Riconoscimento di movimenti e posture del corpo
Report per l'Esame di Fondamenti di Machine Learning

Francesco Baraldi
132097
Corso di Laurea in Ingegneria Informatica
256745@studenti.unimore.it

Abstract

Questo documento presenta un'analisi di diversi modelli di machine learning per il riconoscimento delle attività umane (Human Activity Recognition, HAR). Lo scopo è quello di riconoscere correttamente la posizione di una persona tra 5 possibili classi, per farlo si è utilizzato un dataset che contiene i dati che quattro accelerometri, posti su quattro diverse posizioni del corpo, hanno raccolto in un esperimento di 8 ore su quattro soggetti diversi. Inoltre vengono utilizzati dati generici sui soggetti, come il nome, il peso, l'altezza, l'indice di massa corporea, l'età e il sesso. Gli accelerometri sono posti sulla vita, sulla coscia sinistra, sul braccio destro e sulla caviglia destra. Il dataset contiene più di 165 mila samples.

1 Introduzione

La ricerca nel campo dello Human activity recognition (HAR) ha diverse applicazioni, soprattutto in ambito medico, infatti può permettere alle persone più anziane, ma anche a persone più deboli o malate, di essere monitorate e in caso di caduta o incidente la tecnologia può intervenire tempestivamente, riconoscendo un'anomalia nella posizione del soggetto, consentendo quindi maggiore sicurezza.

I possibili approcci allo HAR sono due, l'**image processing** consiste nell'analizzare immagini e video per monitorare lo stato di un soggetto, questo implica però che il controllo può essere fatto solamente in determinati luoghi dove sono presenti per esempio delle telecamere adibite allo scopo specifico. Un altro approccio è basato su **sensori indossabili** come degli accelerometri, in questo caso il soggetto dovrà indossare questi accessori per poter effettuare il controllo ma non ci saranno vincoli di spazio. Con il veloce sviluppo dei dispositivi IoT (Internet of Things) questo approccio potrà essere sempre più facile da implementare, per esempio integrando dei dispositivi per il monitoraggio direttamente nei vestiti. Un possibile problema potrebbe essere la miniaturizzazione dei dispositivi hardware, e la relativa gestione energetica, infatti questi dispositivi avranno bisogno di energia per funzionare.

In questo report però ci si limita ad analizzare un problema più ristretto, cioè quello di riconoscere correttamente la posizione di una persona tramite algoritmi di machine learning, in particolare partendo dai dati di 4 accelerometri posti sulla vita, sulla coscia sinistra, sul braccio destro e sulla caviglia destra di 4 soggetti in un esperimento durato 8 ore, si cercherà di classificare correttamente la posizione con degli algoritmi di machine learning. Questi dati sono raccolti, con altre caratteristiche dei soggetti che hanno partecipato all'esperimento (nome, sesso, età, altezza, peso, indice di massa corporea), in un dataset di oltre 165 mila samples in totale. Lo scopo è quello di riconoscere correttamente la posizione di un soggetto con le features disponibili, quindi è un problema di classificazione e le classi sono cinque:

- **sitting-down**
- **standing-up**

- **standing**
- **walking**
- **sitting**

Il documento prosegue con la sezione 2, nella quale viene fatta un'analisi dei dati utilizzati, in particolare verranno mostrati i risultati dell'exploratory data analysis (EDA) del dataset. Nella sezione 3 vengono discussi i possibili modelli di machine learning che possono essere applicati a questo problema e descritti quelli che sono stati scelti. Inoltre per ogni modello scelto verranno presentate tre versioni: una con i dati in input grezzi, non processati, una seconda versione con in input i dati pre-processati, e infine la terza versione sarà la migliore versione tra la 1 e la 2 in ensemble. Nella sezione 4 viene descritto il processo di implementazione dei vari modelli scelti, come è stato fatto il tuning degli iperparametri, il pre-processing e il metodo di valutazione. Infine nella sezione 5 si traggono le conclusioni con un confronto tra i vari modelli.

2 Analisi dei dati

Il dataset [1] contiene 165633 sample e 19 features, dopo una veloce analisi dei dati è stata fatta una conversione di tipo, da stringa a tipo numerico, in particolare la feature *body_mass_index* e la feature *how_tall_in_meters* sono state convertite da stringhe a tipo numerico. Inoltre la feature *z4*, che rappresenta la coordinata z del quarto accelerometro, avendo un sample con un valore non numerico è stata convertita in modo forzato in tipo numerico, trasformando il valore errato in NaN, dopodichè è stato scelto di eliminare il sample con questo valore dal dataset. Nel resto dei dati non sono presenti valori nulli o anomali. Il dataset risulta sbilanciato per quanto riguarda la feature target, chiamata *class*, come si vede dalla figura 1 infatti sono presenti più samples per le classi *sitting*, *standing* e *walking*. Quindi sarà necessario stratificare durante lo splitting del dataset e utilizzare delle metriche opportune, come l'accuracy bilanciata invece dell'accuracy semplice.

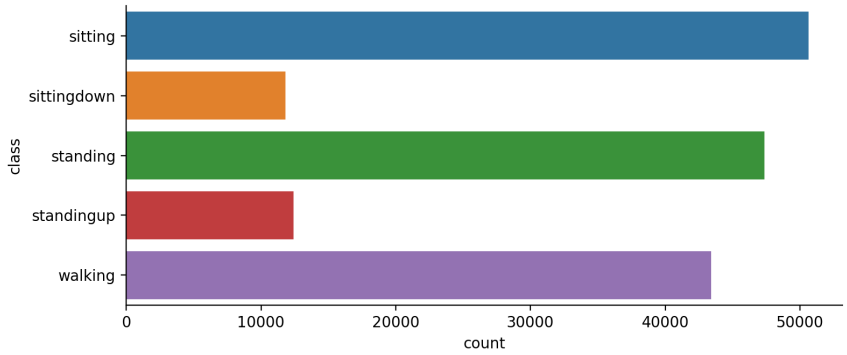


Figure 1: Bilanciamento della variabile target.

Come si vede invece dalla figura 2 i sample raccolti per ogni soggetto dell'esperimento sono bilanciati per tre soggetti mentre sono in misura minore per il soggetto *jose_carlos*. Dalla figura 3 si vede invece per ogni soggetto la percentuale di sample relativi a ogni classe, e questo rispecchia il bilanciamento generale delle classi mostrato in figura 1, infatti le percentuali sono simili, in particolare le classi *sittingdown* e *standingup* sono presenti in misura minore.

In figura 4 sono rappresentate le distribuzioni di probabilità dei dati raccolti dall'accelerometro 2, cioè quello posto sulla coscia sinistra, classificati in base alla posizione dei soggetti. Si può notare come nel caso della classe *sitting* e *standing* tutte e tre le coordinate sono più concentrate nell'intorno di un unico valore, infatti la cosiddetta *campana* è più stretta e alta. Questo può significare che se il valore letto è lontano dal valore di riferimento difficilmente la posizione sarà una delle due tra *sitting* e *standing*.

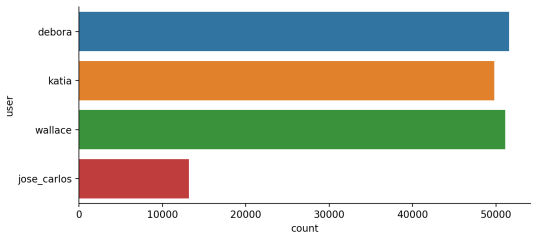


Figure 2: Bilanciamento dei dati raccolti per ogni soggetto.

