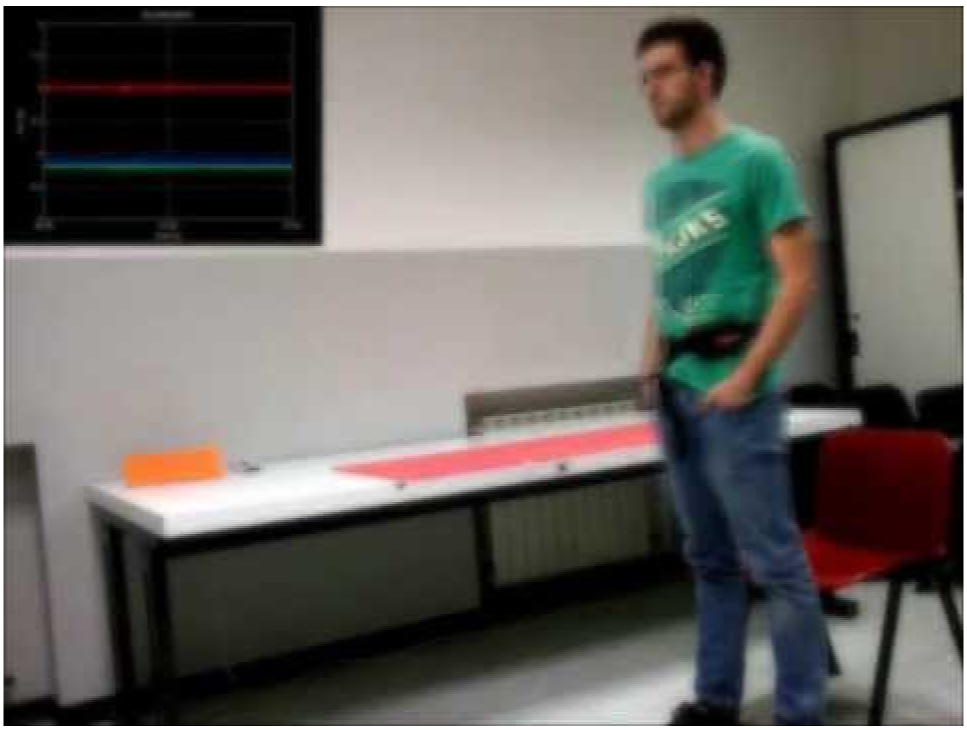
Assignment 1 - Part B.1: working with real data

In this assignment you will import and explore/analyze a dataset for classification. You will explore which ML algorithms are best to classify this and you will present your best solution. For this assignment we will use Human Activity dataset:

Description of the dataset:

The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities performed.



https://youtu.be/XOEN9W05_4A

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKINGUPSTAIRS, WALKINGDOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters. From the procecced input sensors a 561-feature vector with time and frequency domain variables is generated. For more details see: https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones

> NOTE: we have downloaded this dataset already for you and placed it on the github as HAR.zip

Steps

In this assignment you will analyze the data, train and evaluate a model based on this dataset.

These are the generic steps to be taken

1. Frame the problem and look at the big picture.
2. Get the data.
3. Explore the data to gain insights.
4. Prepare the data to better expose the underlying data patterns to Machine Learning algorithms
5. Explore many different models and short-list the best ones.
6. Fine-tune your models and combine them into a great solution.
7. Present your solution.
8. Launch, monitor, and maintain your system.
9. Additional Questions

In the Notebook this structure is used for dividing the different steps, so make sure you do the implementation and analisis at these location in the notebook.

You may add additinal code blocks, but keep the seperation of the given structure.

At the end of each block summarize / comment / conclude your current step in the given textblocks.

At the end you have to hand in this notebook together with the notebooks of Assignment 1, when you hand it in you should make sure that you saved it with all output visible. So we can evaluate your notebooks output without directly ruinning it. In addition (to be sure) you should also save a pdf of the final result.

Hints

The needed dataset is available in our github repository (HAR.zip), how to download this from your notebook and addition hints are available in the Tips & Tricks file

```
Felix Douven, Kasper walraven, Yosha op het Veld
```

1. Frame the problem and look at the big picture

Describe the problem at hand and explain your approach

```
The goal is to create a model which is based on the "Train" data that can
correctly predict which activity a subject is undertaking.
The Data will be analised, and checked for any errors such as missing data or
duplicate data. Multiple algorithms will be tested and plotted to determine the
most succesful Algorithm(s). This model consists of the most succesful
algorithm(s) tested.
```

