Human Activity Recognition Using Smartphones Data Set

Ing. Rosario Moscato, 20 gennaio 2020

Partendo dai dati riguardanti le attività giornaliere svolte abitualmente dalla maggior parte persone, raccolti attraverso l'**accelerometro** e il **giroscopio** di un comune **Smartphone** Android (un Samsung Galaxy SII) e pubblicati da *Smartlab*, si dimostrerà come sia possibile mediante l'impiego di modelli di AI (in questo caso di una **Rete Neurale Artificiale**) individuare con ottima accuratezza cosa gli utilizzatori di smartphone stiano facendo. Le sei attività prese in considerazione sono: camminare, salire le scale, scendere le scale, stare seduti, stare in piedi e stare sdraiati (**Walking, Walking Upstairs, Walking Downstairs, Sitting, Laying down**). Qui il video della fase di raccolta dei dati (https://www.youtube.com/watch?v=XOEN9W05_4A)

In [1]:
```
# Importiamo Pandas
import pandas as pd
```

In [2]:
```
# Carichiamo i dati di Training
train_x = pd.read_csv('C:/Users/rosar/Desktop/Human Activity Recognition Usi
ng Smartphones/UCI HAR Dataset/train/X_train.txt', sep='\s+', header=None)
```

In [3]:
```
# Dimensione dei dati (7352 casi e 561 proprietà)
train_x.shape
```

Out[3]: (7352, 561)

In [4]:
```
# Carichiamo le Labels dei dati di Training
train_y = pd.read_csv('C:/Users/rosar/Desktop/Human Activity Recognition Usi
ng Smartphones/UCI HAR Dataset/train/y_train.txt', sep='\s+', header=None, n
ames=['classe'])
```

In [5]:
```
# Dimensione delle Labels
train_y.shape
```

Out[5]: (7352, 1)

In [6]:
```
# Mettiamo insieme dati e Labels
Dati_Training = pd.concat([train_x, train_y], axis=1).reset_index(drop=True)
```

In [7]:
```
# Dimensione totale dei dati di Training
Dati_Training.shape
```

Out[7]: (7352, 562)

In [8]:
```
# Vediamo i dati
Dati_Training.head()
```

Out[8]:

	0	1	2	3	4	5	6	7	8	
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.9347
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.9430
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.9386
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.9386
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.9424

5 rows × 562 columns

In [9]:
```
# Vediamo i tipi dei dati
Dati_Training.dtypes
```

Out[9]:
```
0          float64
1          float64
2          float64
3          float64
4          float64
            ...
557        float64
558        float64
559        float64
560        float64
classe       int64
Length: 562, dtype: object
```

In [10]:
```
# Le labels valgono 1,2,3,4,5,6 facciamole diventare 0,1,2,3,4,5
Dati_Training['classe']=Dati_Training['classe']-1
```

In [11]:
```
# Controlliamo
Dati_Training.head()
```

Out[11]:

	0	1	2	3	4	5	6	7	8	
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.9347
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.9430
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.9386
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.9386
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.9424

5 rows × 562 columns

In [12]:
```
# Stampiamo le informazioni sui dati
Dati_Training.describe()
```

Out[12]:

	0	1	2	3	4	5	6	
count	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.(
mean	0.274488	-0.017695	-0.109141	-0.605438	-0.510938	-0.604754	-0.630512	-0.5
std	0.070261	0.040811	0.056635	0.448734	0.502645	0.418687	0.424073	0.4
min	-1.000000	-1.000000	-1.000000	-1.000000	-0.999873	-1.000000	-1.000000	-1.(
25%	0.262975	-0.024863	-0.120993	-0.992754	-0.978129	-0.980233	-0.993591	-0.9
50%	0.277193	-0.017219	-0.108676	-0.946196	-0.851897	-0.859365	-0.950709	-0.8
75%	0.288461	-0.010783	-0.097794	-0.242813	-0.034231	-0.262415	-0.292680	-0.(
max	1.000000	1.000000	1.000000	1.000000	0.916238	1.000000	1.000000	0.9

8 rows × 562 columns

I dati sono standardizzati (assumono valori tra -1 e 1, media 0 e deviazione standard 1)