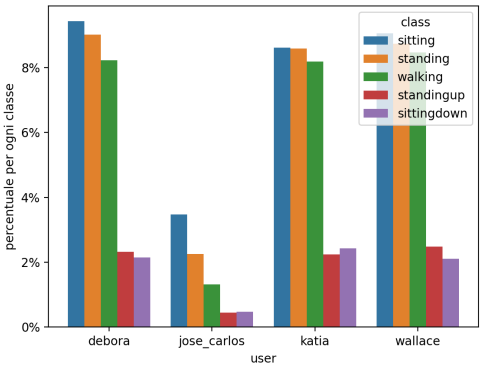


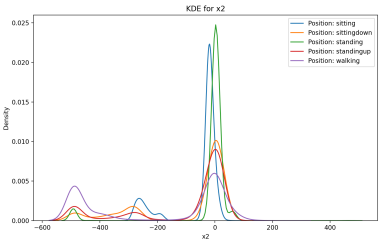
Figure 3: Bilanciamento delle classi per ogni soggetto.

Nella figura 5 si vede la relazione tra le coordinate dell'accelerometro 2 e le coordinate di tutti gli accelerometri divisi per classe. Le informazioni utili che si ricavano da questo grafico sono che le coordinate dell'accelerometro 3, quello sulla caviglia destra, differiscono da quelle dell'accelerometro 2 nel caso di posizione *standingup* e *walking*, infatti la zona rossa, che rappresenta la posizione *standingup*, assume valori minori in termini di coordinate dell'accelerometro 3. Il resto delle classi non sono distinguibili e le coordinate assumono un range di valori simile tra loro.

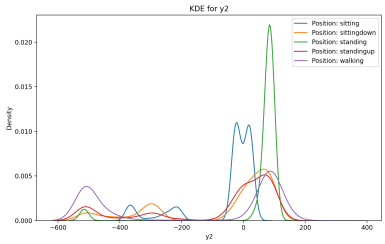
L'accelerometro 1, quello posto sulla vita, legge delle coordinate a seconda della posizione che sono rappresentate in figura 6. Come si vede dai tre grafici rappresentanti le tre coordinate, i valori non variano molto al variare della posizione, eccezion fatta per la coordinata z che diminuisce il suo valore in posizione *sitting down*. Questa informazione è ragionevole, infatti la vita è la parte del corpo che cambia meno la sua posizione nello spazio al variare della postura. Quando ci siede invece l'altezza della vita (rappresentata dalla coordinata z) si abbassa e quindi diminuisce la coordinata relativa. Inoltre dai grafici 6a, 6b, 6c è possibile notare degli high leverage point presenti nei dati, cioè quei sample che differiscono dagli altri in termini di valore di una feature, in questo caso le feature *x1*, *y1* e *z1*.

Nella fase di pre-processing delle variabili è stato fatto un encoding delle features categoriche, in particolare la feature *gender* è stata trasformata con un oggetto di tipo `LabelEncoder()`, che ha mappato i due possibili valori, *man* e *woman*, in numeri, *0* e *1* rispettivamente. Inoltre è stato fatto l'encoding con lo stesso metodo anche della variabile target, *class*, mappando le cinque classi con numeri da *0* a *4*. Dopodiché è stato effettuato il min-max-scaling alle features, che porta il valore di ogni feature tra 0 e 1, per velocizzare il training dei modelli. Inoltre per alcuni modelli è stato deciso di fare una riduzione delle features tramite la PCA (Principal Component Analysis), che sarà però approfondita nella sezione 4. Infine il criterio di splitting scelto per il dataset è stato il seguente:

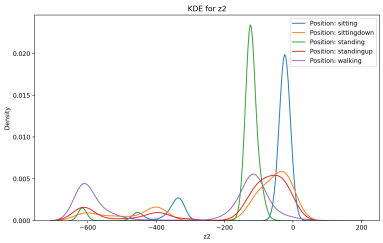
- **training set**: 80% del dataset originale
- **test set**: 20% del dataset originale



(a) Distribuzione della coordinata x.



(b) Distribuzione della coordinata y.



(c) Distribuzione della coordinata z.

Figure 4: Densità delle coordinate dell'accelerometro 2.

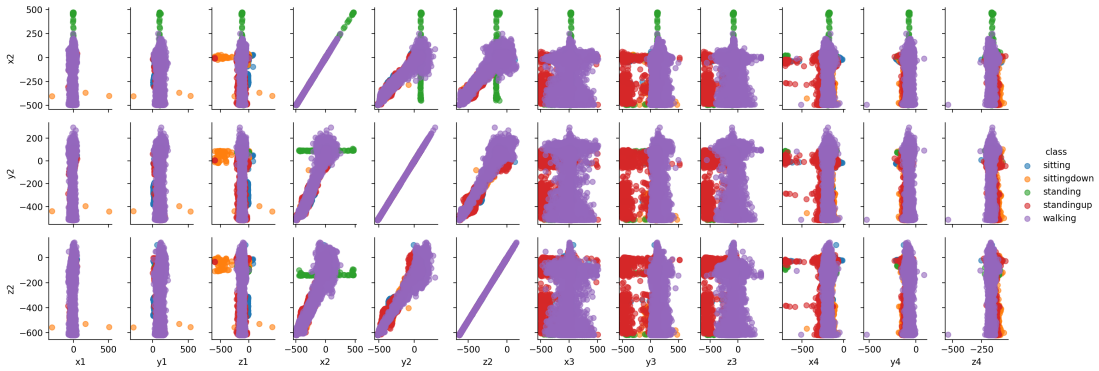


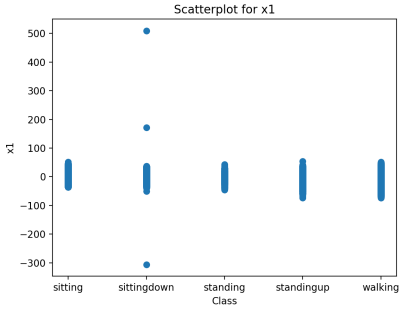
Figure 5: Pairplot delle coordinate dell'accelerometro 2.

Lo splitting è stato fatto in modo stratificato in modo da effettuare la divisione mantenendo bilanciate le classi della variabile target.

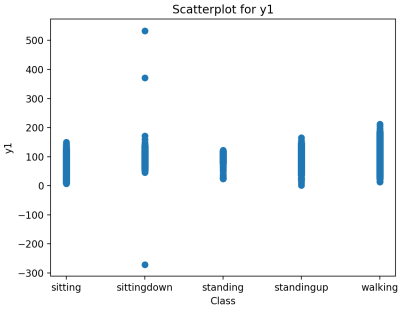
In fase di cross validazione sarà poi effettuato un ulteriore splitting del training set, in validation set e development set per fare la model selection per la scelta opportuna dei parametri dei vari modelli, e la model assessment per stimare le prestazioni dei modelli.

3 Discussione dei modelli

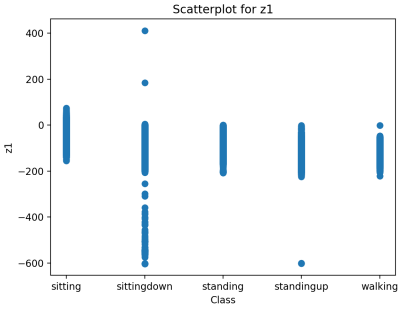
Il problema in esame è un problema di classificazione perché è necessario appunto "classificare" la postura di un soggetto tra cinque possibili posizioni. Gli algoritmi di machine learning adatti a questo tipo di problema sono diversi, i principali sono la *softmax regression*, che è come la logisitc regression ma per il caso multiclasse e utilizza la funzione softmax invece della funzione logistica; il *classificatore naive Bayes*, basato principalmente sul teorema della probabilità di Bayes; il *K-Nearest neighbors*, che assume che sample simili siano vicini nello spazio delle features, quindi calcola la distanza relativa tra le features, per ogni sample prende i K samples più vicini e ne restituisce la moda delle label. Il parametro K è quindi un iperparametro da scegliere opportunamente. Il **decision tree**, o **albero decisionale** in italiano, è un metodo



(a) Scatterplot della coordinata x.



(b) Scatterplot della coordinata y.



(c) Scatterplot della coordinata z.

Figure 6: Scatterplot delle coordinate dell'accelerometro 1.

di apprendimento nel quale i dati sono suddivisi in base a un certo parametro in sottoinsiemi sempre più piccoli, fino ad arrivare ad avere insiemi che non sono più divisibili, questi saranno le foglie dell'albero e definiranno la label. In particolare gli alberi decisionali sono composti da:

- **nodi**: controllano il valore di un determinato parametro sui dati,
- **rami**: collegano più nodi e rappresentano l'esito di un test fatto dal nodo padre,
- **foglie**: sono i nodi terminali e determinano il risultato finale.

Per selezionare le features su cui effettuare il test e il valore *cutpoint*, cioè il valore limite da testare, si utilizza il **recursive binary splitting** che ad ogni passo seleziona le features e il cutpoint che riducono maggiormente l'impurità dei nodi: un nodo è puro quando tutti i samples appartenenti alla medesima suddivisione hanno la stessa label. L'indice per misurare l'impurità è un iperparametro e può essere scelto, una possibile soluzione è usare l'indice di Gini, cioè la probabilità che una variabile sia classificata in modo errato. Uno dei vantaggi degli alberi decisionali è che sono semplici da costruire e veloci, inoltre escludono in modo automatico le features non importanti, ha però lo svantaggio di tendere facilmente all'overfitting.

Un altro modello adatto a questo problema è il **support vector machine** (SVM), infatti il SVM ha lo scopo di dividere i sample in classi con un iperpiano di uno spazio n-dimensionale, dove n è il numero delle features. L'iperpiano da scegliere è quello con il massimo margine, cioè la massima distanza tra sample di classi diverse. Un aspetto fondamentale dell'SVM è che se i sample non sono divisibili in modo lineare è possibile applicare una trasformazione non lineare ai dati, per portarli in uno spazio con più dimensioni nel quale sono divisibili in modo lineare, la funzione che effettua questa trasformazione non lineare è detta **kernel** e può essere scelta in base al problema specifico, possibili scelte sono il kernel polinomiale o la radial basis function (RBF). Inoltre non sempre i sample sono divisibili perfettamente ma è necessario commettere degli errori, in questo caso si utilizza il **coefficiente di regolarizzazione** (C), un iperparametro per gestire il bilanciamento tra errori commessi e la grandezza del margine: per C piccolo si dà più importanza al margine, mentre per C grande si cerca di evitare più errori possibile. L'SVM

ha il vantaggio di occupare poca memoria ed essere molto efficace in caso di dimensionalità delle features molto elevata, grazie all'utilizzo dei kernel.

L'ultimo metodo di learning approfondito sono le **reti neurali**. Una rete è composta da diversi *layer*, e ogni layer ha diversi *nodi*. Le features sono messe in input al primo layer, che a sua volta calcolerà una combinazione lineare degli ingressi e poi ne farà una trasformazione non lineare secondo una certa *funzione di attivazione*, dopodiché l'output del primo layer è mandato in input al secondo layer. Questo processo è iterato fino ad arrivare all'ultimo layer, che calcolerà l'output finale. Il numero di layer e il numero di nodi per ogni layer è un iperparametro da scegliere opportunamente, e determina la profondità della rete, inoltre è possibile scegliere la funzione di attivazione da applicare ai dati, in particolare le funzioni cosiddette "S-shaped", come la funzione sigmoide e la funzione tangente iperbolica (tanh) caratterizzano le reti chiamate *multilayer perceptron* (MLP). Un'altra componente fondamentale delle reti neurali sono i pesi che ogni collegamento tra nodi di livelli diversi ha, questi pesi sono i valori che il modello deve imparare durante la fase di training, per farlo le reti neurali usano l'**error backporpagation**: si confronta il valore in uscita con il valore reale atteso, e in base all'errore, cioè alla differenza tra i due, si modificano i pesi propagando all'indietro, partendo dall'ultimo layer, le informazioni per modificare i pesi. Un alto numero di layer e di nodi può portare a overfitting e ad alta complessità, per evitare questo problema si utilizzano delle tecniche di regolarizzazione, come il **weight decay** e l'**early stopping**. Un difetto delle reti neurali è la loro lentezza nel training dovuta proprio all'applicazione dell'algoritmo di error backpropagation, infatti le reti neurali richiedono molte risorse computazionali e l'addestramento risulta quindi lento.

Per questo progetto è stato scelto di utilizzare tre dei modelli descritti:

- Decision Tree,
- Support Vector Machine,
- Reti Neurali.

In particolare per ogni modello saranno studiate tre versioni:

- versione 1: i modelli sono addestrati con i dati grezzi e originali, senza pre-processing,
- versione 2: i modelli sono addestrati con i dati pre-processati,
- versione 3: si utilizza una versione ensemble della versione migliore tra le prime due

4 Dettagli implementativi

In questo paragrafo verranno descritti in modo specifico i dettagli implementativi di ogni modello e di ogni sua versione, oltre ai vari passaggi di pre-processing e il metodo di valutazione.

4.1 Pre-processing

Per le versioni 1 dei modelli i dati non vengono pre-processati, infatti i dati non vengono modificati come descritto nella sezione 2, ma vengono solamente fatte le modifiche in fase di EDA, modificando quindi i tipi dei dati ed eliminando i valori NaN. Lo splitting in questo caso è fatto quindi sui dati originali, togliendo dalle features lo *user*, che è il nome del soggetto e non influisce sull'apprendimento, ed è stata tolta anche la feature *gender*, in quanto non ancora trasformata con l'encoder. Inoltre la variabile target *class* in questo caso rimane sotto forma di stringa.

Per le versioni 2 invece vengono effettuati gli step di pre-processing descritti brevemente nella sezione 2, in particolare dopo aver fatto l'encoding della feature *gender* e del target *class*, è stata studiata la matrice di correlazione, mostrata in figura 7, per valutare la collinearità delle features tra di loro, e con la variabile target. Ne risulta che le variabili *y1* e *y4* sono particolarmente correlate con la variabile target, quindi saranno importanti ai fini della classificazione. In secondo luogo è possibile notare che le features *how_tall_in_meters*, *weight* e *body_mass_index* sono particolarmente correlate tra di loro e di conseguenza è conveniente eliminarne due e mantenerne una sola. Infine le tre variabili che rappresentano le coordinate dell'accelerometro 2, quello sulla

coscia sinistra, sono molto correlate tra loro, quindi anche in questo caso conviene mantenerne una sola.