2. Get the data.

Initialize the system, get all needed libraries, retreive the data and import it

> NOTE: You can download the dataset directly from github, see Tips & Tricks

In [1]:
```
import zipfile
import pandas as pd
#import numpy as np
import sklearn
import matplotlib.pyplot as plt

from sklearn import model_selection
from sklearn.model_selection import cross_validate
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from pandas.plotting import scatter_matrix

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from warnings import simplefilter
from sklearn.model_selection import cross_val_score
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression


test = pd.read_csv("test.csv")
train= pd.read_csv("train.csv")
data = pd.concat([test, train])

print(data)
```

```
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
0              0.257178          -0.023285          -0.014654
1              0.286027          -0.013163          -0.119083
```

```
2             0.275485          -0.026050          -0.118152
3             0.270298          -0.032614          -0.117520
4             0.274833          -0.027848          -0.129527
...                ...                ...                ...
7347          0.299665          -0.057193          -0.181233
7348          0.273853          -0.007749          -0.147468
7349          0.273387          -0.017011          -0.045022
7350          0.289654          -0.018843          -0.158281
7351          0.351503          -0.012423          -0.203867

      tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
0            -0.938404         -0.920091         -0.667683         -0.952501
1            -0.975415         -0.967458         -0.944958         -0.986799
2            -0.993819         -0.969926         -0.962748         -0.994403
3            -0.994743         -0.973268         -0.967091         -0.995274
4            -0.993852         -0.967445         -0.978295         -0.994111
...                ...               ...               ...               ...
7347         -0.195387          0.039905          0.077078         -0.282301
7348         -0.235309          0.004816          0.059280         -0.322552
7349         -0.218218         -0.103822          0.274533         -0.304515
7350         -0.219139         -0.111412          0.268893         -0.310487
7351         -0.269270         -0.087212          0.177404         -0.377404

      tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
0            -0.925249         -0.674302         -0.894088  ...
1            -0.968401         -0.945823         -0.894088  ...
2            -0.970735         -0.963483         -0.939260  ...
3            -0.974471         -0.968897         -0.938610  ...
4            -0.965953         -0.977346         -0.938610  ...
...                ...               ...               ...  ...
7347          0.043616          0.060410          0.210795  ...
7348         -0.029456          0.080585          0.117440  ...
7349         -0.098913          0.332584          0.043999  ...
7350         -0.068200          0.319473          0.101702  ...
7351         -0.038678          0.229430          0.269013  ...

      fBodyBodyGyroJerkMag-kurtosis()  angle(tBodyAccMean,gravity)  \
0                           -0.705974                     0.006462
1                           -0.594944                    -0.083495
2                           -0.640736                    -0.034956
3                           -0.736124                    -0.017067
4                           -0.846595                    -0.002223
...                               ...                          ...
7347                        -0.880324                    -0.190437
7348                        -0.680744                     0.064907
7349                        -0.304029                     0.052806
7350                        -0.344314                    -0.101360
7351                        -0.740738                    -0.280088

      angle(tBodyAccJerkMean),gravityMean)  angle(tBodyGyroMean,gravityMean)  \
0                                 0.162920                         -0.825886
1                                 0.017500                         -0.434375
2                                 0.202302                          0.064103
3                                 0.154438                          0.340134
4                                -0.040046                          0.736715
...                                    ...                               ...
7347                              0.829718                          0.206972
7348                              0.875679                         -0.879033
7349                             -0.266724                          0.864404
7350                              0.700740                          0.936674
7351                             -0.007739                         -0.056088

      angle(tBodyGyroJerkMean,gravityMean)  angle(X,gravityMean)  \
0                                 0.271151             -0.720009
1                                 0.920593             -0.698091
2                                 0.145068             -0.702771
```

```
3                                  0.296407               -0.698954
4                                 -0.118545               -0.692245
...                                     ...                     ...
7347                              -0.425619               -0.791883
7348                               0.400219               -0.771840
7349                               0.701169               -0.779133
7350                              -0.589479               -0.785181
7351                              -0.616956               -0.783267

      angle(Y,gravityMean)  angle(Z,gravityMean)  subject          Activity
0                 0.276801             -0.057978        2          STANDING
1                 0.281343             -0.083898        2          STANDING
2                 0.280083             -0.079346        2          STANDING
3                 0.284114             -0.077108        2          STANDING
4                 0.290722             -0.073857        2          STANDING
...                    ...                   ...      ...               ...
7347              0.238604              0.049819       30  WALKING_UPSTAIRS
7348              0.252676              0.050053       30  WALKING_UPSTAIRS
7349              0.249145              0.040811       30  WALKING_UPSTAIRS
7350              0.246432              0.025339       30  WALKING_UPSTAIRS
7351              0.246809              0.036695       30  WALKING_UPSTAIRS

[10299 rows x 563 columns]
```

```
Importing CSV files from same directory and printing them
```

3. Explore the data to gain insights.

Explore the data in any possible way, visualize the results (if you have multiple plots of the same kind of data put them in one larger plot)

> NOTE:You can visualize high-dimensional data in 2-d using T-distributed Stochastic Neighbor Embedding, see Tips & Tricks. (You can also visualze it in 3D, as described in the tutorial)

In [2]:

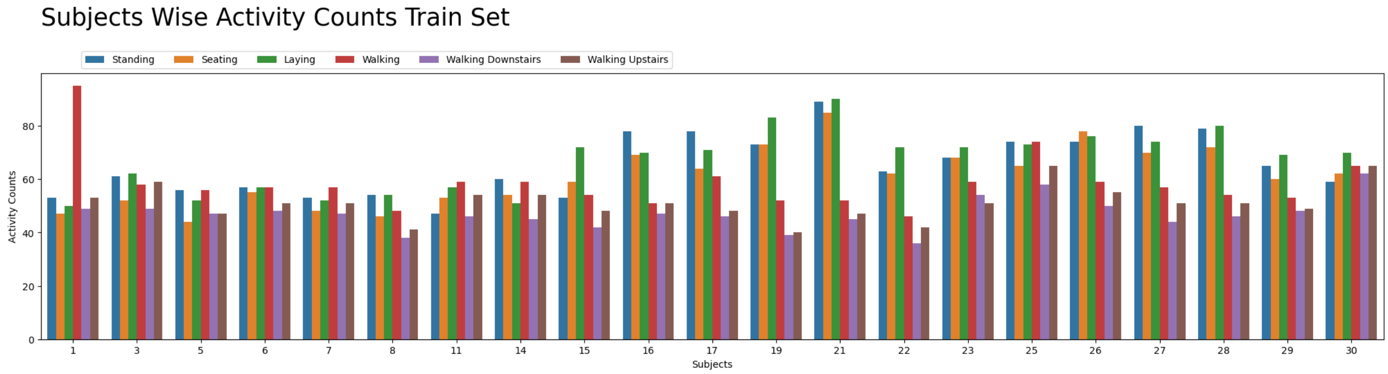
```python
print(f"Train Dataset Shape: {train.shape}")
print(f"Test Dataset Shape: {test.shape}")
print("")
print(f"Train Dataset Missing Data Counts: {train.isna().sum().sum()}")
print(f"Test Dataset Missing Data Counts: {test.isna().sum().sum()}")
print("")
print(f"Train Dataset Duplicate Data Counts: {train.duplicated().sum()}")
print(f"Test Dataset Duplicate Data Counts: {test.duplicated().sum()}")
```