In [13]:
```
# Facciamo lo splitting dei dati
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(Dati_Training.drop(['cla
sse'], axis=1),
                                                    Dati_Training['classe'],
                                                    test_size=0.2,
                                                    random_state=0)

X_train.shape, X_test.shape
```

Out[13]:
```
((5881, 561), (1471, 561))
```

In [14]:
```
# Verifichiamo che le labels assumano valori da 0 a 5
y_train.unique()
```

Out[14]:
```
array([3, 0, 2, 4, 5, 1], dtype=int64)
```

In [15]:
```
# Labels Encoding
from keras.utils import to_categorical

num_classes=6

y_train_dummy = to_categorical(y_train, num_classes)
y_test_dummy = to_categorical(y_test, num_classes)
```

```
Using TensorFlow backend.
```

In [16]:
```
# Definiamo il Modello di Rete Neurale Artificiale (un Input Layer con 561 p
roprietà in ingresso, un Hidden Layer di 284
# Nodi, un altro Hidden Layer di 142 Nodi e un Output Layer che restituisce
6 possibili valori)
from keras.models import Sequential
from keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(284, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.3))
model.add(Dense(142, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

In [17]:
```
# Vediamo il Modello Creato
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 284)               159608    
_________________________________________________________________
dropout_1 (Dropout)          (None, 284)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 142)               40470     
_________________________________________________________________
dropout_2 (Dropout)          (None, 142)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 858       
=================================================================
Total params: 200,936
Trainable params: 200,936
Non-trainable params: 0
_________________________________________________________________
```

Il Modello è composto da 200936 parametri da addestrare.

In [18]:
```
# Compiliamo il Modello
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['a
ccuracy'])
```

In [19]:
```
# Addestramento del Modello
from keras.callbacks import EarlyStopping

earlyStopping = EarlyStopping(min_delta=0.001, patience=5, restore_best_weig
hts=True)

model.fit(X_train, y_train_dummy, batch_size=256, validation_split=0.2, epoc
hs=100, callbacks=[earlyStopping])
```