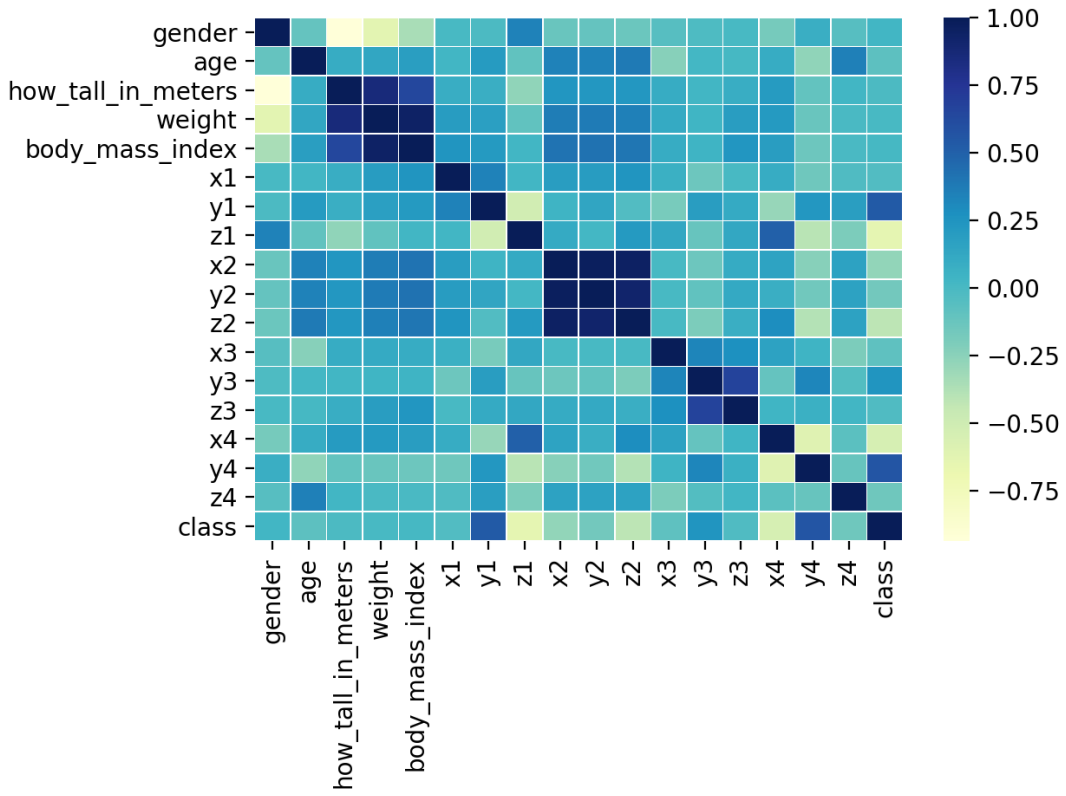


Figure 7: Matrice di correlazione.

Dalle informazioni raccolte dalla matrice di correlazione, in figura 7, effettuiamo la features selection eliminando le variabili *weight* e *body_mass_index* perché altamente correlate con *how_tall_in_meters* e viene mantenuta quest'ultima; vengono anche eliminate le coordinate x e z del secondo accelerometro, mantenendo solo la coordinata y. Infine vengono eliminate le variabili *user* e *age* in quanto non influenti sul riconoscimento della postura.

Le features selezionate vengono poi sottoposte a uno scaling, in particolare si applica il metodo del **min-max-scaling**: trasforma il valore di ogni features portandolo a un valore compreso tra 0 e 1, per farlo applica a ogni variabile la seguente formula:

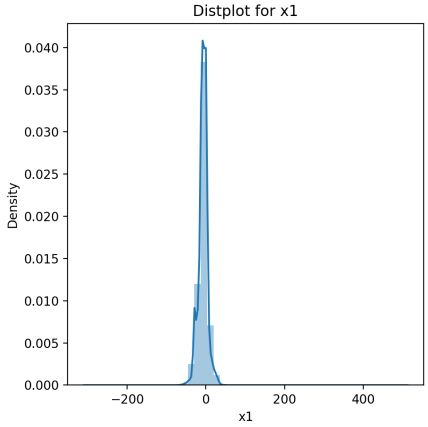
$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

È importante sottolineare che i parametri $min(x)$ e $max(x)$ vanno calcolati sulla base del solo training set, dopodiché si applica la trasformazione al training set e al test set.

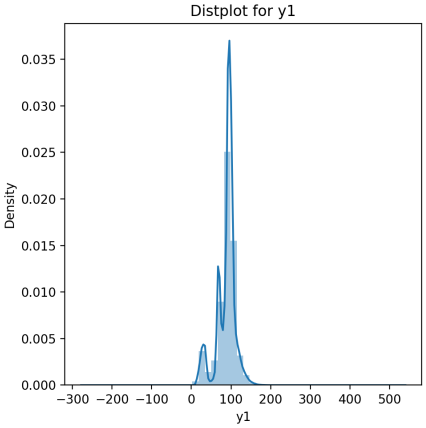
In figura 8 sono mostrati i grafici di distribuzione di alcune variabili prima di aver subito il min-max-scaling, come si vede i valori variano nell'ordine delle centinaia intorno allo zero.

In figura 9 invece sono mostrati i grafici delle distribuzioni delle stesse variabili dopo aver subito la trasformazione del min-max-scaling, come si vede il valore è compreso tra 0 e 1 ma l'andamento è lo stesso e la distribuzione mantiene la stessa forma, a prova del fatto che i dati mantengono le stesse informazioni rappresentative. Grazie a quest'operazione di trasformazione delle features i modelli saranno più efficienti e veloci in fase di training.

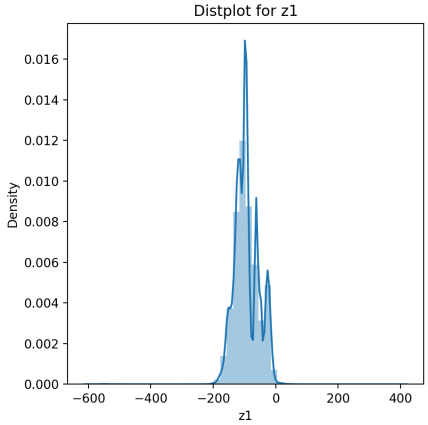
Infine per due dei tre modelli testati è stata fatta un'ulteriore operazione di riduzione delle features, infatti per le seconde e terze versioni dei modelli *SVM* e *rete neurale* è stata applicata la **principal component analysis** (PCA), un metodo che riduce il numero delle features proiettando i punti in un sottospazio più piccolo dimensionalmente rispetto allo spazio originario delle



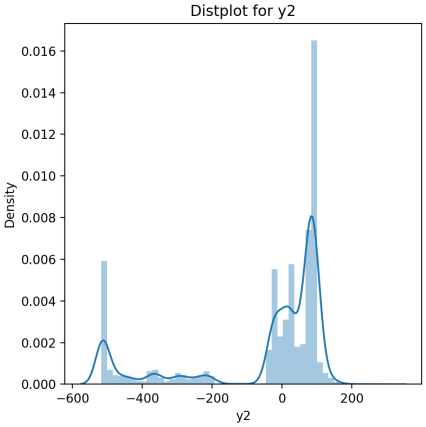
(a) Distribuzione di x1 prima del min-max-scaling.



(b) Distribuzione di y1 prima del min-max-scaling.



(c) Distribuzione di z1 prima del min-max-scaling.

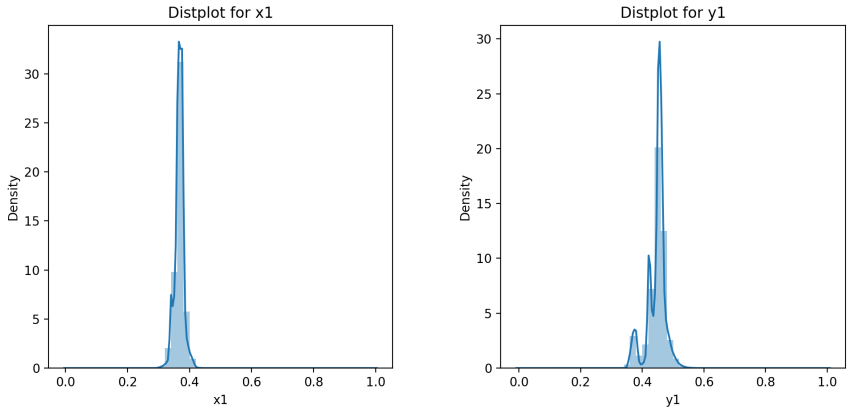


(d) Distribuzione di y2 prima del min-max-scaling.

Figure 8: Distribuzione di variabili prima del min-max-scaling.

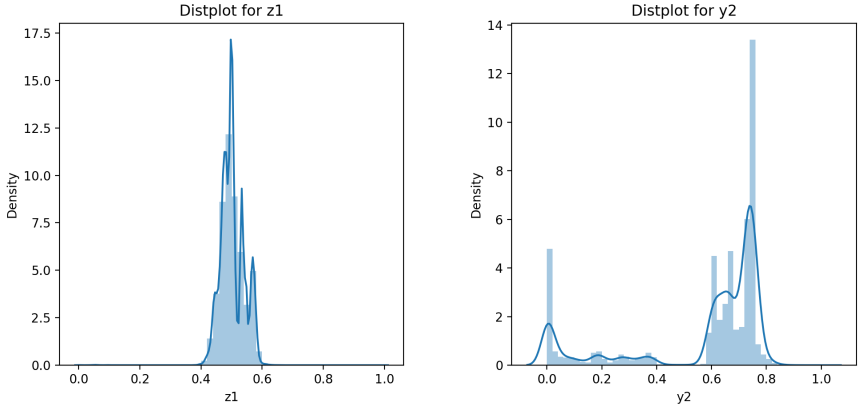
features. Il modello decision tree è più efficiente in termini di prestazioni e velocità, quindi non è stato applicato il metodo PCA a questo algoritmo perché già efficiente con le features selezionate precedentemente. Le SVM e le reti neurali invece essendo più lente nella fase di training sono state velocizzate dalla riduzione delle features fatta dalla PCA.

Per ogni variante del modello è stata fatta una selezione degli iperparametri tramite cross-validation per model selection, per valutare quali iperparametri portano ad un modello migliore, dopodiché è stata applicata la cross-validation per model assessment, cioè per stimare le prestazioni che avrà il modello ricavato dalla model selection. La model selection è stata effettuata usando l'oggetto `GridSearchCV` di *scikit-learn* [2], in particolare applica la strategia di cross-validation **K-fold**: il training set viene diviso in K parti uguali, K-1 parti vengono usate come training set mentre la parte rimanente viene usata come validation set, questo processo viene ripetuto K volte cambiando ogni volta la parte del validation set, infine le prestazioni del modello vengono calcolate come media delle prestazioni nelle K iterazioni. Il parametro K è stato mantenuto quello di default, cioè K = 5. Per la fase di model assessment è stata utilizzata la funzione di *scikit-learn* [2] `cross_validate()`, che divide il dataset passato come parametro in training set e development set, addestra il modello, passato anch'esso come parametro, sul training set e ne valuta le prestazione sul development set in base a certe metriche specificate. In questo caso la valutazione è stata fatta con le metriche **F1-score** e **accuracy bilanciata**.



(a) Distribuzione di x1 dopo il min-max-scaling.

(b) Distribuzione di y1 dopo il min-max-scaling.



(c) Distribuzione di z1 dopo il min-max-scaling.

(d) Distribuzione di y2 dopo il min-max-scaling.

Figure 9: Distribuzione di variabili dopo il min-max-scaling.

4.2 Decision Tree

4.2.1 Prima versione

Per la prima versione dell'algoritmo *Decision tree* è stata fatta la model selection testando i seguenti iperparametri:

- **max_depth**: rappresenta la profondità massima che l'albero decisionale può avere, sono stati testati i valori *5, 10, 20, 50*.
- **class_weight**: è il peso da dare a ogni classe, i valori testati sono *balanced* che assegna un peso a ogni classe sulla base del bilanciamento che questa classe ha nel dataset di riferimento, e il valore *None* che assegna un peso uguale a ogni classe.
- **criterion**: l'indicatore con cui misurare l'impurità dei nodi, i valori testati sono *gini* che usa l'indice di Gini per l'impurità, e il valore *entropy* che utilizza una seconda metodologia per calcolare l'impurità.
- **splitter**: la strategia con cui dividere i nodi, cioè il valore su cui fare il test per ogni nodo, sono stati testati i valori *best*, cioè scegliendo lo splitting migliore, e il valore *random* con il quale invece lo splitting viene scelto casualmente.

Dopo la fase di model selection sono stati ricavati i migliori valori per ogni iperparametro, rappresentati in tabella 1, che portano ad avere un valore di F1-score, usata per la cross-validation, pari a 0.985.

max_depth	*20*
class_weight	*None*
criterion	*entropy*
splitter	*best*

Table 1: Iperparametri migliori per la prima versione del Decision tree.

Con la fase di model assessment si sono ricavate le metriche:

$$F1 - score = 0.985$$

$$accuracy\_bilanciata = 0.974$$

Inoltre per l'algoritmo Decision tree è possibile analizzare l'importanza di ogni features per valutare quali features sono più incisive sulla classificazione, in figura 10 è mostrata la features importance per la prima versione del modello Decision tree, dalla quale si nota che le tre features più importanti sono le coordinate *z1*, *z2* e *y3*, probabilmente queste tre variabili influiscono molto sul riconoscimento della postura rispetto alle altre variabili con importanza minore.

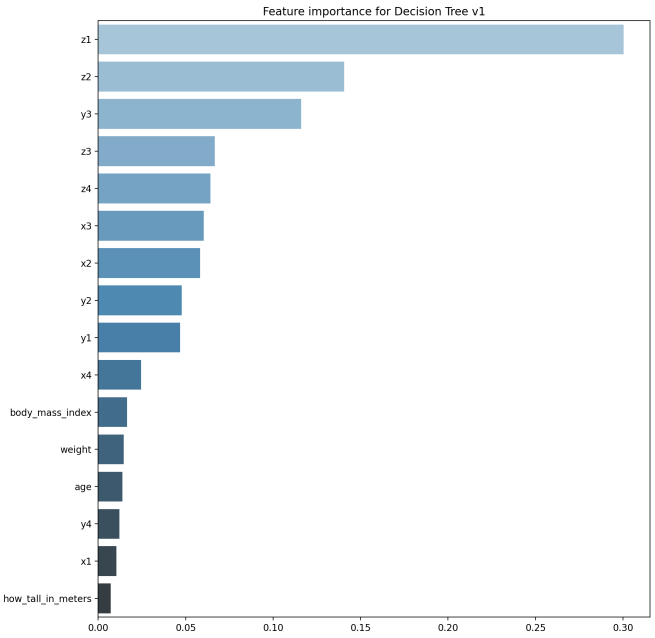


Figure 10: Features importance per la prima versione del modello Decision tree.

4.2.2 Seconda versione

Per la seconda versione vengono utilizzati i dati pre-processati come descritto nella sezione 4.1. Gli iperparametri testati durante la model selection sono gli stessi testati per la prima versione del decsion tree (si veda 4.2.1), con i valori rappresentati in tabella 2, mentre in tabella 3 sono mostrati i valori migliori ricavati dopo la model selection, che portano ad un valore ottimo di F1-score pari a 0.980, in questo caso quindi c'è un leggero degrado delle prestazioni.

Con la fase di model assessment, per la seconda versione del Decision tree, le metriche ricavate sono:

$$F1 - score = 0.980$$

max_depth	*5, 10, 20, 50*
class_weight	*balanced, None*
criterion	*gini, enotrpy*
splitter	*best, random*

Table 2: Valori testati per gli iperparametri della seconda versione del Decision tree.

max_depth	*50*
class_weight	*None*
criterion	*enotrpy*
splitter	*best*

Table 3: Iperparametri migliori per la seconda versione del Decision tree.

$$accuracy\_bilanciata = 0.967$$

Anche la metrica *accuracy_bilanciata* quindi mostra un degrado delle prestazioni rispetto alla prima versione. Infine anche per questa seconda versione si analizza l'importanza delle features, mostrata in figura 11, dalla quale si nota che le variabili *z1* e *y3* rimangono molto importanti come nella prima versione, in questo caso però la variabile *z2* essendo stata eliminata nella fase di features selection non è presente, e assume molta importanza la variabile *y2*.

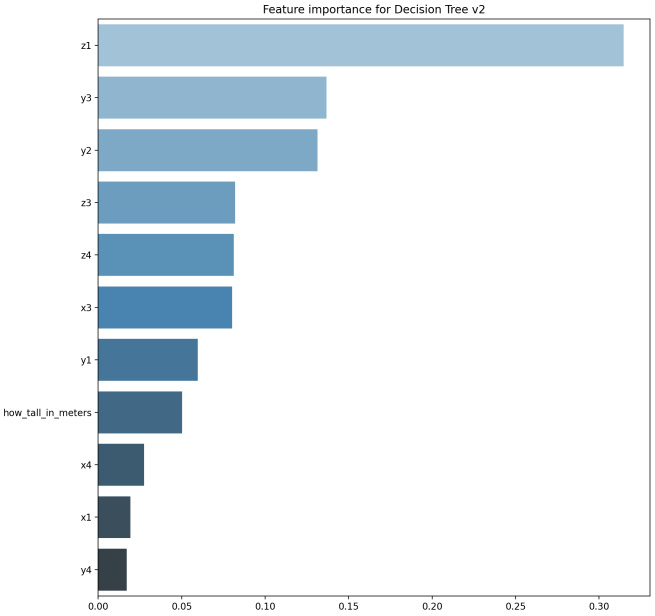


Figure 11: Features importance per la seconda versione del modello Decision tree.

4.2.3 Terza versione

La terza versione è costituita da un modello in ensemble della migliore tra le prime due varianti, in questo caso come visto dalle metriche la versione che garantisce una migliore classificazione è la prima, di conseguenza verrà usata questa per creare il modello in ensemble, e come training set sarà quindi mantenuto quello della prima versione, senza pre-processing.

L'algoritmo sarà costruito usando il metodo del *RandomForest*, che rappresenta appunto una versione ensemble del Decision tree, cioè addestrerà tanti Decision tree e unirà le loro predizioni per ottenere un risultato migliore di quello che raggiungerebbe un singolo decision tree. I valori degli iperparametri in questo caso saranno gli stessi di quelli trovati con la model selection nella prima versione, mentre la model selection per questa versione sarà fatta esclusivamente

sull'iperparametro **n_estimators**, che rappresenta il numero di decision tree che il modello in ensemble dovrà addestrare., in particolare i valori testati sono *5, 10, 20, 50*. In tabella 4 vediamo quindi i valori ottimi degli iperaparametri di questa versione, che portano a un valore ottimo di F1-score = 0.995.

max_depth	*20*
class_weight	*None*
criterion	*enotrpy*
n_estimators	*50*

Table 4: Iperparametri migliori per la terza versione del Decision tree.

Durante la fase di model assessment sono stati ricavati i valori delle metriche di valutazione:

$$F1 - score = 0.995$$

$$accuracy\_bilanciata = 0.992$$

La variante in ensemble garantisce quindi un discreto incremento di prestazioni rispetto a entrambe le versioni precedenti. In figura 12 è mostrata l'importanza delle features per questa terza versione.

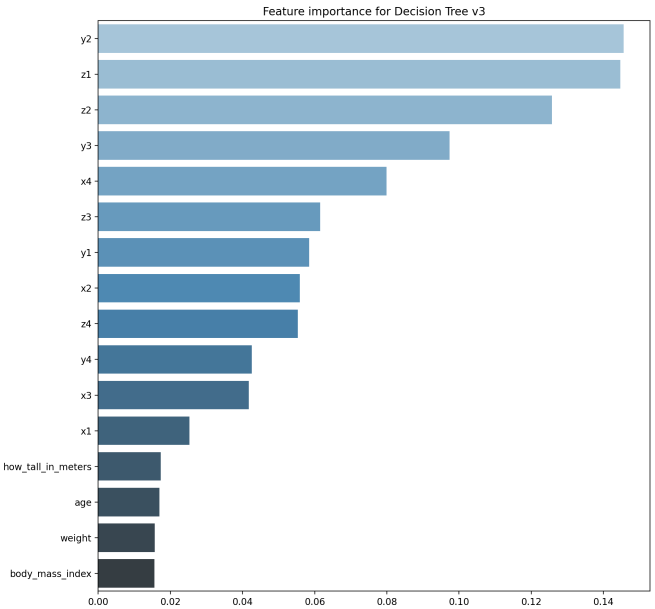


Figure 12: Features importance per la terza verisone del modello Decision tree.

4.3 Support Vector Machine

4.3.1 Prima versione

Per la prima versione del modello *SVM*, durante la model selection, sono stati testati i seguenti iperparametri:

- **C**: è il coefficiente di regolarizzazione che regola il trade-off tra grandezza del margine ed errori commessi dall'algoritmo, i valori testati sono *0.1 e 1000*.
- **kernel**: rappresenta la funzione usata come kernel trick, cioè per mappare le features in uno spazio dimensionalmente diverso, i valori testati sono *linear*, cioè la funzione lineare, *poly* per avere una funzione polinomiale, e *rbf* per usare la radial basis function.

- **class_weight**: come bilanciare le classi, sono stati testati i valori *balanced* per bilanciare le classi nel modo in cui sono bilanciare nel dataset, e *None* per assegnare lo stesso peso a tutte le classi.
- **max_iter**: il numero massimo di iterazioni che l'algoritmo può fare, i valori testati sono *10, 50*.
- **decision_function_shape**: è la metodologia con cui adattare il modello binario per un problema multiclasse, i valori testati sono *ovo*, cioè lo *One-vs-One*, e *ovr*, cioò lo *One-vs-Rest*.

Con la fase di model selection sono stati ricavati i valori migliori per ogni iperaparametro, mostrati in tabella 5, che portano a un valore di F1-score = 0.468.

C	*1000*
kernel	*rbf*
class_weight	*balanced*
max_iter	*10*
decision_function_shape	*ovo*

Table 5: Iperparametri migliori per la prima versione di SVM.

Il model assessment garantisce le prestazioni seguenti:

$$F1 - score = 0.468$$

$$accuracy\_bilanciata = 0.423$$

4.3.2 Seconda versione

La seconda versione della *SVM* utilizza i dati pre-processati e inoltre utilizza la *PCA* tramite una *pipeline*, un modo che mette in sequenza una metodologia di trasformazione dei dati, in questo caso la PCA, e un modello di learning, la SVM. La fase di model selection testerà tutti gli iperparametri testati per la prima versione 4.3.1 per quanto riguarda la SVM, inoltre testerà anche il parametro *n_components*, cioè il numero di componenti della PCA. I valori testati per questo iperparametro sono *1, 5, 10, 20*. I valori ottimi risultanti sono mostrati in tabella 6 e sono relativi a un valore di F1-score = 0.555.

C	*1000*
kernel	*rbf*
class_weight	*None*
max_iter	*50*
decision_function_shape	*ovo*
n_components	*10*

Table 6: Iperparametri seconda versione SVM.

In questo caso si ottengono delle prestazioni leggermente superiori rispetto alla prima versione, seppur non ancora soddisfacenti, infatti i valori delle metriche ottenute dal model assessment sono:

$$F1 - score = 0.567$$

$$accuracy\_bilanciata = 0.530$$

4.3.3 Terza versione

L'algoritmo della terza versione sarà una versione ensemble della seconda versione, in quanto la migliore tra le prime due, di conseguenza il training set usato sarà quello delle seconde versioni, cioè quello con il pre-processing. È stata utilizzata la tecnica di ensemble detta **Bagging**: si

addestrano tanti *weak learner* parallelamente e in modo indipendente e si unisce il risultato di ognuno in modo deterministico. In particolare è stato usato l'oggetto `BaggingClassifier` di *scikit-learn* [2], il quale prende un modello di base e ne addestra un certo numero, ognuno su un sottoinsieme diverso del dataset originario, infine ne combina i vari risultati in un risultato unico. Il parametro *base_estimator* sarà il modello SVM della seconda versione, che sarà il modello base su cui il classificatore di ensemble si baserà. Durante la model selection sarà valutato solamente il parametro *n_estimators* che rappresenta il numero di weak learner da addestrare, i valori testati per questo iperparametro sono *10, 50, 100*. Il valore ottimo ottenuto dalla fase di model selection è 100, e porta ad avere un'F1-score pari a 0.615. Infine la fase di model assessment restituisce le metriche:

$$F1 - score = 0.610$$

$$accuracy\_bilanciata = 0.602$$

Quindi la tecnica di ensemble migliora le prestazioni del modello originale, aumentando sia la F1-score che l'accuracy bilanciata, nonostante ciò la bontà di questo modello non eccelle e i valori rimangono piuttosto bassi.

4.4 Reti neurali

4.4.1 Prima versione

Gli iperparametri testati durante la cross-validation per la prima versione delle reti neurali sono i seguenti:

- **hidden_layer_sizes**: rappresenta il numero di layer che la rete avrà e quanti nodi ogni layer avrà, viene rappresentato sotto forma di *tupla* Python, dove ogni elemento rappresenta un layer e il numero stesso rappresenta il numero di nodi di quel layer. I valori testati sono *(50, 50), (20, 20, 20), (5, 5, 5, 5)*.
- **max_iter**: è il numero massimo di iterazioni che l'algoritmo farà, i valori testati sono *10, 50*.
- **early_stopping**: parametro che indica se adottare o no l'early stopping per evitare l'overfitting, in questo si testano i valori *True, False*.
- **activation**: è la funzione di attivazione dei nodi della rete, sono stati testati i valori *logistic*, che rappresenta la funzione sigmoide, e il valore *tanh*, che rappresenta la funzione tangente iperbolica.

In tabella 7 sono rappresentati i valori ottimi per ciascun iperparametro trovati con la model selection, che portano ad un valore dell'F1-score uguale a 0.980.

hidden_layer_sizes	*(50, 50)*
max_iter	*50*
early_stopping	*False*
max_iter	*50*
activation	*logistic*

Table 7: Iperparametri migliori per la prima versione della rete neurale.

Dalla fase di model assessment si ricavano i valori:

$$F1 - score = 0.980$$

$$accuracy\_bilanciata = 0.965$$

4.4.2 Seconda versione

Per la seconda versione della rete neurale si utilizzano i dati pre-processati e la PCA, nello stesso modo in cui è utilizzata nella seconda versione della SVM (si veda 4.3.2), cioè usando la pipeline tra la PCA e il modello vero e proprio. Gli iperparametri testati durante la model selection sono gli stessi testati nella prima versione della rete neurale e descritti nella sezione 4.4.1 e in più il parametro *n_components* della PCA, testato con i valori *1, 5, 10, 20*. I valori ottimi ottenuti in questo caso sono mostrati nella tabella 8 e portano ad avere una F1-score = 0.990.

hidden_layer_sizes	*(50, 50)*
max_iter	*50*
early_stopping	*False*
max_iter	*50*
activation	*tanh*
n_components	*10*

Table 8: Iperparametri migliori della seconda versione della rete neurale.

La model assessment mostra i valori delle metriche:

$$F1 - score = 0.990$$

$$accuracy\_bilanciata = 0.985$$

La seconda versione ha quindi un miglioramento dal punto di vista delle prestazioni e risulta più precisa.

4.4.3 Terza versione

La terza variante della rete neurale sarà una versione ensemble della seconda versione, dato che quest'ultima è la migliore. Saranno quindi usati i dati pre-processati come training set. Il metodo di ensemble utilizzato in questo caso è lo stesso descritto nella sezione 4.3.3, cioè si usa il *BaggingClassifier* che avrà come modello di base la pipeline relativa alla seconda versione della rete neurale, e verrà fatta la valutazione del solo parametro *n_estimators* con i valori *10, 20, 50*, che avrà come valore ottimo 20, con un valore di F1-score stimato pari a 0.989.

Infine il processo di model assessment garantisce i valori:

$$F1 - score = 0.992$$

$$accuracy\_bilanciata = 0.986$$

Rispetto alla versione due quindi si ha un leggero aumento delle prestazioni grazie alla metodologia di ensemble.

4.5 Metriche di valutazione

Per valutare le prestazioni complessive del modello, dopo le fasi di cross-validation per model selection e model assessment, l'algoritmo viene riaddestrato sull'intero training set iniziale, quello senza pre-processing per le prime versioni e quello pre-processato per le seconde e terze versioni, e infine testato sul test set, di cui è stato fatto lo splitting al 20% all'inizio e mai toccato prima di questa fase finale di testing. La valutazione è stata fatta usando le seguenti metriche:

- **Accuracy bilanciata**, perché come visto nella sezione 2 il dataset iniziale è sbilanciato rispetto alla variabile target, quindi è necessario usare un bilanciamento;
- **Precision**, calcolata come la percentuale di predizioni corrette di una certa classe sul totale delle predizioni con quella label. Essendo un problema multiclasse il valore finale è calcolato come media pesata dei vari valori delle classi;

- **Recall**, calcolata come la percentuale di predizioni corrette di una certa classe rispetto al totale di samples effettivamente appartenenti a quella classe. Anche in questo il valore finale è calcolato come media pesata dei valori delle varie classi;
- **F1-score**, calcolata come

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2}$$

L'analisi delle prestazioni complessive dei vari modelli e dei risultati finali è approfondita nella prossima sezione 5.

5 Risultati e discussione

In questa sezione verranno discussi e confrontati i risultati che i tre modelli nelle varie versioni hanno raggiunto nella fase di testing sul test set.

Dalla tabella 9 si nota che la versione migliore del Decision tree è la terza, quella in ensemble, infatti supera le prime due in tutte le metriche e raggiunge una precisione molto alta.

Versione	**Accuracy bilanciata**	**Precision**	**Recall**	**F1-score**
Versione 1	0.976	0.986	0.986	0.986
Versione 2	0.973	0.984	0.984	0.984
Versione 3	**0.993**	**0.996**	**0.996**	**0.996**

Table 9: Prestazioni delle varianti del Decision tree.

In tabella 10 invece vediamo i risultati delle tre versioni del modello SVM, dai quali si deduce che anche in questo caso la versione migliore è quella in ensemble in quanto migliora di circa il 15-20% i risultati delle prime due versioni, grazie proprio alla combinazione dei risultati di più modelli, cosiddetti weak learner, in un risultato unico. I risultati raggiunti comunque non sono soddisfacenti e non sono paragonabili a quelli che garantisce il Decision tree.

Versione	**Accuracy bilanciata**	**Precision**	**Recall**	**F1-score**
Versione 1	0.420	0.551	0.490	0.482
Versione 2	0.440	0.500	0.452	0.434
Versione 3	**0.610**	**0.675**	**0.655**	**0.620**

Table 10: Prestazioni delle varianti dell'SVM.

In tabella 11 sono confrontati i risultati della rete neurale nelle tre versioni, e come si vede la migliore è anche in questo caso la terza, anche se la differenza con la seconda versione è minima quindi l'incremento di prestazioni dato dall'ensemble è trascurabile in questo caso, seppur non nullo.

Versione	**Accuracy bilanciata**	**Precision**	**Recall**	**F1-score**
Versione 1	0.963	0.981	0.981	0.981
Versione 2	0.986	0.991	0.991	0.991
Versione 3	**0.990**	**0.993**	**0.993**	**0.993**

Table 11: Prestazioni delle varianti della rete neurale.

Infine in tabella 12 sono messi a confronto i tre modelli nelle loro versioni migliori, come si vede l'algoritmo che performa meglio è il Decision tree, anche se la rete neurale ha prestazioni molto simili, solo leggermente inferiori, invece la SVM ha prestazioni nettamente inferiori e quindi è sconsigliata in questo problema.

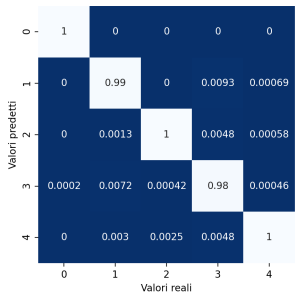
In tutti e tre i modelli la versione migliore è risultata essere la terza, grazie all'incremento di prestazioni garantito dall'ensemble, nel caso però della SVM e della rete neurale questo avviene

Modello	Accuracy bilanciata	Precision	Recall	F1-score
Decision Tree	**0.993**	**0.996**	**0.996**	**0.996**
SVM	0.610	0.675	0.655	0.620
Rete neurale	0.990	0.993	0.993	0.993

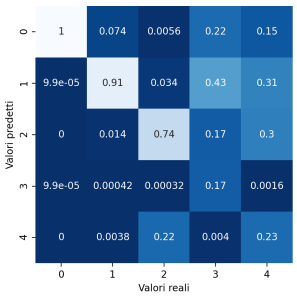
Table 12: Confronto tra le migliori versioni dei tre modelli analizzati.

a un costo di efficienza in termini di velocità, infatti applicando il metodo Bagging il processo di training è rallentato considerevolmente rispetto alle prime due versioni. Questo rallentamento non avviene nel caso del Decision tree, infatti il RandomForest ha tempi paragonabili alla versione non in ensemble.

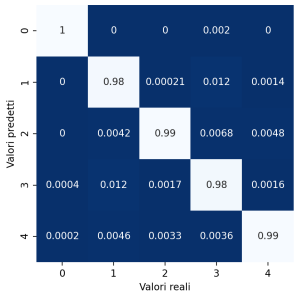
In figura 13 sono rappresentate le tre confusion matrix delle versioni migliori dei tre modelli, i valori sono normalizzati in questo caso, quindi non sono mostrati i valori assoluti ma la percentuale sul totale. Come si vede le matrici del Decision tree e della rete neurale hanno valori prossimi a uno sulla diagonale principale, il che significa che i valori predetti per una classe sono quasi tutti corretti. Al contrario la matrice del modello SVM ha valori minori nella diagonale principale, a dimostrazione del fatto che le prestazioni di questo algoritmo non sono eccellenti. In particolare l'algoritmo ha pessime prestazioni con le classi *3* e *4*, mentre sulle classi *0, 1 e 2* ha valori paragonabili a quelli ottenuti dal Decision tree e dalla rete neurale.



(a) Confusion matrix della terza versione del Decision tree.



(b) Confusion matrix della terza versione dell'SVM.



(c) Confusion matrix della terza versione della rete neurale.

Figure 13: Confusion matrix dei tre modelli migliori.

Per questo progetto sia la SVM che la rete neurale sono state testate con dei valori per gli iperparametri non ottimali, infatti per questioni di tempo e hardware a disposizione è stato deciso di non usare valori troppo alti per alcuni degli iperparametri, che probabilmente avrebbero portato a risultati migliori ma a discapito del tempo di training e di model selection, tenendo conto anche del fatto che il dataset usato contiene un notevole numero di sample, anche questo contribuisce ad allungare i tempi di addestramento. Nonostante ciò comunque la rete neurale è riuscita a performare molto bene avvicinandosi molto all'algoritmo che è risultato migliore, il

Decision tree; la SVM invece non ha raggiunto questi risultati.

Per concludere, quindi, per tutti i modelli testati la versione migliore risulta essere la terza, cioè quella in ensemble, anche se per la rete neurale il miglioramento è molto piccolo e visto il peggioramento che il metodo di ensemble introduce in termini di velocità potrebbe essere conveniente utilizzare la versione numero due, in quanto ha tempi di addestramento minori e prestazioni molto simili. Tra i tre modelli infine il più prestante risulta essere il Decision tree in ensemble, quindi nello specifico il RandomForest, con gli iperparametri descritti nella tabella 4.

References

[1] Ugulino W.; Cardador D.; Vega K.; Velloso E.; Milidiu R.; Fuks H. *Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements*. URL: `http://groupware.les.inf.puc-rio.br/work.jsf?p1=10335`.

[2] Scikit-learn. *Scikit-learn*. URL: `https://scikit-learn.org/stable/`.