Human Activity Recognition

Francesco Vannoni

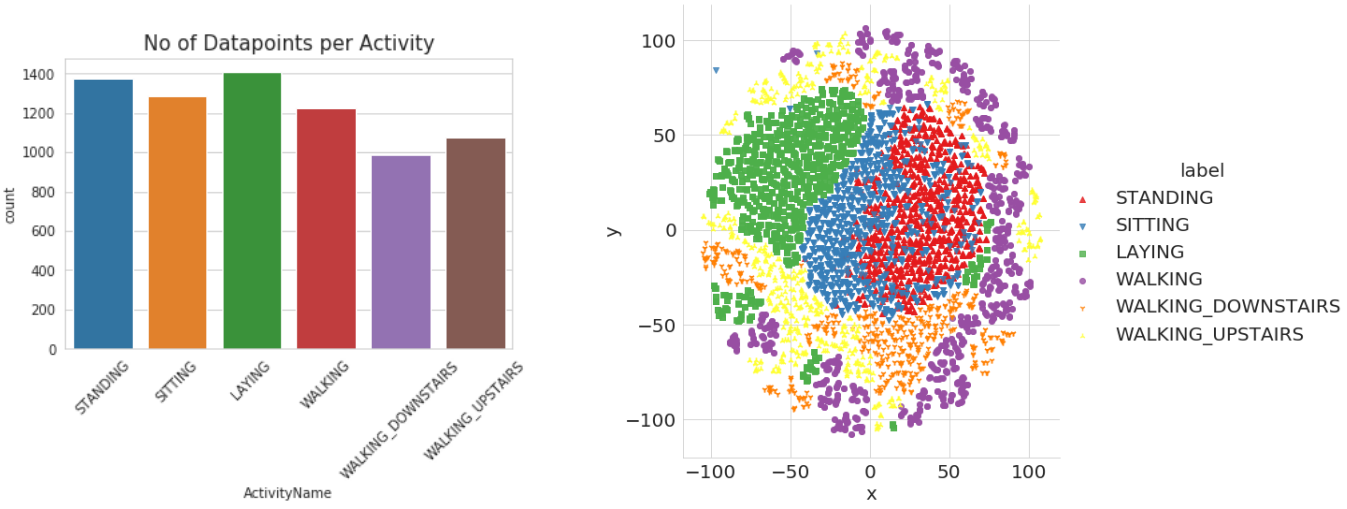
1 Introduzione

"A decision tree represents a function that takes as input a vector of attribute values and returns a "decision" —a single output value. The input and output values can be discrete or continuous." (Russel et al. 2010)
In questo lavoro si utilizza il modello decision tree (tramite scikit-learn (Pedregosa et al.2011) in Python) al fine di riconoscere sei attività umane (*walking*, *standing*, *sitting*, *laying*, *walking*, *walking upstairs*, *walking downstairs*). Si utilizza un dataset UCI, le modalità con cui sono rilevati i segnali utilizzati per il lavoro sono presenti in Anguita et al. 2012. I dati sono rilevati da 30 soggetti attraverso il giroscopio e l'accelerometro di uno smartphone.

2 Data pre-processing

Dopo aver importato il dataset tramite la libreria pandas (McKinney 2010) si analizza quest'ultimo prima di procedere con il processamento dei dati. Il dataset è già diviso in train e test, il 70% è usato per il train set e il rimanente per il test set. Tramite le funzioni *isnull()* e *duplicated()* di pandas si registra che non presenta valori duplicati o NaN/Null. Di seguito si mostrano inoltre due grafici:

- Figura 1.a mostra il "bilanciamento" dei dati; il dataset è piuttosto ben bilanciato, non c'è un'attività rilevata un numero di volte nettamente maggiore/minore rispetto alle altre.
- Figura 1.b mostra un grafico di dispersione realizzato tramite la funzione della libreria sklearn "sklearn.manifold.TSNE()" che permette di visualizzare in due dimensioni un dataset di alta-dimensione (maggiori dettagli riguardo la funzione in Maate et al. 2008)



(a) (b)

Figura 1: (a) bilanciamento dei dati, (b) grafico di dispersione

3 Implementazione modello (Decision Tree)

Il fine di questo lavoro è utilizzare un'implementazione di decision tree per riconoscere attività umane e produrre come risultato una matrice di confusione. Si è cercato di ottimizzare il modello sfruttando due tecniche: "cross validation" e "cost complexity pruning".

Il modo in cui queste agiscono è presentato nei paragrafi seguenti. L'idea generale è quella di modellare l'albero in modo da evitare overfitting e underfitting. Il modello è underfitting quando non è capace di catturare la relazione tra gli esempi in input e i valori target, in questo caso si hanno delle scarse performance sul training data. Al contrario un modello overfitting ha ottime performance sul training data ma non sul testing data. Di seguito si riporta una visualizzazione di un sottoalbero di altezza 4 ottenuto con il dataset in questione.

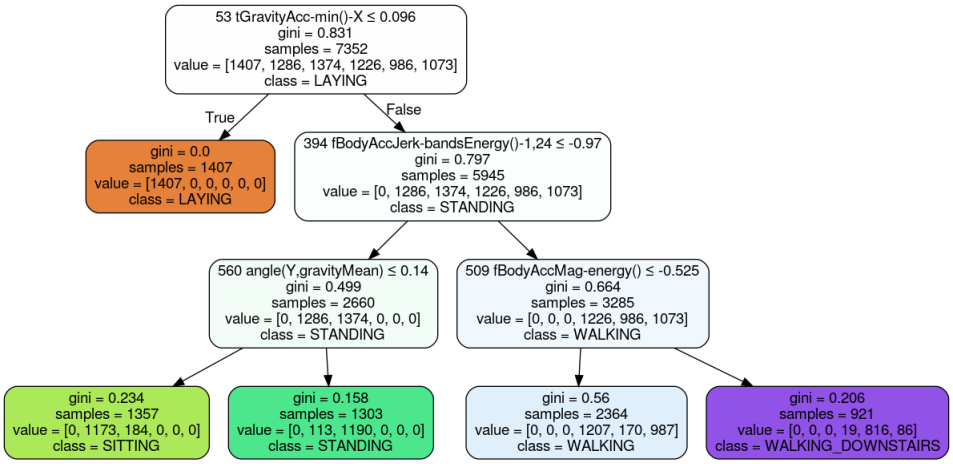
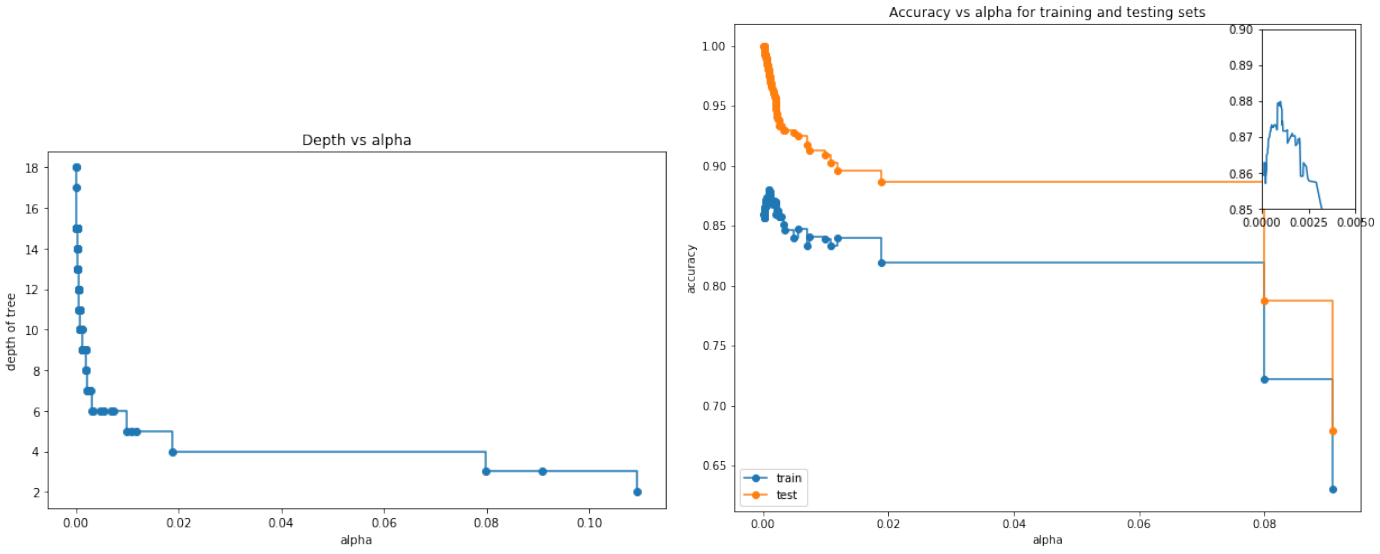


Figura 2: decision tree

3.1 Cost complexity pruning

Minimal cost complexity pruning è un algoritmo che incrementa il numero di nodi tagliati all'aumentare di un parametro che viene nominato alpha. Si basa sulla ricerca ricorsiva del "weakest link" che dunque verrà eliminato. L'algoritmo è descritto nel capitolo 3 di Breiman et al. 1984.
Di seguito è riportato un grafico(Figura 3.a) che mostra l'andamento della profondità dell'albero al variare del valore di *ccp_alpha*. Come prima cosa si deve capire quali possono essere i valori appropriati per ccp_alpha. Per farlo si usa la funzione della libreria sklearn *DecisionTreeClassifier.cost_complexity_pruning_path* la quale ritorna un dizionario formato dagli effettivi valori di alpha per ogni sottoalbero e la somma delle impurità delle foglie del sottoalbero per il corrispondente valore di alpha.
Conseguentemente si addestra il decision tree usando tali valori di alfa.



(a) (b)

Figura 3: (a) profondità al variare di alpha, (b) accuratezza per training e test set al variare di alpha

La decrescita è molto rapida per valori piccoli di alpha (compresi fra 0.00 e 0.02) mentre tende ad rallentare dopo questo intervallo.
Da Figura 3.b si nota invece che quando ccp_alpha è settato a 0 si ha overfit, infatti il training set ha un'accuratezza del 100% ma questo non corrisponde al massimo valore del testing set. Si nota dal grafico che nel punto in cui l'accuratezza per il testing set è massima rimane comunque molto alta l'accuratezza del training set(sicuramente maggiore di 0.9 non comparendo nell'ingrandimento posto in alto a destra). Il valore di ccp_alpha per cui si raggiunge la massima accuratezza nel testing set risulta essere 0.000979.

3.2 Cross Validation

Una diversa soluzione per evitare l'overfitting è dividere il dataset non più in due set (train set e test set) ma in tre set: *train set, validation set, e test set.* Il dataset è stato diviso per questo studio in maniera diversa, in particolare il test set non è costituito più dal 30% del dataset ma dal 20% mentre il rimanente è lasciato a train e validation. Nel validation set si sperimentano alcune varianti del decision tree agendo sui suoi parametri. In particolare si è usato la 5-fold cross validation che consiste nella suddivisione del training set in 5 parti di uguale numerosità e, ad ogni passo, la k-esima parte del dataset è il validation set mentre la restante costituisce il training set. Per farlo si è usato il modulo di scikit-learn GridSearchCV che prende in ingresso una griglia di parametri e come risultato si ottiene la combinazione di questi per i quali è massima l'accuratezza sul validation set. I parametri su cui si è lavorato sono quattro, in parentesi l'intervallo di valori testati per ciascun parametro:

- *criterion*: funzione per misurare la qualità di uno split. [”gini”, ”entropy”].
- *max_depth*: massima profondità dell'albero. [4 - 17]
- *min_samples_leaf*: minimo numero di ”samples” richiesti per fare lo split di un nodo [2 - 14]
- *min_samples_split*: minimo numero di ”samples” richiesti per essere una foglia [2 - 14]

La migliore combinazione di parametri risulta essere: *criterion = 'entropy', 'max_depth'= 13, 'min_samples_leaf'= 2, 'min_samples_split'= 14.*

4 Risultati

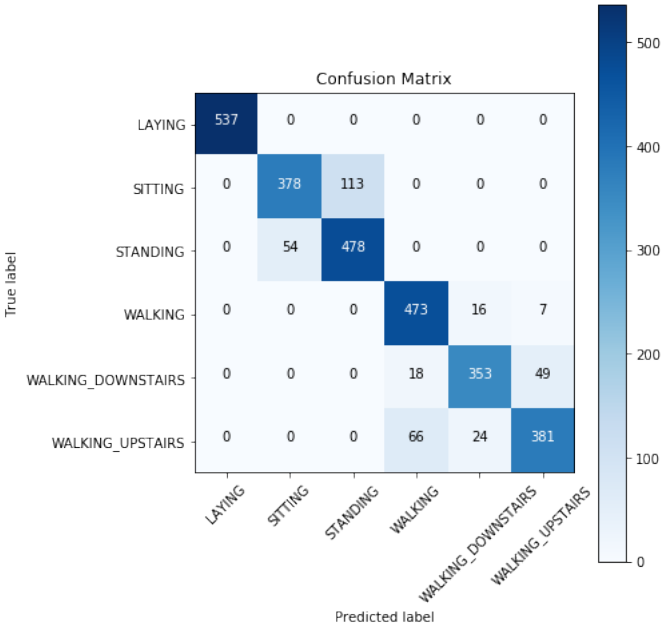
Figura 4 mostra i risultati in termini di accuratezza per ciascuna tecnica. La massima accuratezza si ottiene nel caso CV ma si deve considerare il fatto che è calcolata su un test set ridotto rispetto a quello di CCP e NO-Pruning. Infatti come visto in Sez. 3.2 in questo caso si è lavorato con un test set di dimensione 20% rispetto al dataset. Di seguito sono riportate matrice di confusione e classification report per CCP(Figra 5.a) e CV(figura 5.b)

Parameters	NO-Pruning	CCP	CV*
ccp_alpha	0.00	0.000979	0.00
criterion	gini	gini	entropy
depth	18	18	13
min_samples_leaf	1	1	2
min_samples_split	2	2	14
ACCURACY	0.861	0.881	0.941*

Figura 4: Risultati

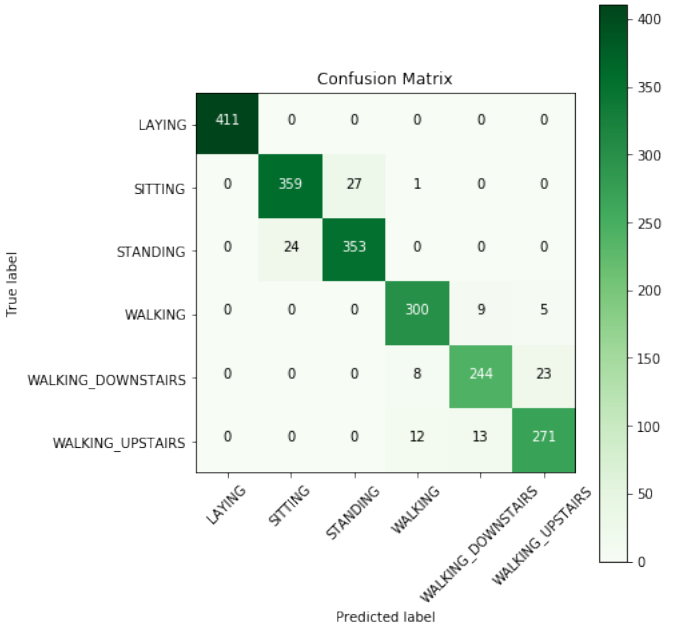
Matrice di confusione e classification report sono prodotti rispettivamente con *confusion_matrix()* e *classification_report()* di sklearn.metrics.

In entrambi i casi non ci errori nel riconoscere laying. Ha senso il fatto che siano "confuse" in piccola percentuale le attività di standing e sitting trattandosi di due attività entrambe statiche. Ciò si poteva anche pronosticare dal grafo di dispersione (Figura 1.b) dove si hanno dei samples corrispondenti al sitting (in blu) nella parte rossa corrispondente a standing e viceversa. Allo stesso modo si può commentare le attività dinamiche ("walking, waking upstairs e walking downstairs"). Di seguito alla matrice di confusione si riporta il classification report con precision, recall, f1-score, e support per ciascuna attività.



Classification Report				
	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537
SITTING	0.87	0.77	0.82	491
STANDING	0.81	0.90	0.85	532
WALKING	0.84	0.95	0.89	496
WALKING_DOWNSTAIRS	0.90	0.84	0.87	420
WALKING_UPSTAIRS	0.88	0.80	0.84	471
accuracy			0.88	2947
macro avg	0.88	0.88	0.88	2947
weighted avg	0.88	0.88	0.88	2947

(a)



Classification Report				
	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	411
SITTING	0.94	0.93	0.93	387
STANDING	0.93	0.94	0.93	377
WALKING	0.93	0.96	0.94	314
WALKING_DOWNSTAIRS	0.92	0.89	0.90	275
WALKING_UPSTAIRS	0.91	0.92	0.91	296
accuracy			0.94	2060
macro avg	0.94	0.94	0.94	2060
weighted avg	0.94	0.94	0.94	2060

(b)

Figura 5: matrice di confusione e classification report, (a) CCP (b) CV

5 References

D. Anguita, A.Ghio, L.Oneto, X.Parra and J.L.Reyes-Ortiz. Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine. 2012

L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.

L.v.d. Maaten, G. Hinton. Visualizing data using t-SNE , L.v.d. Maaten, G. Hinton. Journal of Machine Learning Research, Vol 9(Nov), pp. 2579—2605. 2008.

Pedregosa et al. Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Ap- proach. 3rd edition. Pearson, 2010.

Wes McKinney. Data Structured for Statistical Computing in Phyton. pp.51-56. 2010