```
Train on 4704 samples, validate on 1177 samples
Epoch 1/100
4704/4704 [==============================] - 1s 207us/step - loss: 1.0511 - a
ccuracy: 0.5599 - val_loss: 0.5138 - val_accuracy: 0.8216
Epoch 2/100
4704/4704 [==============================] - 0s 68us/step - loss: 0.4865 - ac
curacy: 0.8023 - val_loss: 0.2992 - val_accuracy: 0.8912
Epoch 3/100
4704/4704 [==============================] - 0s 71us/step - loss: 0.3115 - ac
curacy: 0.8773 - val_loss: 0.2194 - val_accuracy: 0.9125
Epoch 4/100
4704/4704 [==============================] - 0s 70us/step - loss: 0.2426 - ac
curacy: 0.9060 - val_loss: 0.1891 - val_accuracy: 0.9184
Epoch 5/100
4704/4704 [==============================] - 0s 74us/step - loss: 0.2062 - ac
curacy: 0.9182 - val_loss: 0.1591 - val_accuracy: 0.9320
Epoch 6/100
4704/4704 [==============================] - 0s 72us/step - loss: 0.1798 - ac
curacy: 0.9296 - val_loss: 0.1406 - val_accuracy: 0.9439
Epoch 7/100
4704/4704 [==============================] - 0s 69us/step - loss: 0.1534 - ac
curacy: 0.9403 - val_loss: 0.1342 - val_accuracy: 0.9473
Epoch 8/100
4704/4704 [==============================] - 0s 72us/step - loss: 0.1413 - ac
curacy: 0.9473 - val_loss: 0.1035 - val_accuracy: 0.9592
Epoch 9/100
4704/4704 [==============================] - 0s 71us/step - loss: 0.1211 - ac
curacy: 0.9534 - val_loss: 0.1244 - val_accuracy: 0.9439
Epoch 10/100
4704/4704 [==============================] - 0s 70us/step - loss: 0.1180 - ac
curacy: 0.9537 - val_loss: 0.1205 - val_accuracy: 0.9431
Epoch 11/100
4704/4704 [==============================] - 0s 69us/step - loss: 0.1073 - ac
curacy: 0.9594 - val_loss: 0.0969 - val_accuracy: 0.9643
Epoch 12/100
4704/4704 [==============================] - 0s 68us/step - loss: 0.1160 - ac
curacy: 0.9573 - val_loss: 0.1032 - val_accuracy: 0.9575
Epoch 13/100
4704/4704 [==============================] - 0s 69us/step - loss: 0.0879 - ac
curacy: 0.9683 - val_loss: 0.0759 - val_accuracy: 0.9711
Epoch 14/100
4704/4704 [==============================] - 0s 69us/step - loss: 0.0761 - ac
curacy: 0.9700 - val_loss: 0.0750 - val_accuracy: 0.9728
Epoch 15/100
4704/4704 [==============================] - 0s 71us/step - loss: 0.0740 - ac
curacy: 0.9724 - val_loss: 0.0839 - val_accuracy: 0.9669
Epoch 16/100
4704/4704 [==============================] - 0s 71us/step - loss: 0.0797 - ac
curacy: 0.9690 - val_loss: 0.0675 - val_accuracy: 0.9754
Epoch 17/100
4704/4704 [==============================] - 0s 72us/step - loss: 0.0696 - ac
curacy: 0.9739 - val_loss: 0.0642 - val_accuracy: 0.9779
Epoch 18/100
4704/4704 [==============================] - 0s 69us/step - loss: 0.0646 - ac
curacy: 0.9751 - val_loss: 0.0627 - val_accuracy: 0.9788
Epoch 19/100
4704/4704 [==============================] - 0s 77us/step - loss: 0.0651 - ac
curacy: 0.9745 - val_loss: 0.0778 - val_accuracy: 0.9720
Epoch 20/100
4704/4704 [==============================] - 0s 78us/step - loss: 0.0729 - ac
curacy: 0.9713 - val_loss: 0.0740 - val_accuracy: 0.9711
Epoch 21/100
4704/4704 [==============================] - 0s 77us/step - loss: 0.0869 - ac
curacy: 0.9653 - val_loss: 0.0572 - val_accuracy: 0.9830
Epoch 22/100
4704/4704 [==============================] - ETA: 0s - loss: 0.0561 - accurac
y: 0.97 - 0s 76us/step - loss: 0.0561 - accuracy: 0.9792 - val_loss: 0.0597 -
val_accuracy: 0.9762
```

Out[19]: <keras.callbacks.callbacks.History at 0x2296a2e55c8>

L'Addestramento si è arrestato alla 26a epoca con un valore di accuracy di circa il 98%

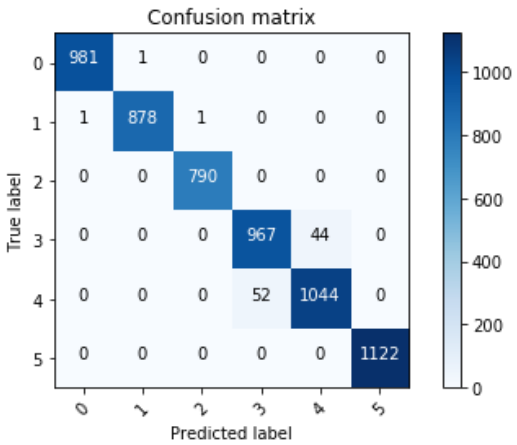
In [20]:
```
# Verifichiamo i valori sul Test Set
model.evaluate(X_test, y_test_dummy)
```

```
1471/1471 [==============================] - 0s 78us/step
```

Out[20]: [0.06824743313895935, 0.9762066602706909]

In [21]:
```
# Anche sul Test Set l'accuracy è pari al 97%, stampiamo la Confusion Matrix
from sklearn.metrics import confusion_matrix
from viz import plot_confusion_matrix
%matplotlib inline