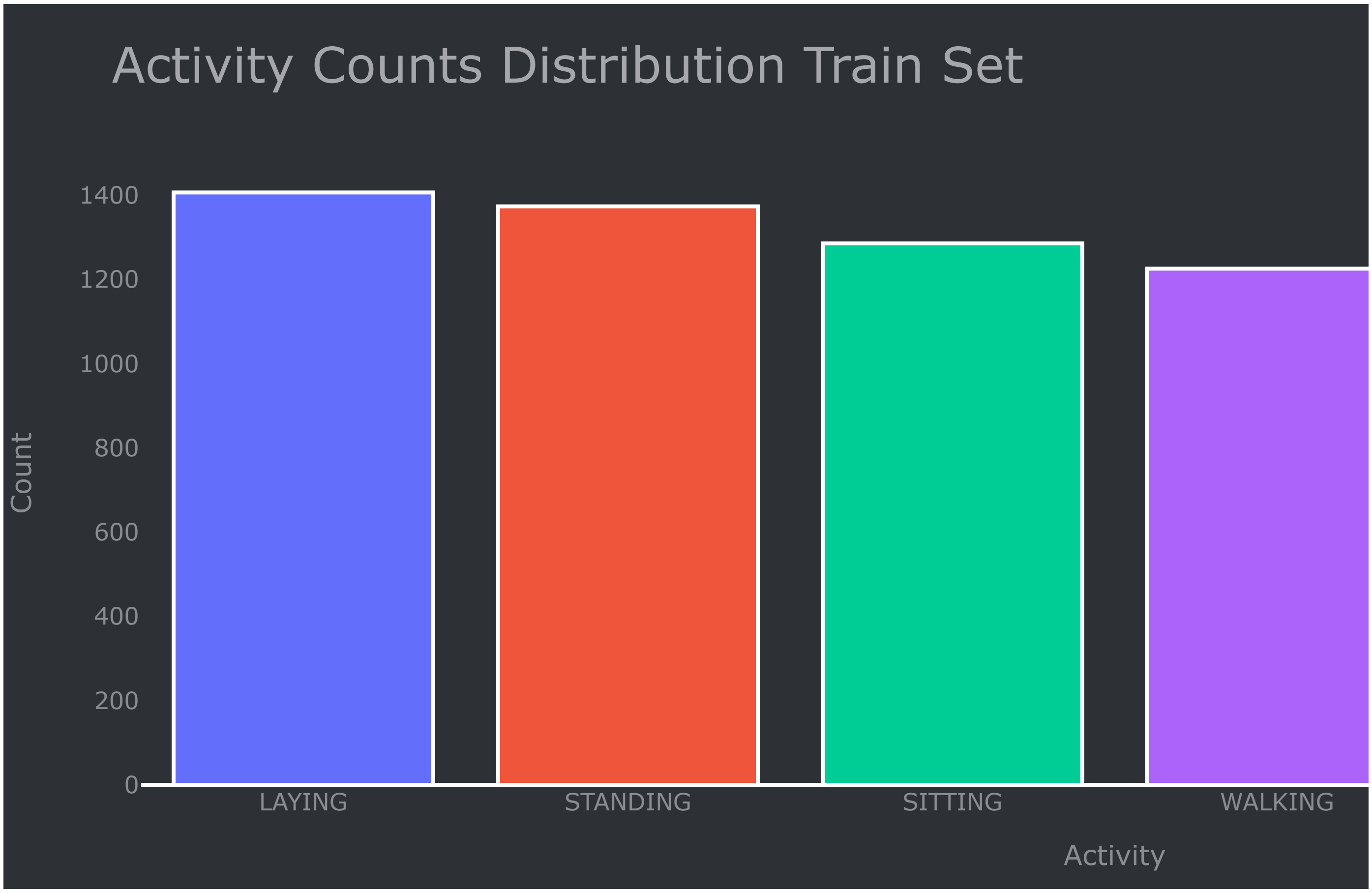
```
Train Dataset Shape: (7352, 563)
Test Dataset Shape: (2947, 563)

Train Dataset Missing Data Counts: 0
Test Dataset Missing Data Counts: 0

Train Dataset Duplicate Data Counts: 0
Test Dataset Duplicate Data Counts: 0
```

```
This shows The shape of the data [columns, rows]

The "missing data counts" shows that no data is missing

The "duplicate data counts" shows that no data is the same
```

In [3]:

```
import os
import time

import warnings
import pandas as pd
#import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
from matplotlib.colors import rgb2hex
from matplotlib.cm import get_cmap
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
from mpl_toolkits import mplot3d
from pylab import rcParams

from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression,RidgeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier, GradientBoostin
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import train_test_split


plt.figure(figsize=(25, 5))
count_plot=sns.countplot(data=train, x='subject', hue='Activity')
plt.gca().tick_params(axis='x')
plt.gca().tick_params(axis='y')
plt.xlabel( xlabel='Subjects')
plt.ylabel( ylabel='Activity Counts')
plt.legend(["Standing", "Seating", "Laying", "Walking", "Walking Downstairs", "Walking U
plt.title("Subjects Wise Activity Counts Train Set", fontsize=25, loc='left', pad=50)
plt.show()


plt.figure(figsize=(5, 5))
label_counts = train['Activity'].value_counts()
colors = px.colors.qualitative.Plotly

graph = go.Bar(x=label_counts.index, y=label_counts.values, marker = dict(color = colors
layout = go.Layout(
    height=450, width=1100,
    title = 'Activity Counts Distribution Train Set',
    xaxis = dict(title = 'Activity', tickangle=0, showgrid=False),
    yaxis = dict(title = 'Count', showgrid=False),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    title_font=dict(size=25, color='#a5a7ab'),
    margin=dict(t=80, b=30, l=70, r=40),
    font=dict(color='#8a8d93'))
fig = go.Figure(data=[graph], layout = layout)
fig.update_traces(textfont=dict(color='#fff'), marker=dict(line=dict(color='#ffffff', wi

iplot(fig)
```





<Figure size 500x500 with 0 Axes>

This plot shows how much data each activity contains. The amount of rows per activity equals to how much data is aquired for that activity.

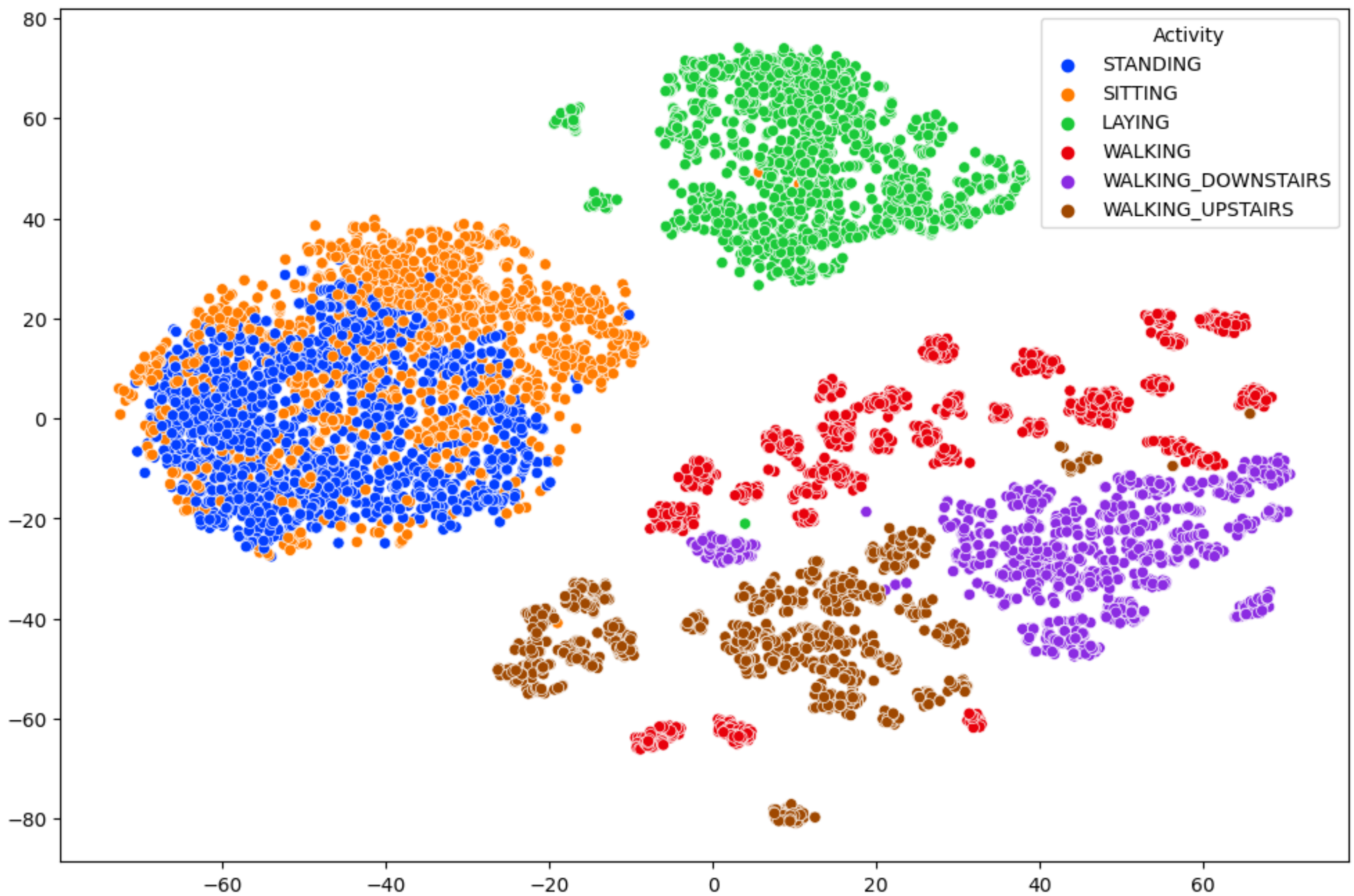
In [4]:
```
# t-sne (2D)
x_for_tsne = train.drop(['subject', 'Activity'], axis=1)

tsne = TSNE(random_state = 42, n_components=2, verbose=1, perplexity=50, n_iter=1000).fi
plt.figure(figsize=(12,8))
sns.scatterplot(x =tsne[:, 0], y = tsne[:, 1], hue = train["Activity"],palette="bright")
```

```
D:\programs\anaconda\lib\site-packages\sklearn\manifold\_t_sne.py:780: FutureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

D:\programs\anaconda\lib\site-packages\sklearn\manifold\_t_sne.py:790: FutureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.010s...
[t-SNE] Computed neighbors for 7352 samples in 1.568s...
```

```
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.437672
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.125526
[t-SNE] KL divergence after 1000 iterations: 1.280823
```

Out[4]: <AxesSubplot:>



In a scatterplot groups are very easily identifiable except standing and sitting. As the picture shows there are clear groups per activity with some outliars.

In [5]:

```
x_for_tsne = train.drop(['subject', 'Activity'], axis=1)

tsne = TSNE(random_state = 42, n_components=3, verbose=1, perplexity=50, n_iter=1000).fi

fig = px.scatter_3d(
    x =tsne[:, 0],
    y = tsne[:, 1],
    z = tsne[:, 2],
    color=train['Activity']
)
fig.update_layout(
    title="Cluster Of Activities",
    title_font=dict(size=25, color='#a5a7ab'),
    font=dict(color='#8a8d93'),
    plot_bgcolor='#2d3035', paper_bgcolor='#2d3035',
    margin=dict(t=100, b=10, l=70, r=40),
```

```
    )
fig.show()
```

```
D:\programs\anaconda\lib\site-packages\sklearn\manifold\_t_sne.py:780: FutureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

D:\programs\anaconda\lib\site-packages\sklearn\manifold\_t_sne.py:790: FutureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.006s...
[t-SNE] Computed neighbors for 7352 samples in 1.304s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.437672
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.207909
[t-SNE] KL divergence after 1000 iterations: 1.122070
```

3d Scatterplot in xyz

The scatterplot can properly define which subject did which activity with the use of clusters.

4. Prepare the data to better expose the underlying data patterns to Machine Learning algorithms

prepare your data, is it normalized? are there outlier? Make a training and a test set.

In [6]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

test_data=pd.read_csv("test.csv", usecols=range(0,561))
train_data=pd.read_csv("train.csv", usecols=range(0,561))
data_data = pd.concat([test_data, train_data])

X = data_data
Y = data.Activity
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=No

print("There are sampels and dimensions for the features", X.shape)
print("There are sampels and dimensions for the Targets", Y.shape)

print("Training Data input")
print(X_train)
print("")
print("Training Activity Data")
print(y_train)
print("")
print("Testing Data input")
print(X_test)
print("")
print("Testing Activity Data")
print(y_test)
```

```
There are sampels and dimensions for the features (10299, 561)
There are sampels and dimensions for the Targets (10299,)
Training Data input
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
3618           0.276350          -0.017840          -0.108187
2122           0.259121          -0.020214          -0.107022
1714           0.276454          -0.018011          -0.112289
5251           0.275105          -0.091865          -0.042848
4148           0.234858          -0.008744          -0.102860
...                 ...                ...                ...
2364           0.119551          -0.076462          -0.087633
1816           0.265116           0.011964          -0.122945
2655           0.300491          -0.011320          -0.121599
2079           0.283215          -0.017897          -0.114807
3893           0.318113          -0.013463          -0.105884

      tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
3618         -0.996679         -0.986409         -0.985721         -0.997725
2122         -0.984459         -0.981731         -0.992721         -0.986036
1714         -0.998086         -0.986495         -0.991307         -0.998558
5251         -0.630739         -0.336784         -0.197750         -0.671136
4148         -0.389979          0.080706         -0.328358         -0.440863
...                ...               ...               ...               ...
2364         -0.327485          0.250129          0.545038         -0.359978
1816         -0.948191         -0.874576         -0.746579         -0.952750
2655         -0.970115         -0.965097         -0.971075         -0.972175
2079         -0.996795         -0.994197         -0.988049         -0.997367
```

```
3893        -0.949817        -0.987826        -0.980174        -0.952695

      tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
3618         -0.986947         -0.985126         -0.938876  ...
2122         -0.982402         -0.993335         -0.925993  ...
1714         -0.986315         -0.991080         -0.941172  ...
5251         -0.332226         -0.094576         -0.354311  ...
4148          0.038294         -0.292347         -0.305557  ...
...                ...               ...               ...  ...
2364          0.230040          0.432268         -0.311373  ...
1816         -0.876316         -0.743528         -0.879449  ...
2655         -0.978152         -0.970350         -0.916746  ...
2079         -0.993857         -0.986238         -0.937850  ...
3893         -0.988527         -0.981203         -0.873537  ...

      fBodyBodyGyroJerkMag-meanFreq()  fBodyBodyGyroJerkMag-skewness()  \
3618                         0.196921                        -0.722753
2122                         0.429988                        -0.793231
1714                         0.497133                        -0.599359
5251                        -0.036928                        -0.192382
4148                        -0.011568                        -0.231377
...                               ...                              ...
2364                        -0.063802                        -0.589820
1816                         0.173982                        -0.759867
2655                        -0.039724                        -0.340901
2079                         0.574323                        -0.692909
3893                        -0.046037                        -0.082870

      fBodyBodyGyroJerkMag-kurtosis()  angle(tBodyAccMean,gravity)  \
3618                        -0.951052                     0.234107
2122                        -0.959971                     0.203545
1714                        -0.870441                     0.071602
5251                        -0.524444                    -0.145601
4148                        -0.570441                     0.923038
...                               ...                          ...
2364                        -0.884238                     0.349288
1816                        -0.966328                    -0.021564
2655                        -0.739608                     0.033990
2079                        -0.901913                    -0.150503
3893                        -0.543959                     0.029901

      angle(tBodyAccJerkMean),gravityMean)  angle(tBodyGyroMean,gravityMean)  \
3618                              0.223342                         -0.579929
2122                              0.378064                         -0.271598
1714                              0.036404                         -0.484351
5251                              0.500261                         -0.849437
4148                             -0.892936                         -0.773701
...                                    ...                               ...
2364                             -0.823770                         -0.962928
1816                              0.471434                          0.050466
2655                              0.496431                          0.705421
2079                             -0.218535                          0.120355
3893                              0.064021                          0.393896

      angle(tBodyGyroJerkMean,gravityMean)  angle(X,gravityMean)  \
3618                              0.726697             -0.741352
2122                             -0.488304              0.396828
1714                              0.717393             -0.729331
5251                             -0.590291             -0.727763
4148                              0.634180             -0.440143
...                                    ...                   ...
2364                              0.633123             -0.446078
1816                              0.150964              0.564165
2655                             -0.513216              0.589764
2079                              0.218468             -0.717280
3893                              0.545465              0.861152
```

```
      angle(Y,gravityMean)  angle(Z,gravityMean)
3618              0.257321             -0.065613
2122             -0.306216             -0.679791
1714              0.283799             -0.005184
5251              0.211477             -0.137747
4148              0.400211              0.252875
...                    ...                   ...
2364              0.271849              0.365005
1816             -0.152679             -0.867329
2655             -0.537132             -0.468555
2079             -0.034180             -0.134069
3893             -0.434905             -0.524960

[8239 rows x 561 columns]

Training Activity Data
3618            STANDING
2122              LAYING
1714            STANDING
5251    WALKING_UPSTAIRS
4148    WALKING_UPSTAIRS
              ...
2364    WALKING_UPSTAIRS
1816              LAYING
2655              LAYING
2079             SITTING
3893              LAYING
Name: Activity, Length: 8239, dtype: object

Testing Data input
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
6288           0.274171          -0.024419          -0.105057
5980           0.356439          -0.008397          -0.131603
5511           0.276257          -0.016113          -0.104247
954            0.671887          -0.014351          -0.159904
2449           0.272332          -0.008278          -0.154984
...                 ...                ...                ...
4106           0.263356          -0.042670          -0.089629
4349           0.199160           0.007932          -0.118969
2854           0.160887           0.027138          -0.074331
5550           0.280448          -0.010378          -0.118801
2438           0.276846          -0.016135          -0.105712

      tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
6288         -0.989976         -0.978888         -0.953020         -0.991369
5980         -0.330567         -0.142191         -0.258825         -0.357955
5511         -0.997458         -0.983537         -0.990844         -0.997708
954          -0.723274         -0.712494         -0.832049         -0.731259
2449         -0.963339         -0.931837         -0.887040         -0.967374
...                ...               ...               ...               ...
4106         -0.319975         -0.009298         -0.206526         -0.349792
4349         -0.285981          0.085926         -0.242673         -0.317570
2854         -0.071306         -0.157939         -0.376917         -0.146765
5550         -0.997560         -0.990822         -0.970924         -0.997642
2438         -0.999159         -0.991561         -0.988873         -0.999141

      tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
6288         -0.979123         -0.949340         -0.936269  ...
5980         -0.147799         -0.303396         -0.031278  ...
5511         -0.982723         -0.990336         -0.943105  ...
954          -0.762424         -0.836992         -0.462383  ...
2449         -0.936477         -0.893442         -0.895867  ...
...                ...               ...               ...  ...
4106          0.000059         -0.181353         -0.208157  ...
4349          0.067290         -0.259282         -0.197206  ...
```

```
2854        -0.140669          -0.339094          0.145756  ...
5550        -0.989987          -0.970163         -0.942603  ...
2438        -0.991199          -0.988271         -0.943487  ...

      fBodyBodyGyroJerkMag-meanFreq()  fBodyBodyGyroJerkMag-skewness()  \
6288                        -0.436397                        -0.534076
5980                        -0.017969                         0.176492
5511                         0.436541                        -0.421554
954                         -0.312687                         0.104009
2449                         0.303382                        -0.436925
...                               ...                              ...
4106                         0.108847                         0.316145
4349                         0.040860                        -0.583636
2854                         0.006947                        -0.737691
5550                         0.620536                        -0.806744
2438                         0.580231                        -0.793868

      fBodyBodyGyroJerkMag-kurtosis()  angle(tBodyAccMean,gravity)  \
6288                        -0.891850                     0.119745
5980                        -0.130638                    -0.642215
5511                        -0.711950                     0.115875
954                         -0.323837                     0.069077
2449                        -0.766334                     0.025370
...                               ...                          ...
4106                         0.044828                    -0.013065
4349                        -0.902788                     0.671309
2854                        -0.962371                     0.625118
5550                        -0.952757                     0.004945
2438                        -0.973005                    -0.016905

      angle(tBodyAccJerkMean),gravityMean)  angle(tBodyGyroMean,gravityMean)  \
6288                             -0.116853                         -0.296840
5980                              0.325625                         -0.851538
5511                              0.081942                          0.289036
954                              -0.330374                         -0.150341
2449                              0.078480                          0.073750
...                                    ...                               ...
4106                             -0.748439                          0.566667
4349                              0.146944                          0.455758
2854                             -0.251341                          0.913197
5550                              0.016978                         -0.056392
2438                             -0.252440                         -0.197753

      angle(tBodyGyroJerkMean,gravityMean)  angle(X,gravityMean)  \
6288                             -0.111986             -0.539553
5980                              0.834120             -0.864330
5511                             -0.239476             -0.868190
954                               0.001035              0.644271
2449                             -0.117424             -0.927072
...                                    ...                   ...
4106                             -0.588309             -0.582142
4349                              0.573820             -0.448166
2854                             -0.684287             -0.684052
5550                              0.094890             -0.849510
2438                              0.478495             -0.732457

      angle(Y,gravityMean)  angle(Z,gravityMean)
6288             -0.000985             -0.296717
5980              0.189600              0.039720
5511              0.178597             -0.027016
954              -0.777778             -0.215257
2449              0.132262              0.057362
...                    ...                   ...
4106              0.336687              0.180546
4349              0.397239              0.248203
2854              0.309557             -0.037125
```

```
5550            0.107354          -0.091660
2438           -0.086340          -0.021667

[2060 rows x 561 columns]

Testing Activity Data
6288              SITTING
5980              WALKING
5511             STANDING
954                LAYING
2449              SITTING
               ...
4106              WALKING
4349     WALKING_UPSTAIRS
2854     WALKING_UPSTAIRS
5550              SITTING
2438              SITTING
Name: Activity, Length: 2060, dtype: object
```

```
Data is getting shuffled meaning the rows are getting randomized. Splitting
data in data inputs and data outputs (activity) which is used for learning.
```

5. Explore many different models and short-list the best ones.

Explore / train and list the top 3 algorithms that score best on this dataset.

In [7]:

```
X = data_data
Y = data.Activity
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=No
i = 0

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import ConfusionMatrixDisplay




#algorithm functions
#----------------------------------------------------------------------------
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
#----------------------------------------------------------------------------
svclassifier_lin = SVC(kernel='linear')
svclassifier_lin.fit(X_train, y_train)
y_pred_lin = svclassifier_lin.predict(X_test)
#----------------------------------------------------------------------------
svclassifier_pol = SVC(kernel='poly', degree=8)
svclassifier_pol.fit(X_train, y_train)
y_pred_pol = svclassifier_pol.predict(X_test)
#----------------------------------------------------------------------------
svclassifier_rbf = SVC(kernel='rbf')
svclassifier_rbf.fit(X_train, y_train)
y_pred_rbf = svclassifier_rbf.predict(X_test)
#----------------------------------------------------------------------------
dtc = DecisionTreeClassifier()
```

```
dtc.fit(X_train, y_train)
y_pred_dtc = dtc.predict(X_test)
#---------------------------------------------------------------------------
etc = ExtraTreesClassifier()
etc.fit(X_train, y_train)
y_pred_etc = etc.predict(X_test)
#---------------------------------------------------------------------------
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
y_pred_rfc = rfc.predict(X_test)
#---------------------------------------------------------------------------
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_test)
#---------------------------------------------------------------------------
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
y_pred_qda = qda.predict(X_test)


#plotting non normalized confusion matrices
print("KNN Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_knn))
print("Linear Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_lin))
print("Poly Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_pol))
print("Rbf Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_rbf))
print("DecisionTree Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_dtc))
print("ExtraTrees Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_etc))
print("RandomForest Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_rfc))
print("GaussianNB Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_gnb))
print("QuadraticDiscriminantAnalysis Not normalized confusion matrix")
print(confusion_matrix(y_test,y_pred_qda))



#lists of algorithms
class_names = data.Activity
titles_options = [
    ("KNN Normalized confusion matrix", "true"),
    ("Linear_SVM Normalized confusion matrix", "true"),
    ("Poly_SVM Normalized confusion matrix", "true"),
    ("rbf_SVM Normalized confusion matrix", "true"),
    ("DecisionTree Normalized confusion matrix", "true"),
    ("ExtraTrees Normalized confusion matrix", "true"),
    ("RandomForest Normalized confusion matrix", "true"),
    ("GaussianNB Normalized confusion matrix", "true"),
    ("QuadraticDiscriminantAnalysis Normalized confusion matrix", "true")
]

algo_names = [
    knn,
    svclassifier_lin,
    svclassifier_pol,
    svclassifier_rbf,
    dtc,
    etc,
    rfc,
    gnb,
    qda
```

```
]


title_names = [
    "knn",
    "lin",
    "pol",
    "rbf",
    "dtc",
    "etc",
    "rfc",
    "gnb",
    "qda"

]

pred_names = [
    y_pred_knn,
    y_pred_lin,
    y_pred_pol,
    y_pred_rbf,
    y_pred_dtc,
    y_pred_etc,
    y_pred_rfc,
    y_pred_gnb,
    y_pred_qda
]

#plotting normalized matrices
for title, normalize in titles_options:
    disp = ConfusionMatrixDisplay.from_estimator(algo_names[i], X_test, y_test, cmap=plt
    disp.ax_.set_title(title)
    i = i + 1
plt.show()

#printing and calculting accuracy
i = 0
for title in title_names:
    print(title,"accuracy :", round(sklearn.metrics.accuracy_score(y_test, pred_names[i]
    i = i + 1


#disp = ConfusionMatrixDisplay.from_estimator(svclassifier_lin, X_test, y_test, cmap=plt
#disp.ax_.set_title(title)
#disp = ConfusionMatrixDisplay.from_estimator(svclassifier_pol, X_test, y_test, cmap=plt
#disp.ax_.set_title(title)
#disp = ConfusionMatrixDisplay.from_estimator(svclassifier_rbf, X_test, y_test, cmap=plt
#disp.ax_.set_title(title)

#print("knn accuracy :", round(sklearn.metrics.accuracy_score(y_test, y_pred_knn)*100, 2
#print("lin accuracy :", round(sklearn.metrics.accuracy_score(y_test, y_pred_lin)*100, 2
#print("pol accuracy :", round(sklearn.metrics.accuracy_score(y_test, y_pred_pol)*100, 2
#print("rbf accuracy :", round(sklearn.metrics.accuracy_score(y_test, y_pred_rbf)*100, 2

#print(classification_report(y_test,y_pred))
```

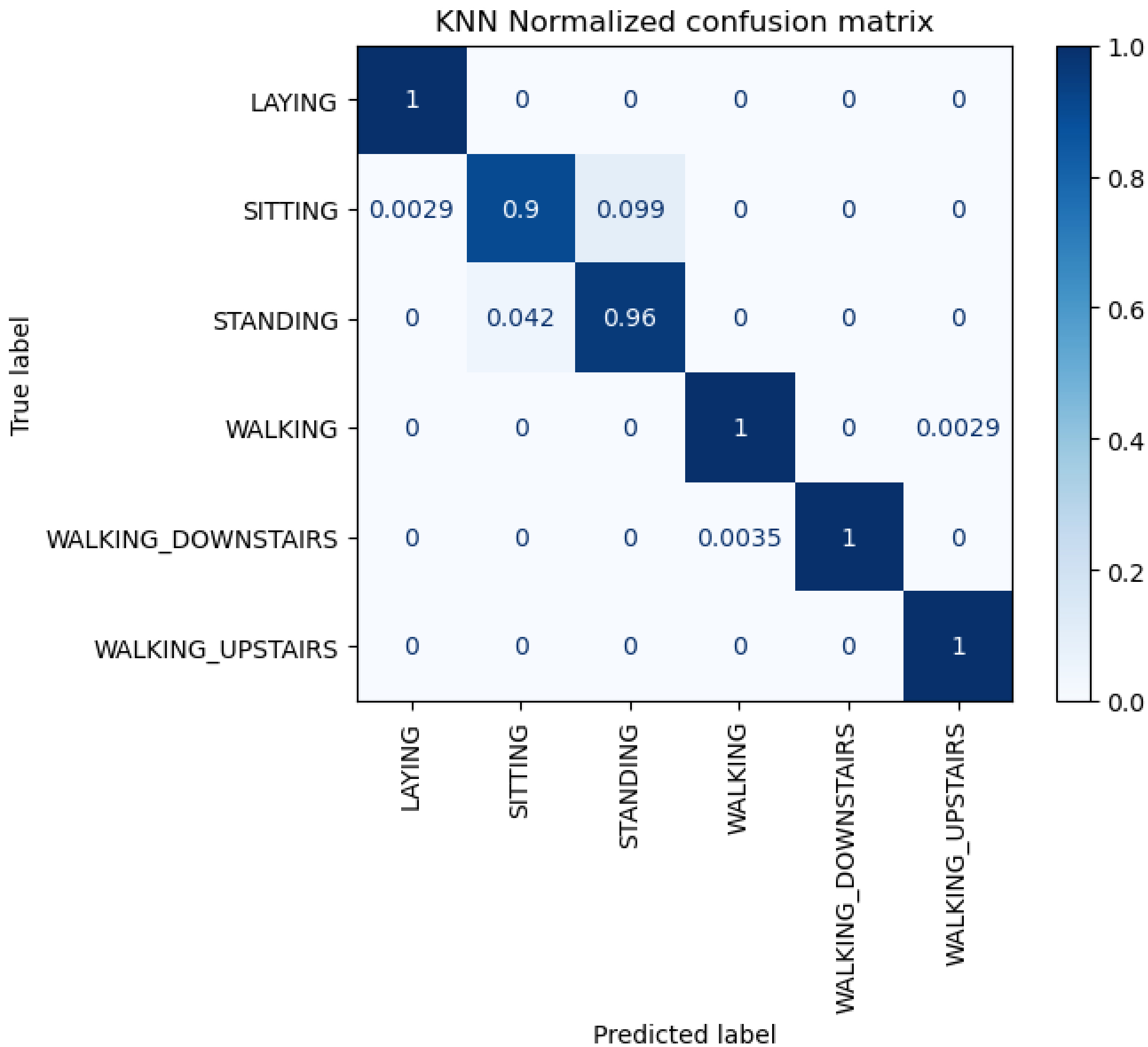
```
D:\programs\anaconda\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureW
arning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mod
e` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will chang
e: the default value of `keepdims` will become False, the `axis` over which the statisti
c is taken will be eliminated, and the value None will no longer be accepted. Set `keepd
ims` to True or False to avoid this warning.
```

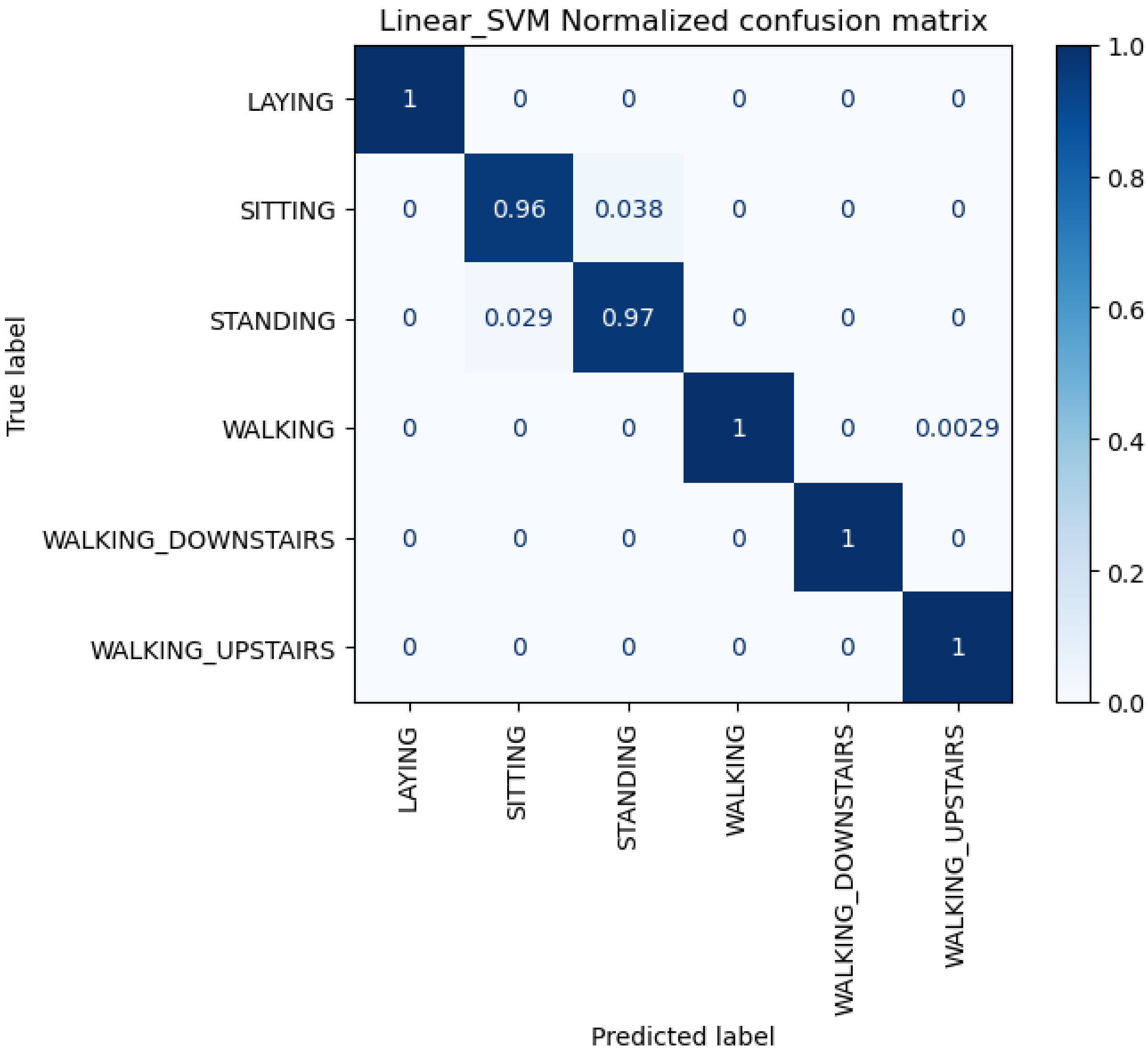
```
D:\programs\anaconda\lib\site-packages\sklearn\discriminant_analysis.py:878: UserWarnin
g:

