nel test risulta essere **92.2%**, per la cross-validation 93.1% e per il train 99% con 50 stimatori ed all'aumentare di questi l'accuratezza non varia. Anche provando ad aumentare max_depth l'accuratezza rimane invariata. La foto sottostante rappresenta il secondo albero prodotto dall'algoritmo mostrando le variabili scelte ed i relativi valori in termini di "split gain", e la root è rappresentata qui dalla colonna 7 (ossia la variabile "*tGravityAcc-min()-X*").



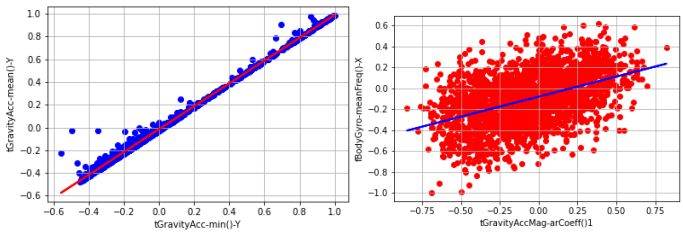
7. LINEAR REGRESSION

Al fine di selezionare adeguatamente le variabili da utilizzare si sono valutati due fattori. Per la regressione bidimensionale si è osservato un pairplot per quella multivariata, invece, si è calcolato il VIF (variance inflation factor) con lo scopo di comprendere il funzionamento dei vari algoritmi su modelli con diverso grado di multicollinearità.

feature	VIF		
tBodyAcc-entropy()-X	6.659564	tBodyGyro-min()-X	31.654431
tBodyAcc-correlation()-X,Y	1.850855	tBodyGyro-arCoeff()-Y,1	3.421671
tBodyAcc-correlation()-Y,Z	2.033321	tBodyGyro-correlation()-Y,Z	1.689664
tGravityAcc-mean()-Y	767.621752	tBodyGyroJerk-iqr()-Y	115.801643
tGravityAcc-max()-X	628.504939	tGravityAccMag-arCoeff()1	3.559376
tGravityAcc-max()-Z	3.441810	fBodyAcc-max()-X	44.188619
tGravityAcc-min()-X	747.266905	fBodyAcc-max()-Y	43.122039
tGravityAcc-min()-Y	455.076561	fBodyAccJerk-max()-Y	42.936568
tGravityAcc-energy()-Y	66.456906	fBodyAccJerk-bandsEnergy()-1,16	241.162386
tGravityAcc-arCoeff()-Y,1	80.310831	fBodyGyro-maxInds-X	24.846113
tGravityAcc-arCoeff()-Y,3	70.319966	fBodyGyro-meanFreq()-X	1.901710
tGravityAcc-arCoeff()-Z,2	92.361925	fBodyAccMag-mad()	113.773395
tGravityAcc-arCoeff()-Z,4	114.797802	fBodyAccMag-energy()	348.745694
tBodyGyro-max()-X	26.324663	fBodyBodyGyroJerkMag-std()	88.647771
		angle(Y,gravityMean)	335.342327

7.1 Regressione in due dimensioni

Si sono utilizzate diverse tecniche per individuare i parametri della retta di regressione: la regressione lineare classica attraverso l'implementazione della libreria sklearn, il Gradient Descent e la Maximum Likelihood Estimation (implementati manualmente). Si è considerata significativa la comparazione delle varie tecniche tra due diverse coppie di variabili, scelte utilizzando il pairplot. La prima con una palese correlazione lineare positiva, la seconda, invece, con un andamento grafico indicante sempre una correlazione positiva, ma meno netta. Si è presa questa scelta al fine di poter valutare un eventuale miglioramento attraverso la regressione multivariata mantenendo come variabile dipendente la stessa della seconda coppia. Di seguito gli scatter plot delle due



coppie di variabili con le rispettive rette di regressione.

I punteggi ottenuti non variano al variare della tecnica utilizzata.

Si sottolinea che il modello di regressione classico e la MLE non necessitano un tuning dei parametri dell'algoritmo. Infatti, nonostante per MLE fosse necessario inserire dei valori iniziali di coefficiente angolare, intercetta e sigma, questi non influenzano il risultato. Per quanto riguarda il Gradient descent, invece, il learning rate e il numero di iterazioni, se scelti senza criterio, possono portare a risultati pessimi, ma nuovamente l'inizializzazione dei valori non risulta cruciale. I valori migliori per la prima coppia sono un learning rate di 0.02 con 1000 iterazioni, per la seconda, invece, il learning rate è 0.2 con sole 90 iterazioni.

	Coppia 1	Coppia 2
R2	0.996	0.206
MSE	0.000	0.050
MAE	0.010	0.179

Si conclude, dunque, che a parità di risultato il modello di regressione lineare classico risulta il migliore, non necessitando dell'inizializzazione dei valori manuale, né di tuning dei parametri.

7.2 Regressione multivariata

Utilizzando come variabile dipendente la stessa della seconda coppia discussa nella sessione precedente si è voluto osservare il comportamento dei vari algoritmi sia considerando tutte le variabili sia mantenendo solamente quelle con un VIF minore di 5. Per gli algoritmi Lasso e Ridge, che presentano una correzione del modello di regressione lineare, si è effettuata una grid search per il valore di *alpha*, essenziale affinché il comportamento di tali modelli sia migliore di quello classico. Inoltre, in letteratura si è osservato come Lasso sia molto sensibile anche al preprocessing e come migliori con l'aggiunta di caratteristiche polinomiali (attraverso la funzione di sklearn PolynomialFeatures), nei risultati riportati si mostra come questo valga anche per Ridge. Si sono poi comparati i risultati anche con quelli di un gradient boosting regressor, utilizzato senza tuning dei parametri. Lo score riportato nelle tabelle è l'R2.

MODELLO CON PRESENZA DI MULTICOLLINEARITÀ TRA LE VARIABILI INDIPENDENTI

	No tuning	Grid search	Grid + polynomial
OLS train	0.381576	-	0.540091
OLS test	0.376688	-	0.438953
Lasso train	0	0.141888	0.311411
Lasso test	-0.000729	0.139645	0.308855
Ridge train	0.380138	0.382311	0.529681
Ridge test	0.374473	0.373662	0.464391

Il gradient boosting invece ottiene i seguenti risultati sul test set:

R2: 0.503

MSE: 0.031

MAE: 0.137

MODELLO SENZA MULTICOLLINEARITÀ

	No tuning	Grid search	Grid + polynomial
OLS train	0.251859	-	0.268925
OLS test	0.248627	-	0.262642
Lasso train	0	0.121538	0.258520
Lasso test	-0.000961	0.121082	0.249504
Ridge train	0.251175	0.252039	0.269371
Ridge test	0.251408	0.247855	0.260785

Il gradient boosting sul test set:

R2: 0.289

MSE: 0.044

MAE: 0.167

Il gradient boosting ottiene risultati migliori in entrambi per entrambi i dataset senza necessità di tuning. Si osserva, inoltre, che Lasso non raggiunge mai i risultati degli altri algoritmi e che, nonostante effettui sia una selezione delle variabili che una regolarizzazione, ottiene un risultato più vicino agli altri nel dataset senza multicollinearità. Ridge, la cui regolarizzazione mitiga proprio la multicollinearità, quando ottimizzato con tuning e preprocessing, riesce a superare il modello classico nel dataset con multicollinearità.

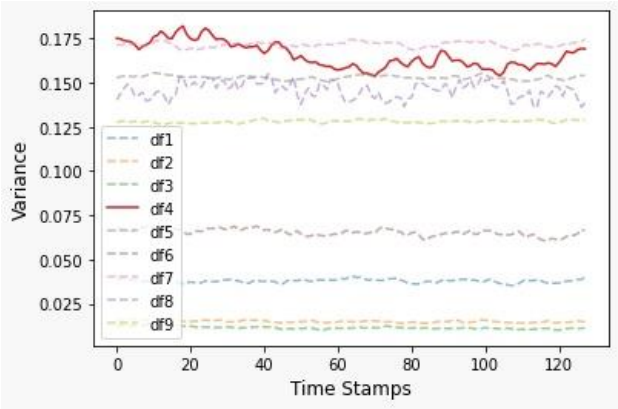
Conclusioni

In conclusione, è possibile affermare che nel complesso, tutti i classificatori analizzati forniscono ottime performance. In particolare, le migliori sono ottenute maggiormente dalle tecniche di Boosting, in particolare con XGBoost, LightGBM, HistGradientBoosting. Anche con Naive Bayes (utilizzando il Categorical), Logistic Regression e Neural network si sono ottenute delle alte performance, in particolare anche senza l'utilizzo del Bagging per le prime due. L'algoritmo di classificazione da cui otteniamo risultati "peggiori", ma pur sempre buoni, sono il Naive Bayes con il metodo Bernoulli (in quanto non adatto al nostro dataset) e Decision Tree. Per quest'ultimo, si ottengono dei miglioramenti inizialmente sia con il Bagging che con AdaBoost in quanto, da un 86% di accuratezza ottenuto dal Decision Tree si passa inizialmente ad un 89% ottenuto con il Bagging, e successivamente, con AdaBoost viene raggiunto un 91.3%. Dunque, è possibile affermare che per il nostro dataset i classificatori utilizzati singolarmente siano ottimali pur non utilizzando alcuna tecnica di Bagging ad eccezione per il Decision Tree.

8. TIME SERIES

Il dataset su cui si sta svolgendo il progetto è stato creato basandosi sull'estrazione di features da time series di segnali ottenuti da sensori spaziali. Questi segnali sono composti da più time series univariate. Ogni time serie dunque indica l'andamento di un segnale in un determinato arco temporale e può essere etichettata attraverso l'assegnazione della classe di movimento di appartenenza o dell'ID del soggetto che performa il movimento. Sono tre i tipi di segnali a disposizione: l'accelerazione totale , l'accelerazione del corpo, e la velocità angolare (i primi due ottenuti dall'accelerometro, il terzo dal giroscopio). Di ognuno si ha l'andamento sui tre assi *x*, *y* e *z*.

Si è deciso di visualizzare la varianza di ogni segnale al fine di individuare quale tra questi potesse contenere più informazione.



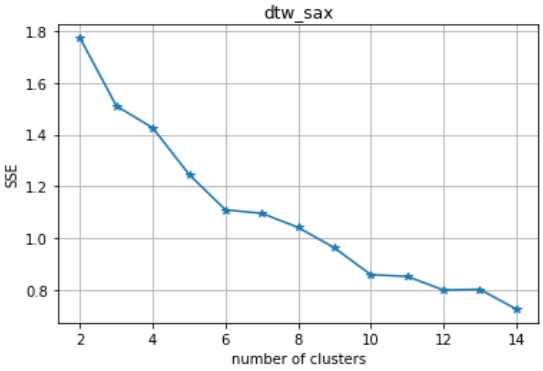
8.1 Clustering

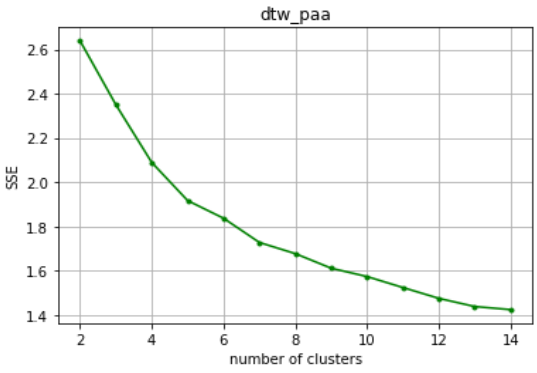
In queste sezione abbiamo usato il segnale con più varianza (df4 nel plot) che corrisponde alla velocità angolare misurata dal giroscopio sull'asse *x*. L'algoritmo utilizzato è il time series kmeans implementato dalla libreria tslearn comparando le performance al variare della metrica di distanza utilizzata, del preprocessing dei dati e del numero di cluster da individuare. Come metrica di valutazione si è utilizzata la SSE, poiché risulta computazionalmente molto complesso calcolare la silhouette con la metrica Dynamic Time Warping e, inoltre, non si ritiene corretto valutare un clusterizzatore basato su dtw con una distanza euclidea.

Per comprendere se fosse necessaria una standardizzazione dei dati si sono comparati i risultati ottenuti dal kmeans sui dati grezzi e su quelli trasformati attraverso l'*amplitude scaling*, ottenendo risultati decisamente peggiori sui dati trasformati, così come ripotato in tabella. Si è deciso dunque di utilizzare la standardizzazione solo come preprocessing per le approssimazioni delle time series.

	Euclidean	DTW
SSE_dati grezzi	17.22	4.51
SSE_dati standardizzati	108.82	26.01

Il numero di cluster è stato scelto soprattutto in base al task di analisi che si è voluto effettuare : 6 cluster per comprendere se ci fosse corrispondenza con le classi, 30 cluster per individuare i soggetti. Si è comunque deciso di visualizzare il variare della SSE all'aumentare del numero di cluster, si riportano, a titolo di esempio, i grafici ottenuti con metrica DTW sui dati approssimati con PAA e con SAX.





I valori ottenuti sui dati con e senza approssimazione sono riassunti nella tabella. Le stringhe utilizzate per nominare i vari "modelli" verranno utilizzati in seguito per riferirsi a tale configurazione.

	K = 6		K = 30	
	SSE	silhouette	SSE	silhouette
eu	15.06	0.420	11.15	0.411
dtw	3.59		2.38	
PAA_dtw	1.82		1.20	
SAX_dtw	1.25		0.42	
PAA_eu	3.47	0.112	2.16	0.099
SAX_eu	2.65	0.098	1.74	0.103

Si può subito notare come all'aumentare del numero di cluster diminuisca la SSE, questo risulta ovvio essendo tale misura la somma tra la distanza di ogni elemento dal cluster al quale è stato assegnato (al quadrato), infatti, osservando la silhouette, che introducendo il concetto di distanza tra cluster dà una misura di definizione del cluster, si può ottenere una valutazione più completa. Risulta dunque interessante rilevare che attraverso l' approssimazione con SAX all'aumentare del numero di cluster non migliori solamente la SSE ma anche la silhouette.

Per valutare la diverse configurazioni si sono utilizzate le classi di movimento per k = 6 , i soggetti per k = 30. Partendo da k =6, si è prima valutato il tipo di cluster creati dai diversi modelli, poi si sono suddivise le time series per classe e si è valutato se il clustering avesse individuato qualche proprietà. La prima evidenza scaturisce dal semplice conteggio del numero di time series per ogni cluster. Infatti, eu e dtw assegnano rispettivamente il 65,6 % e il 54,6 % delle time series ad un solo cluster, invece, le altre configurazioni le distribuiscono uniformemente. Inoltre, come si può notare da un'analisi più approfondita, il cluster più grande è lo stesso al quale vengono assegnate la quasi totalità delle time series delle classi SITTING, STANDING, LAYING. Si può dunque immaginare che questo cluster raccolga una proprietà comune alle tre classi, ad esempio il tratto di staticità. I modelli con dati approssimati invece non riescono ad individuare tale proprietà, inoltre, non ne evidenziano alcun'altra che possa essere collegata direttamente alle classi di movimento, poiché le time series appartenenti alla stessa classe vengono distribuite equamente tra i clusters e ciò avviene per tutte le classi. Nelle tabelle si riportano i diversi conteggi per le classi SITTING e STANDING, al fine di sottolineare la differenza tra l'utilizzo di dati approssimati e non, a titolo di esempio PAA_dtw e dtw.

PAA_dtw	SITTING	STANDING
0	275	222
1	215	275
2	216	314
3	193	248
4	128	172
5	279	143

dtw	SITTING	STANDING
0	1279	1360
1	5	8
2	0	2
3	1	2
4	1	2
5	0	0

Procedendo i maniera simile per K = 30 ed utilizzando dunque gli id dei soggetti come riferimento, non abbiamo ottenuto alcuna correlazione tra clusterizzazione e soggetti. Si pensa, tuttavia, che il dataset utilizzato non fosse adatto a questo scopo, e s'individua come possibile dataset adeguato, uno che comprenda i segnali provenienti da più sensori e su più assi di alcuni soggetti che performano un solo tipo di movimento.

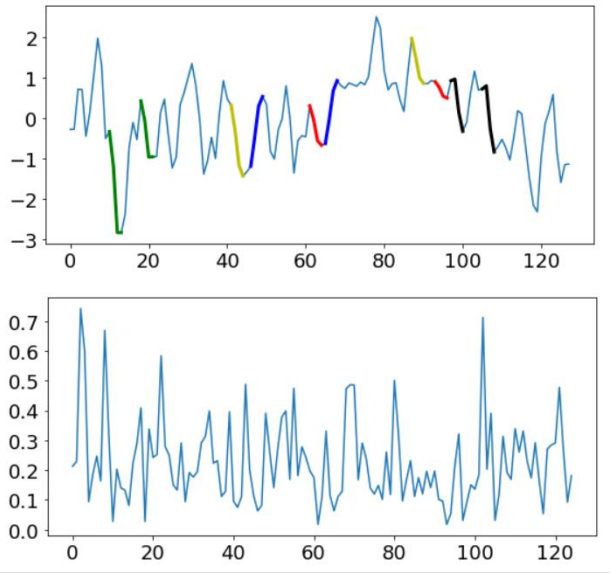
Si conclude, dunque, affermando che per il nostro tipo di dati il clustering attraverso kmeans con metrica euclidea e con metrica dtw possono individuare una proprietà comune alle classi statiche. A parità di risultato, quindi, si preferisce la metrica euclidea perché decisamente più veloce.

8.2 Motif and discords

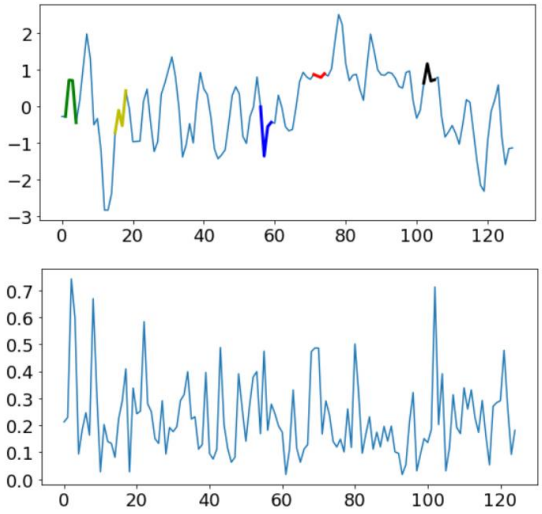
In questa sezione verranno analizzati i motifs ed i discords. I motifs consistono in sottosequenze ripetute all'interno di una singola times series ed, in questo caso, si è scelto di analizzare i motifs del segnale di accelerazione del corpo sull'asse x *(t_body_acc_x)*, come per la sezione di

classificazione. Anche per questa sezione è stato utilizzato lo scaler MeanVariance.

Analizzando la Times Series numero 5 del dataset utilizzato, si osservano in seguito, utilizzando una window di 4 e un valore massimo di motifs di 7, i seguenti motifs rilevati in relazione alla matrix profile:



Si osserva che i patterns riscontrati, oltre a risultare per definizione nei minimi locali della matrix profile, risultino dipendere, maggiormente, dalla pendenza negativa del segnale (4 motifs su 5). Osservando, invece, i discords sempre in relazione alla matrix profile, rilevando un numero k di anomalie uguali a 10 ed ex_zone=3, sempre nella stessa times series si rilevano le seguenti:



Si osserva come, i discords rilevati, risultino essere in concomitanza con alcuni picchi nella matrix profile, ad esempio nei casi evidenti rilevati nei timestamps 0 e 100 circa. Inoltre, tra il secondo discords ed il terzo intercorre abbastanza tempo.

8.3 Time Series Classification

A differenza del clustering dove, così come precedentemente menzionato, abbiamo selezionato ed elaborato il segnale "body_gyro_x" per via della sua varianza, la classificazione su questo segnale è risultata poco accurata.

Ognuno dei 9 segnali è stato elaborato e l'unico ad aver offerto risultati accettabili è stato "body_acc_x", su cui sono stati di seguito analizzati alcuni metodi.

8.2.1 KNN – Decision Tree

Come prima cosa, ci siamo curati di analizzare le differenze tra le varie metriche di distanza utilizzando un classificatore benchmark quale il KNN.

I dati sono stati normalizzati dapprima con il metodo TimeSeriesScalerMeanVariance, e anche con TimeSeriesScalerMeanMinMax così da sfruttare un'ulteriore comparazione utile alla nostra causa, e applicati ai modelli di entrambe le librerie SKLearn e PyTs, nonostante la seconda, nell'effettivo, non apporti sostanziali cambiamenti.

Differenti distanze sono state usate:

- Minowski
- Eucliedean
- Manhattan
- DTW_Sakoechiba
- DTW_Itakura
- DTW_Fast

Nonostante le aspettative che vedevano come una possibile chiave di sviluppo le metriche realizzate "ad-hoc" per le TS quali le varie Dinamic Time Warping, i classificatori che ottengono migliori risultati sono il Manhattan e l'Euclideo, che ottengono rispettivamente 62 e 63% di accuratezza.

Per non stazionare sulla sola metrica di accuracy, segnaliamo inoltre che questo classificatore fa fatica a riportare risultati accettabili sui livelli di Precision e Recall con alti livelli di varianza da classe a classe e valori che vanno dal 30% agli 85%.

Sfortunatamente in questo modello, così come in altri, non è stato possibile approfondire molto il discorso di classificazione e gestione degli hyperparameters per via dell'altissimo costo computazionale degli stessi. Va detto infatti che le varie metriche DTW sarebbero sicuramente potute esser sfruttate meglio.

Il discorso non cambia nei confronti del Decision Tree: diversi iperparametri (Min_sample_split, Min_sample_leaf, Max_depth e class_weight a sfavore della classe 6) sono stati alterati in più combinazioni, ma i risultati si tendono a peggiorare ulteriormente.

8.2.2 CNN

Al fine di sviluppare un modello più performante abbiamo scelto di modellare una Convolutional Neural Network che ha preso vita con il seguente scheletro:

Layer (type)	Output Shape	Param #
conv1d_15 (Conv1D)	(None, 121, 16)	144
activation_15 (Activation)	(None, 121, 16)	0
dropout_15 (Dropout)	(None, 121, 16)	0
conv1d_16 (Conv1D)	(None, 117, 32)	2592
activation_16 (Activation)	(None, 117, 32)	0
dropout_16 (Dropout)	(None, 117, 32)	0
conv1d_17 (Conv1D)	(None, 115, 64)	6208
activation_17 (Activation)	(None, 115, 64)	0
dropout_17 (Dropout)	(None, 115, 64)	0
global_average_pooling1d_5 (GlobalAveragePooling1D)	(None, 64)	0
dense_5 (Dense)	(None, 7)	455

Total params: 9,399
Trainable params: 9,399
Non-trainable params: 0

Diversi strati convuluzionali 1d sono stati alternati a drop_outs di 0.2 ed uno stato finale che interpretasse le features generate dal modello. Gli strati Conv1D con activation function "Relu" mentre lo strato finale "SoftMax". Il compilatore si è basato su "Sparse_categorical_Crossentropy" ed Optimizer ADAM.

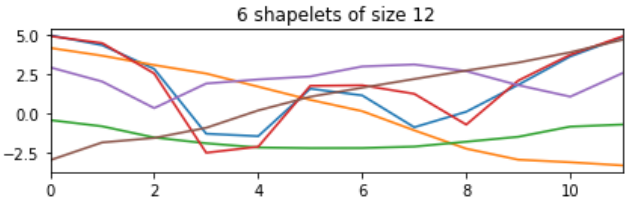
Il modello è stato implementato con l'ausilio di un Validation_Set delle dimensioni del 10% del Training_set ed un learning rate dinamico e con un lower bound settato a 0.001.

I risultati ottenuti dopo un quantitativo di epoch pari a 300 è stato del 71% di accuratezza su Test_set con valori di precision e recall più omogenei.

8.4 SHAPELET DISCOVERY

La computazione degli shapelet è risultata essere un grosso ostacolo al nostro studio in quanto avremmo voluto testare diversi metodi appartenenti a diverse librerie. Ciò non è stato possibile, per cui ci siamo limitati a sfruttare la libreria TSLearn ed i suoi metodi.

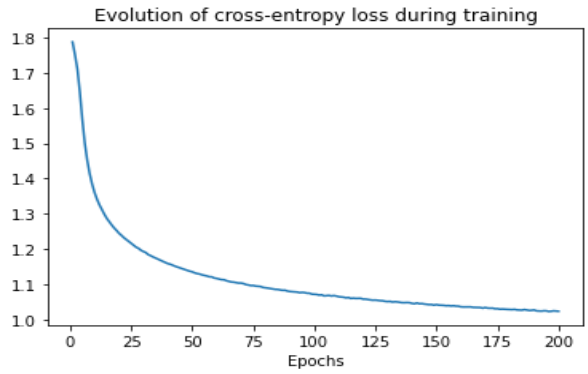
Grazie alla funzione grabocka_params_to_shapelet_size_dict la libreria ci ha fornito un dizionario composto da un numero suggerito di Shapelts (6) e una Time Windows (12) su cui applicare la misure.



Il modello LearningShapelets è stato definito tramite il tuning di diversi parametri, la cui combinazione migliore (seppure non ottimale) riporta come #Shapelets chiaramente 6, Window di 12, Optimizer ADAM con learning_reate dello 0.001, un batch di 32 elementi ed un weight regularizer di 0.001.

I risultati di questa configurazione non si distanziano molto da quelli dei primi classificatori, raggiungendo valori del 60% di Accuratezza, con Precision Recall ed F1-Score particolarmente eterogenei.

Come è inoltre possibile notare dal seguente grafico l'entropia tende sì a migliorare raggiungendo livelli minimi intorno alle 200 epochs, livelli minimo che però risultano comunque lontani da risultati ottimali intorno allo zero, e non oltre.



8.3.1 LSTM Model

Per creare un ultimo modello che generasse delle performance migliori abbiamo voluto prendere in considerazione tutti i 9 segnali a nostra disposizione e creare un unico dataset a tre dimensioni: Train_set (7352, 128, 9) e Test_set (2947, 128, 9).

Questo ci ha permesso di creare un modello LSTM che potesse lavorare in parallelo su un grande quantitativo di dati, e la cui peculiarità è di poter interagire con raw data senza problemi.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm_11 (LSTM)               (None, 100)               44000

 dropout_14 (Dropout)         (None, 100)               0

 dense_22 (Dense)             (None, 100)               10100

 dense_23 (Dense)             (None, 6)                 606

=================================================================
Total params: 54,706
Trainable params: 54,706
Non-trainable params: 0
```

Per svilupparlo abbiamo utilizzato la libreria Keras. Un modello sequenziale è stato inizializzato e così è stato composto: Un singolo hidden layer LSTM seguito da un DropOut dello 0.2 per ovviare ad eventuale OverFitting ed uno strato finale denso per interpretare i risultati del modello. Come loss function infine si è sfruttata la "Categorical_Crossentropy" con ottimizzazione stocastica ADAM, con focalizzazione sull' Accuracy.

Lo schema qui riportato rappresenta quanto appena detto.

Infine, dopo 10 tentativi composti da 200 epochs l'uno e realizzati per via della natura stocastica del modello, otteniamo ottimi risultati che, in m edia, riportano il 90.431% di Accuratezza gener ale, il che lascia intendere come il dataset sia di ottima qualità se considerato in un'ottica di com pletezza.

9. Sequential pattern mining

Per questa sezione si è utilizzato il segnale corrispondente alla velocità angolare misurata dal giroscopio sull'asse *x*, per lo stesso motivo per cui è stato utilizzato nella sezione di Clustering.

Le time series di questo segnale sono state discretizzate attraverso la Simbolic Aggregate Approximation (SAX) e trasformate in vettori di lunghezza 20, utilizzando un alfabeto di 10 simboli. La libreria utilizzata è *prefixspan*, la quale si basa su Prefixspan (Prefix-projected Sequential pattern mining)[1], un metodo la cui idea è quella di considerare solamente i prefissi frequenti, invece che le intere sotto-sequenze frequenti.

Com'è noto, all'aumentare della lunghezza della sotto-sequenza, il supporto di questa diminuisce rispetto al supporto di quella che l'ha generata. Le analisi effettuate hanno dimostrato come le sequenze con supporto maggiore del 50% siano al massimo di lunghezza due. Inoltre, la sequenza di lunghezza quattro più frequente ha un supporto del 20 %, ma tale misura cala drasticamente per la sequenza più frequente di lunghezza cinque (0.01%). Si è dunque deciso di analizzare le sequenze di lunghezza quattro, il cui supporto varia dal 20 % al 0.001%. Tra queste, quelle con un supporto maggiore del 10% sono 4819, ed osservando invece solo quelle con supporto vicino al 20% si nota una prevalenza dell'elemento 4. Questo è anche l'elemento con supporto maggiore in generale (supporto del 90%) e la sotto-sequenza di lunghezza due con maggiore supporto, 66%, è [4, 4].

Analizzando invece sequenze con lunghezze maggiori (più di 8 elementi), nonostante il loro supporto non sia significativo, si riscontra che la quasi totalità delle sequenze inizia con l'elemento 0 e termina con l'8, inoltre osservando anche il secondo e il penultimo elemento si riscontra, con meno frequenza, la stessa caratteristica.

[1] PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, Meichun Hsu. Proceedings of the 17th International Conference on Data Engineering, 2001.

1 Sequenze di lunghezza 8 con supporto maggiore

```
[(134, [0, 0, 1, 5, 6, 7, 8, 8]),
 (134, [0, 0, 1, 2, 6, 7, 8, 8]),
 (132, [0, 0, 1, 5, 7, 7, 8, 8]),
 (131, [0, 0, 2, 5, 6, 7, 8, 8]),
 (130, [0, 0, 0, 2, 6, 7, 8, 8]),
 (129, [0, 0, 1, 2, 5, 6, 7, 8]),
```

2 Sequenze di lunghezza 11 con supporto maggiore

```
[(53, [0, 0, 1, 2, 3, 5, 6, 7, 8, 8, 8]),
 (53, [0, 0, 1, 2, 4, 5, 6, 7, 8, 8, 8]),
 (52, [0, 0, 0, 1, 2, 4, 5, 6, 7, 8, 8]),
 (52, [0, 0, 0, 1, 2, 5, 6, 7, 8, 8, 8]),
 (52, [0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 8])]
```

Inoltre, attraverso un esempio fornito dalla libreria, si sono trovate le sotto-sequenze frequenti più lunghe per ogni sequenza, osservando come tutte siano composte dai 12 ai 16 elementi.

10. Advanced Clustering

In questa sezione si analizzeranno i risultati ottenuti da due algoritmi di clustering avanzato: X-means e Optics. Per l'analisi si è utilizzato il dataset visto in precedenza nella sezione della Classificazione ed i dati sono stati normalizzati con MinMax.

L'algortimo **X-means** è considerato una variante del K-means, ed affronta alcune limitazioni di quest'ultimo come, ad esempio, la scelta del numero K in quanto non viene più fornito come input. Utilizzando la libreria pyclustering, l'algoritmo ha rilevato la presenza di *20 clusters* (il valore di default impostato come numero massimo) di cui 4 contengono un numero elevato di osservazioni rispetto agli altri (rispettivamente contengono 1022, 773, 614 e 609 osservazioni). Plottando uno scatterplot rappresentativo gli indici delle variabili *'tBodyAcc-entropy()-X' e 'tBodyAcc-correlation()-X,Y'*, i centroidi di appartenenza , in alcune aree come quella caratterizzata da alti valori di tBodyAcc-correlation()-X,Y e valori medio/bassi di tBodyAcc-entropy()-X, risultano molto ravvicinati tra loro (8 centroidi) e solo nella zona centrale si trovano pochi centroidi abbastanza separati dagli altri. Si è inoltre utilizzato la PCA per ridurre le dimensioni del dataset sia a 2 che a 3 in modo da meglio poter osservare i risultati ottenuti graficamente e valutare le differenze ottenute da essi. Utilizzando un dataset 3D, l'algoritmo rileva un numero di cluster inferiore ossia *17*, individuando un errore totale di ***267 (WCE)***. Tali numeri diminuiscono ancora di più utilizzando il dataset a 2D, in particolare trovando 12 cluster con un errore totale di 125. In generale, i valori di silhouette che si ottengono, sono abbastanza bassi, per il 2D arrivano a **0.33**, per il 3D a **0.35** ed utilizzando tutte le dimensioni si scende a **0.13.**

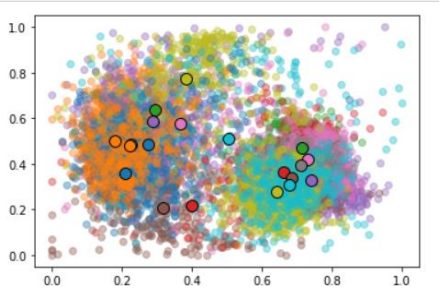
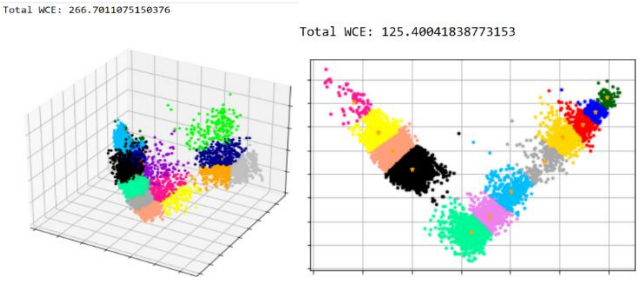


Figure 14. Plot x-means con indici delle 2 variabili



Per quanto riguarda, invece, l'utilizzo di **Optics**, si è utilizzato sia la libreria di scikit-learn e quella di pyclustering. Con la libreria di scikit-learn, si è voluto vedere le differenze che si ottenessero nei cluster utilizzando come metodo sia 'xi' che 'dbscan'. Con 'xi', impostando min_samples=5 ed eps=0.05, si ottiene uno score di silhouette molto basso utilizzando il dataset con le 28 dimensioni. Utilizzando, anche in questo caso, una PCA per ridurre le dimensioni a 2 si ottengono risultati migliori di silhouette ossia **0.54**, valore che rimane pressocchè invariato provando differenti valori di min_samples ed eps. Anche qui, per esemplificazioni grafiche, si riporta il dataset 2D selezionando come variabili le stesse scelte per xmeans ed il *Reachability plot*, da cui è possibile rilevare, nelle "valli", i punti appartenenti ad uno stesso cluster ed, al contrario, dove la reachability distance è più elevata e dunque alta la distanza di un punto dai suoi vicini, automaticamente quell'area sarà meno densa e saranno considerati outliers.

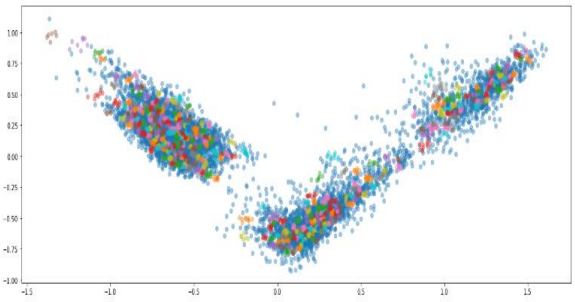


Figure 15.Plot 2D, optics con indici delle 2 variabili