y_pred = model.predict_classes(X_train)
cm = confusion_matrix(y_train, y_pred)
plot_confusion_matrix(cm,['0', '1', '2','3','4','5'])
```



Dati di Test

In [22]:
```
# Carico i Dati di Test
dati_test = pd.read_csv('C:/Users/rosar/Desktop/Human Activity Recognition Using Smartphones/UCI HAR Dataset/test/X_test.txt', sep='\s+', header=None)
```

In [23]:
```
# Dimensione dei dati (2947 casi e 561 proprietà)
dati_test.shape
```

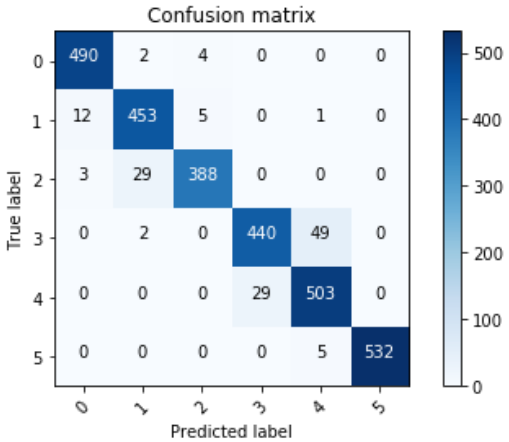
Out[23]: (2947, 561)

In [24]:
```
# Carico le Labels dei dati di Test
risultati_test = pd.read_csv('C:/Users/rosar/Desktop/Human Activity Recognition Using Smartphones/UCI HAR Dataset/test/y_test.txt', sep='\s+', header=None)
```

In [25]:
```
# Le labels valgono 1,2,3,4,5,6 facciamole diventare 0,1,2,3,4,5
risultati_test=risultati_test-1
```

In [26]:
```
# Eseguiamo la predizione delle attività umane utilizzando il modello di ANN addestrato in precedenza
test_predict = model.predict_classes(dati_test)
```

In [27]:
```
# Confrontiamo le previsioni con i dati reali e stampiamo la Confusion Matrix del confronto
cm = confusion_matrix(risultati_test, test_predict)
plot_confusion_matrix(cm,['0', '1', '2','3','4','5'])
```



Come si vede abbiamo 2806 previsioni corrette a fronte di 141 errori con una accuracy che quindi è pari a 95,4% circa

In [28]:
```
# Dizionario degli stati
stati = {0:'WALKING',1:'WALKING_UPSTAIRS',2:'WALKING_DOWNSTAIRS',3:'SITTING',4:'STANDING',5:'LAYING'}
```

In [29]:
```
import numpy as np
risultati_test_array=np.array(risultati_test)
```

In [30]:
```
# Verifichiamo quanti errori ci sono in 50 predizioni
print("Lista degli stati:")
for val in stati:
    print(val, stati[val])
print()

for i in range(1,50):
    test_predict = model.predict_classes(dati_test)[i*50]
    valore_reale = risultati_test_array[i*50,0]
    if test_predict==valore_reale:
        print("Stato Predetto:", test_predict, "\tStato Reale:", valore_real
e)
    else:
        print("Stato Predetto:", test_predict, "\tStato Reale:", valore_real
e, "\t---ERRORE---")
```

```
Lista degli stati:
0 WALKING
1 WALKING_UPSTAIRS
2 WALKING_DOWNSTAIRS
3 SITTING
4 STANDING
5 LAYING

Stato Predetto: 4       Stato Reale: 3  ---ERRORE---
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 2       Stato Reale: 2
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 2       Stato Reale: 2
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 4       Stato Reale: 3  ---ERRORE---
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 5       Stato Reale: 5
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 5       Stato Reale: 5
Stato Predetto: 2       Stato Reale: 2
Stato Predetto: 4       Stato Reale: 4
Stato Predetto: 5       Stato Reale: 5
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 4       Stato Reale: 4
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 2       Stato Reale: 2
Stato Predetto: 4       Stato Reale: 4
Stato Predetto: 5       Stato Reale: 5
Stato Predetto: 0       Stato Reale: 0
Stato Predetto: 1       Stato Reale: 1
Stato Predetto: 3       Stato Reale: 3
Stato Predetto: 5       Stato Reale: 5
Stato Predetto: 2       Stato Reale: 2
Stato Predetto: 4       Stato Reale: 4
Stato Predetto: 3       Stato Reale: 3
```

2 errori in 50 predizioni confermano il dato sull'accuratezza.

In [ ]: