```
%%capture
!pip install everywhereml
!apt install xxd
import pprint
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf

from numpy import array
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras import backend
from tensorflow.keras.models import load_model
```

Fase 0: Caricamento Dataset

Il dataset è costituito da una raccolta dati in formato CSV.

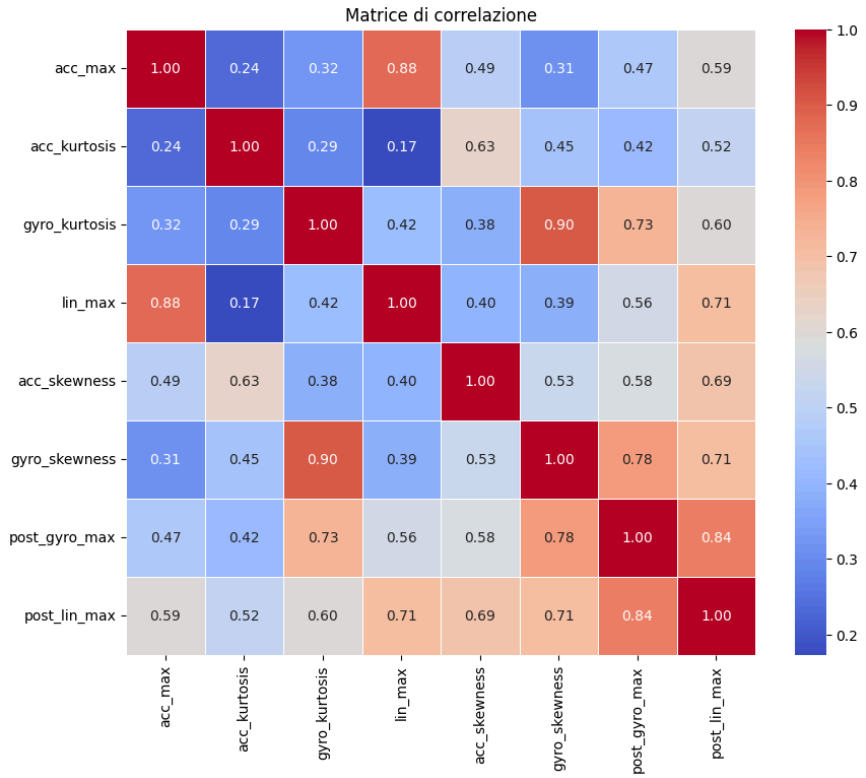
```
# Caricamento dataset con heading incluso
dataset = pd.read_csv("dataset.csv")
# Eseguiamo uno shuffle sul dataset rimuovendo l'indicizzazione
dataset = dataset.sample(frac=1, random_state=1999).reset_index(drop=True)
# Stampiamo le prime righe del dataset
dataset.head()
```

	acc_max	acc_kurtosis	gyro_kurtosis	lin_max	acc_skewness	gyro_skewness	post
0	24.634089	27.145404	3.427851	7.883738	3.901807	2.111136	
1	19.669705	5.408625	4.959198	3.908466	0.518804	2.214420	
2	12.868962	6.485444	0.036070	2.163744	1.884349	1.207264	
3	22.401442	0.621157	-0.633419	6.010449	0.782731	0.307339	
4	27.279259	55.655055	0.304565	8.694916	6.277500	1.198586	

Fase 1: Analisi Statistica

Eseguiamo una semplice e veloce analisi statistica al fine di individuare le feature meno rilevanti e rimuoverle.

```
# Calcoliamo la correlazione e visualizziamola sotto forma di Heatmap
corr_matrix = dataset.drop(['label'], axis=1).corr()
plt.figure(figsize=(10, 8))
plt.title('Matrice di correlazione')
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.show()
```



Osservando la matrice di correlazione potremmo dire "ad occhio" che le prime 3 feature del dataset non sembrano avere una grande influenza.

Potremmo eseguire un'analisi più approfondita ma esula dagli obbiettivi.

Per curiosità eseguiamo RandomForest:

```
# Separiamo il nostro dataset in valori e target
x_features = dataset.drop(columns=["label"])
y_target = dataset['label']
# Utiliziamo un modello di RF con un seed fisso per ripetibilità
model = RandomForestClassifier(random_state=1999)
model.fit(x_features, y_target)
# Estraiamo l'importanza delle feature dal modello addestrato
feature_importances = model.feature_importances_
# Riportiamo su dataframe
feature_importance_df = pd.DataFrame({'Feature': x_features.columns, 'Importanza': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importanza', ascending=True)
# Visualizziamo l'importanza delle feature
print(feature_importance_df)
```

```
          Feature  Importanza
2  gyro_kurtosis    0.040440
5  gyro_skewness    0.047879
3        lin_max    0.056931
0        acc_max    0.061430
1   acc_kurtosis    0.116101
4   acc_skewness    0.161451
6  post_gyro_max    0.222316
7   post_lin_max    0.293452
```

Dal RandomForest otteniamo risultati diversi a seconda del random_state. Per questo motivo eseguiremo successivamente dei test valutando l'accuratezza a seconda delle feature rimosse.

Fase 2: Preparazione Dataset

Prepariamo il dataset per essere dato in pasto a un modello. Essenzialmente dobbiamo ottenere un training set e un validation set.

Fase 2.1: Covertire le etichette in indici numerici

Questo blocco va eseguito solo che si hanno etichette letterali anziché numeriche.

```
values = array(dataset['label'])
# Passiamo per un encoder
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
# Convertiamo le etichette in variabili categoriali
onehot_encoder = OneHotEncoder(sparse_output=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
decoding_test = label_encoder.inverse_transform(integer_encoded.ravel())
print("Prime 6 etichette:", values[0:6])
print("Codifica:", [ x[0] for x in integer_encoded[0:6]])
print("Decodifica:", decoding_test[0:6])
dataset['label'] = integer_encoded
```

```
Prime 6 etichette: ['fallen' 'not_fallen' 'not_fallen' 'not_fallen' 'fallen' 'fallen']
Codifica: [0, 1, 1, 1, 0, 0]
Decodifica: ['fallen' 'not_fallen' 'not_fallen' 'not_fallen' 'fallen' 'fallen']
```

Fase 2.2: Creare training set e validation set

```
# Verifichiamo lo stato del dataframe
print(dataset.describe())
```

```
            acc_max  acc_kurtosis  gyro_kurtosis      lin_max  acc_skewness  \
count  1428.000000   1428.000000    1428.000000  1428.000000   1428.000000
mean     21.753410      9.964558       3.906186     7.934861      1.711623
std       5.479771     11.987739       5.495657     4.249702      1.530360
min       9.787964     -1.743347      -1.532044     0.043625    -14.066208
25%      18.822419      0.468756       0.182250     4.818588      0.452138
50%      22.866648      8.575117       2.028413     8.233184      1.513125
75%      25.863247     15.475659       5.456435    11.064898      2.865099
max      32.885551    231.134385      32.301675    25.382307      6.782592

       gyro_skewness  post_gyro_max  post_lin_max        label
count    1428.000000    1428.000000   1428.000000  1428.000000
mean        1.626049       3.232624      5.190129     0.570028
std         0.999605       3.432610      4.991879     0.495245
min        -0.460160      -4.939745     -5.382828     0.000000
25%         0.811533       0.347634      0.893603     0.000000
50%         1.529417       2.456817      3.696021     1.000000
75%         2.291334       5.224912      9.486155     1.000000
max         5.174101      16.204944     23.972115     1.000000
```

```
# IMPOSTAZIONI DEL DATASET
target_column = ['label']
# I Predittori sono le colonne di interesse per la predizione
predictors = ["lin_max","acc_skewness","gyro_skewness","post_gyro_max","post_lin_max"]

# Creiamo un dataframe con le sole colonne d'interesse, i predittori
x = dataset[predictors].values
# Dataframe contenente l'etichette
y = dataset[target_column].values
# Suddividiamo il dataset ottenuto in 2 sottodataset: training e test con un rapporto del 50%
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.80, random_state=1999)
# Usiamo un encoder per trasformali in variabili categoriali
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
count_classes = y_test.shape[1]
print("Esiti possibili:", count_classes)
```

```
Esiti possibili: 2
```

Fase 3: Alleniamo un modello

Basandoci su Tensorflow e Keras alleniamo una rete neurale per poter classificare nuovi record di dati.

```
# Creiamo il modello con 3 layer
model = Sequential()
model.add(Dense(6, activation='relu', input_dim=len(predictors)))
model.add(Dense(count_classes, activation='softmax'))
# Compiliamo il modello
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
# Eseguiamo l'allenamento:
EPOCHS=40
BATCH_SIZE=16
model.fit(x_train, y_train, BATCH_SIZE, EPOCHS, verbose=1)
print("~Allenamento ultimato~")
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_30 (Dense)            (None, 6)                 36

 dense_31 (Dense)            (None, 2)                 14

=================================================================
Total params: 50 (200.00 Byte)
Trainable params: 50 (200.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/40
18/18 [==============================] - 1s 3ms/step - loss: 0.6573 - accuracy: 0.4596
Epoch 2/40
18/18 [==============================] - 0s 3ms/step - loss: 0.5859 - accuracy: 0.4947
Epoch 3/40
18/18 [==============================] - 0s 3ms/step - loss: 0.5516 - accuracy: 0.5895
Epoch 4/40
18/18 [==============================] - 0s 3ms/step - loss: 0.5259 - accuracy: 0.6526
Epoch 5/40
18/18 [==============================] - 0s 3ms/step - loss: 0.5027 - accuracy: 0.7333
Epoch 6/40
18/18 [==============================] - 0s 3ms/step - loss: 0.4737 - accuracy: 0.7684
Epoch 7/40
18/18 [==============================] - 0s 3ms/step - loss: 0.4448 - accuracy: 0.8105
Epoch 8/40
18/18 [==============================] - 0s 3ms/step - loss: 0.4154 - accuracy: 0.8526
Epoch 9/40
18/18 [==============================] - 0s 3ms/step - loss: 0.3858 - accuracy: 0.8667
Epoch 10/40
18/18 [==============================] - 0s 3ms/step - loss: 0.3595 - accuracy: 0.8737
Epoch 11/40
18/18 [==============================] - 0s 3ms/step - loss: 0.3345 - accuracy: 0.8877
Epoch 12/40
18/18 [==============================] - 0s 3ms/step - loss: 0.3126 - accuracy: 0.8877
Epoch 13/40
18/18 [==============================] - 0s 3ms/step - loss: 0.2929 - accuracy: 0.9123
Epoch 14/40
18/18 [==============================] - 0s 3ms/step - loss: 0.2778 - accuracy: 0.9193
Epoch 15/40
18/18 [==============================] - 0s 3ms/step - loss: 0.2666 - accuracy: 0.9298
Epoch 16/40
18/18 [==============================] - 0s 3ms/step - loss: 0.2589 - accuracy: 0.9333
Epoch 17/40
18/18 [==============================] - 0s 3ms/step - loss: 0.2517 - accuracy: 0.9333
Epoch 18/40
18/18 [==============================] - 0s 3ms/step - loss: 0.2454 - accuracy: 0.9333
Epoch 19/40
18/18 [==============================] - 0s 4ms/step - loss: 0.2407 - accuracy: 0.9333
Epoch 20/40
18/18 [==============================] - 0s 4ms/step - loss: 0.2353 - accuracy: 0.9333
Epoch 21/40
18/18 [==============================] - 0s 4ms/step - loss: 0.2302 - accuracy: 0.9368
Epoch 22/40
18/18 [==============================] - 0s 4ms/step - loss: 0.2271 - accuracy: 0.9368
Epoch 23/40
```

```
# Verifichiamo l'accuracy ottenuta
scores = model.evaluate(x_train, y_train, verbose=0)
print('Accuracy sul training set: {:0.3f}% \n Errore sul training set: {:0.3f}%'.format(scores[1], 1 - scores[1]))
scores2 = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy sul test set: {:0.3f}% \n Errore sul test set: {:0.3f}%'.format(scores2[1], 1 - scores2[1]))
```

```
Accuracy sul training set: 0.951%
 Errore sul training set: 0.049%
Accuracy sul test set: 0.928%
 Errore sul test set: 0.072%
```

Fase 3.1: Test Predizione

```
# Tupla di dati
data_record = [0.1177684230459568,-1.0426728309341513,1.0397592018001995,-0.0219207284352537,0.0320953403062221]
prediction = model.predict([data_record], verbose=0)
for class_id, pred_acc in enumerate(prediction[0]):
  print('"{}" con accuratezza del {:0.3f}%'.format(label_encoder.inverse_transform([class_id])[0], pred_acc))
```

```
    "fallen" con accuratezza del 0.054%
    "not_fallen" con accuratezza del 0.946%
```

Fase 4: Esportiamo il modello

Allenato il modello dobbiamo convertirlo ed esportarlo per utilizzarlo nell'esp.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.experimental_new_quantizer = True
converter.experimental_new_converter = True
tflite_float_model = converter.convert()

float_model_size = len(tflite_float_model) / 1024
open("model_tensor.tflite", "wb").write(tflite_float_model)
print('Dimensione modello = %dKBs.' % float_model_size)
```

```
    Dimensione modello = 1KBs.
```

```
!xxd -i model_tensor.tflite > model_tensor.h
print("File model_tensor.h creato")
```