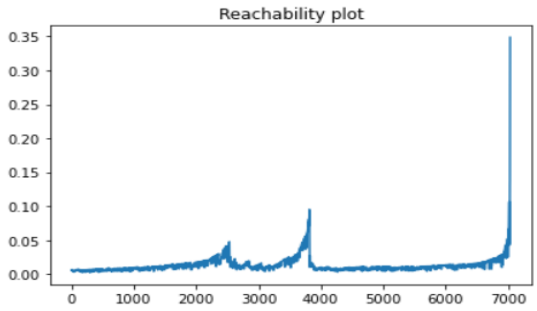


Figure 17. Reachability plot Optics

Utilizzando, invece, come metodo *il 'dbscan'*, per ottenere dei risultati migliori seppur non ottimali si è dovuto utilizzare i seguenti parametri: min_samples=4 ed eps=0.08, ottenendo una silhouette identica rispetto al metodo precedente ossia 0.54, piccole variazioni di tali parametri risultavano in una silhouette vicino lo 0 o addirittura negativa. A livello grafico, prendendo come riferimento le due variabili finora osservare e confrontando il plot in Fig.18 rappresentante le classi in 2D, si denota una distinzione chiara dei cluster in questo caso nella Fig.19, ed in particolare una distinzione netta delle classi relative a situazioni statiche (la zona verde in Fig.19) e le classi relative al movimento (la zona arancione in Fig. 19), con una piccola zona di rumore definita dai puntini celesti ed un mini cluster (quello rosso in alto a sinistra).

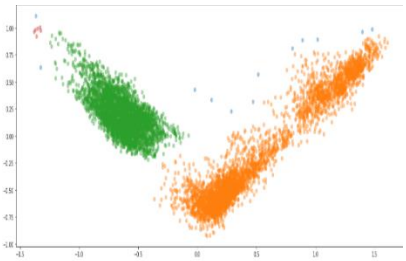


Figure 19.PLot 2D - Optics method 'dbscan'

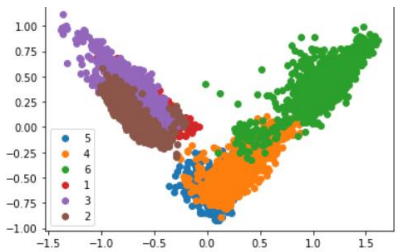


Figure 18. Plot 2D delle labels dataset originale

Si riportano, inoltre, i risultati grafici ottenuti anche dalla libreria di *pyclustering*, la quale identifica dal reachability plot un numero di **4 clusters.**

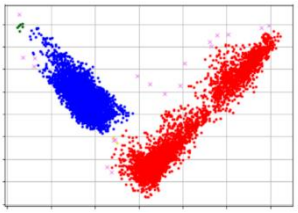


Figure 20. Plot 2D - Optics con pyclustering

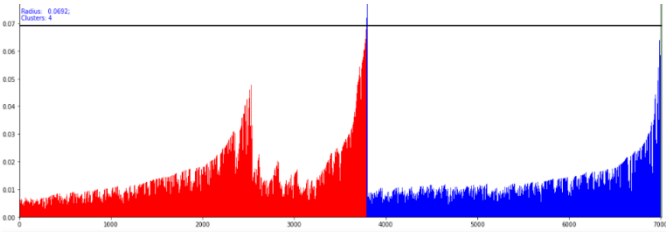


Figure 16. Plot reachability distance, pyclustering

11. EXPLAINABILITY

Per questo modulo si è scelto di prendere in considerazione uno dei classificatori ensamble proposti nel paragrafo 6.5, l'Extreme Gradient Boosting, e sfruttarlo per applicare su di esso alcune tecniche di explainability e mostrare come in letteratura queste tecniche di rappresentazione visiva stiano prendendo sempre più spazio per via del loro impatto.

Abbiamo scelto di sfruttare lo **SHAP** come explanation model in quanto uno dei migliori a livello visivo e soprattutto computazionale, ed applicare la corrispondente libreria per avere un'idea delle feature importance, in relazione prima ad un aspetto globale del modello, per poi prendere in considerazione due specifici campioni, contraddistinti da due classi di poli opposti.

Per quanto riguarda il comportamento delle features a livello globale abbiamo tenuto in considerazione il plot delle features importances generato dalla Random Forest del modulo 6.5 così da tracciare la bontà dei modelli globali realizzati dalla libreria SHAP:

Come è possibile vedere dalla Fig.21, vengono rappresentate le 9 (+1 sommatoria) features più importanti in ordine decrescente, combinate tra di loro per importanza ed impatto sul modello.

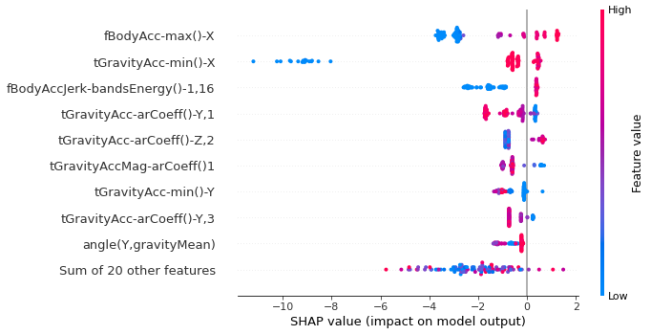


Figure 21. SHAP GLOBALE

I valori di tGravity rispecchiano le aspettative in quanto in linea con lo schema random forest, impattando in maniera sostanziale le decisioni del classificatore.

Il plot in Fig.22 ripete, con grafica differente quanto appena detto.

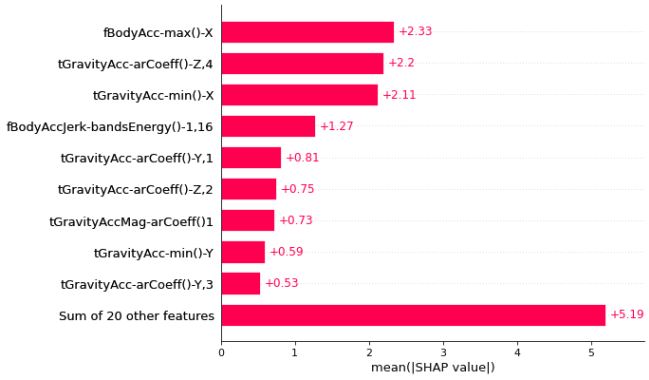


Figure 22. SHAP GLOBALE

Per ciò che concerne lo studio delle features a livello locale sono stati estratti due campioni dal dataset, un primo appartenente alla classe 1 (Walking) ed un altro, all'opposto, alla classe 6 (Laying) per poter valutare l'impatto delle features nel training del modello.

La Fig. 23 mostra lo come ogni variabile contribuisca a "spingere" il risultato di prediction verso una determinata classe, bilanciando il risultato in un valore finale corrispondente alla classe scelta.

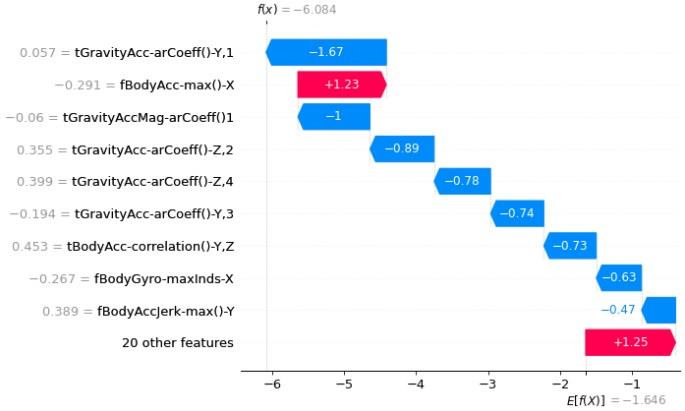


Figure 23. SHAP LOCALE CLASSE 1

Lo stesso vale per il force plot in Figura 24 che mostra come ci sia uno discreto bilanciamento tra "forze".

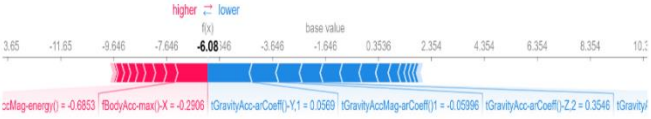


Figure 24.SHAP LOCALE CLASSE 1

Interessante è invece il caso del campione di classe 6 preso in considerazione in quanto conferma, come riportato in altra occasione come siano presenti variabili chiaramente discriminanti per la suddetta classe e come, così come si vede dal waterfall plot in Figura 25, il classificatore vada in maniera abbastanza diretta a trarre le sue conclusioni su di questa classe.

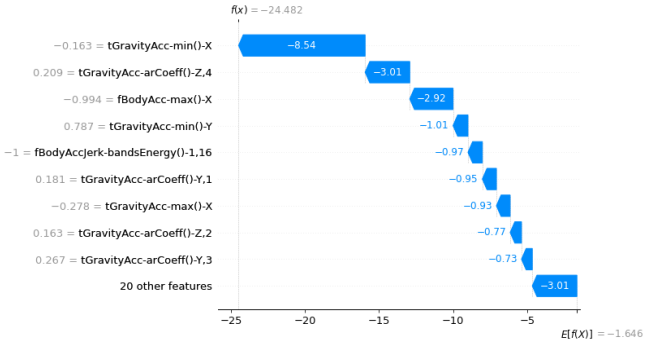


Figure 25. SHAP LOCALE CLASSE 6

Particolare è il force plot in Figura 26 che conferma quanto appena detto, in quanto facilmente identificabili fattori che "spingono" sempre nella stessa direzione.

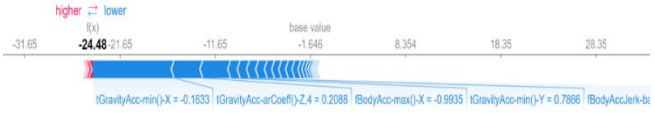
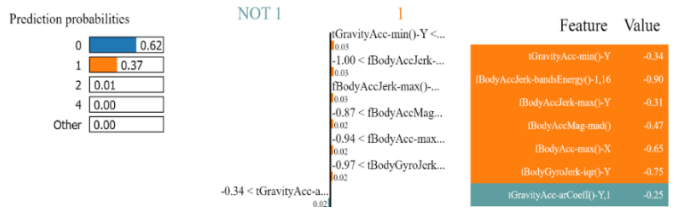


Figure 26. SHAP LOCALE CLASSE 6

Si è infine voluto provare in parallelo un modello **LIME** in questo caso direttamente (e non solo come riferimento) sul random forest (Notare le classi che il modello genera appartengono a [0,5] e non [1,6].

Come possibile vedere in figura 27 e 28, seppur con alcune lievi differenze, il modello, a livello locale dei due campioni selezionati, va a confermare quanto già visto sopra, mostrando in aggiunta le probabilità di split del modello su una o altre classi: nel caso del record appartenente alla classe 1 il modello applica una probabilità più bilanciata attribuendo circa il 62% di probabilità alla classe corretta ed il restante alla "seconda in classifica", mentre l'elemento di classe 6 non lascia dubbi con un 100% netto.

Figure 27. LIME LOCALE CLASSE 1



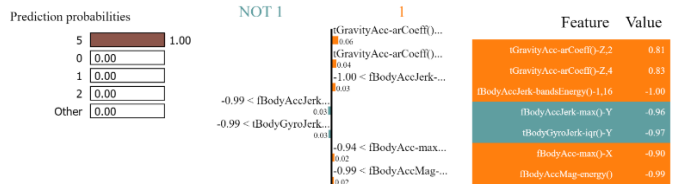


Figure 28. LIME LOCALE CLASSE 6

CONCLUSIONI

Il dataset analizzato ha sempre fornito, in quasi tutti le analisi, dei risultati ottimali soprattutto per quanto riguarda la parte di classificazione, anche in virtù del fatto che in origine, il dataset, fosse ben strutturato.

Nella prima sezione, sono state affrontate differenti tecniche di Feature Reduction riscontrando, come miglior metodo, RFE, che ha rappresentato successivamente la base delle sezioni successsive.

Nella seconda parte di Classificazione, tutti i classificatori analizzati hanno fornito ottime performance. In particolare, le migliori sono ottenute maggiormente dalle tecniche di Boosting, in particolare con XGBoost, LightGBM, HistGradientBoosting. Anche con Naive Bayes (utilizzando il Categorical), Logistic Regression e Neural network si sono ottenute delle alte performance, in particolare anche senza l'utilizzo del Bagging per le prime due.

Per la parte delle Times Series, si è scelto per la parte di Clustering di utilizzare un segnale con una varianza elevata, ossia ""body_gyro_x". Per la ricerca dei motifs e disords, si è preso come riferimento il segnale "*t_body_acc_x",* riscontrando dei pattern in una times series del segnale che risultavano dipendere dalla pendenza negativa del segnale. Per la parte di classificazione, si sono ottenuti risultati discreti con CNN ed anche con le Shapelets, mentre invece si ottengono risultati buoni con LSTD Model.

Per l'ultima parte, per quanto riguarda il Sequential Pattern Mining è stato utilizzato il metodo SAX. Nella sezione Clustering, si è adottato X-means e Optics, ottenendo risultati buoni seppur non ottimali nella silhouette, ed è stata applicata una PCA per meglio visualizzare i clusters. Per la parte di Explanaibility, infine, si è scelto l'XGBoost da analizzare utilizzando SHAP e LIME per Random Forest.