Variables are collinear
```

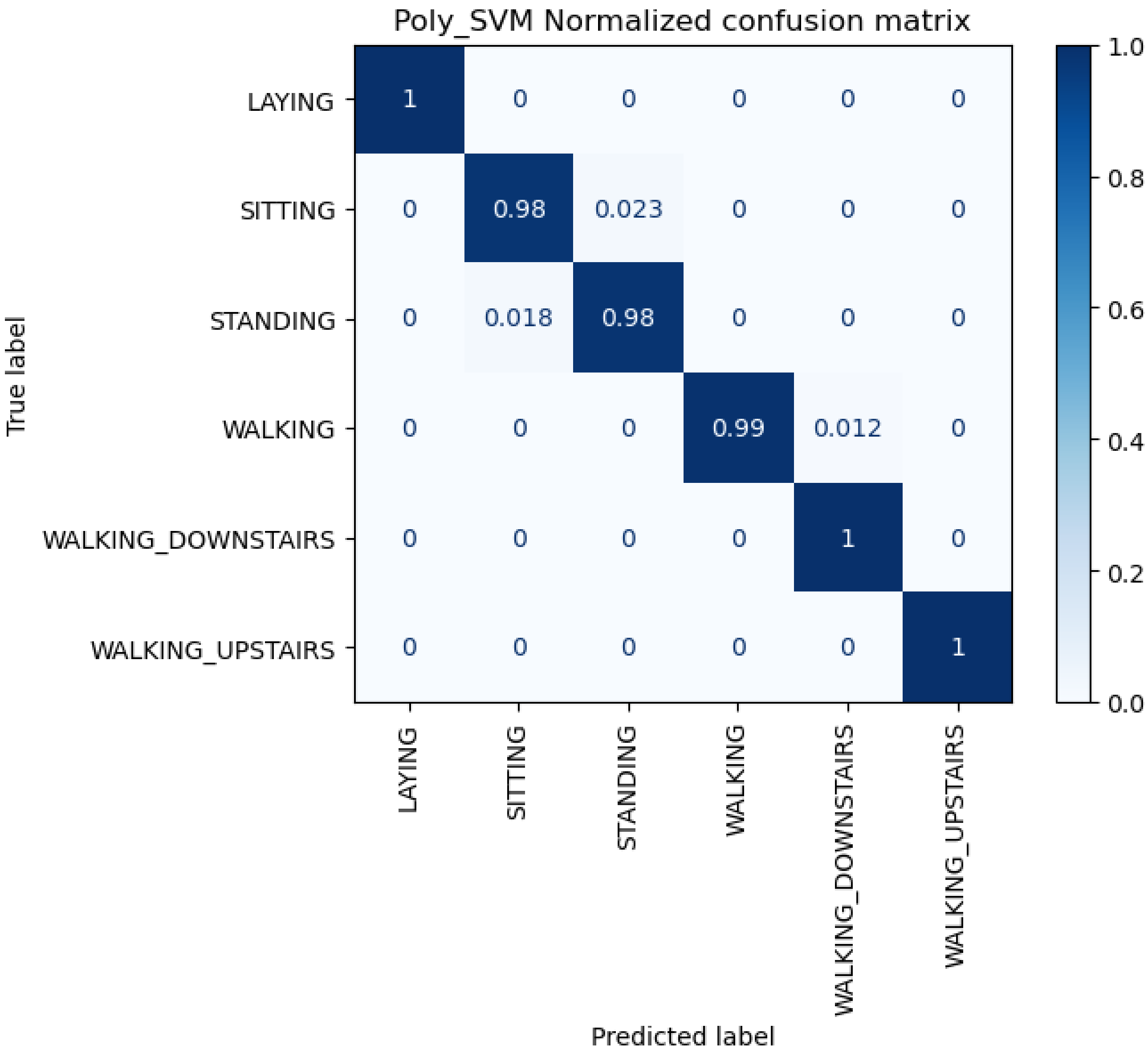
```
KNN Not normalized confusion matrix
[[374   0   0   0   0   0]
 [  1 309  34   0   0   0]
 [  0  16 368   0   0   0]
 [  0   0   0 343   0   1]
 [  0   0   0   1 288   0]
 [  0   0   0   0   0 325]]
Linear Not normalized confusion matrix
[[374   0   0   0   0   0]
 [  0 331  13   0   0   0]
 [  0  11 373   0   0   0]
 [  0   0   0 343   0   1]
 [  0   0   0   0 289   0]
 [  0   0   0   0   0 325]]
Poly Not normalized confusion matrix
[[374   0   0   0   0   0]
 [  0 336   8   0   0   0]
 [  0   7 377   0   0   0]
 [  0   0   0 340   4   0]
 [  0   0   0   0 289   0]
 [  0   0   0   0   0 325]]
Rbf Not normalized confusion matrix
[[374   0   0   0   0   0]
 [  0 321  22   0   0   1]
 [  0  22 362   0   0   0]
 [  0   0   0 343   0   1]
 [  0   0   0   1 288   0]
 [  0   0   0   1   0 324]]
DecisionTree Not normalized confusion matrix
[[374   0   0   0   0   0]
 [  0 308  36   0   0   0]
 [  0  33 351   0   0   0]
 [  0   0   0 330   2  12]
 [  0   0   0   7 269  13]
 [  0   0   0  13  19 293]]
ExtraTrees Not normalized confusion matrix
[[374   0   0   0   0   0]
 [  0 330  14   0   0   0]
 [  0   5 379   0   0   0]
 [  0   0   0 340   2   2]
 [  0   0   0   0 287   2]
 [  0   0   0   0   3 322]]
RandomForest Not normalized confusion matrix
[[374   0   0   0   0   0]
 [  0 329  15   0   0   0]
 [  0  12 372   0   0   0]
 [  0   0   0 339   1   4]
 [  0   0   0   1 283   5]
 [  0   0   0   1   5 319]]
GaussianNB Not normalized confusion matrix
[[303  70   0   0   0   1]
 [  4 301  34   0   0   5]
 [  2 255 120   0   0   7]
 [  0   0   0 271  30  43]
 [  0   0   0  13 228  48]
 [  0   0   0   6  18 301]]
QuadraticDiscriminantAnalysis Not normalized confusion matrix
[[373   1   0   0   0   0]
 [  1 296  37   5   2   3]
 [  0 138 235   6   2   3]
```

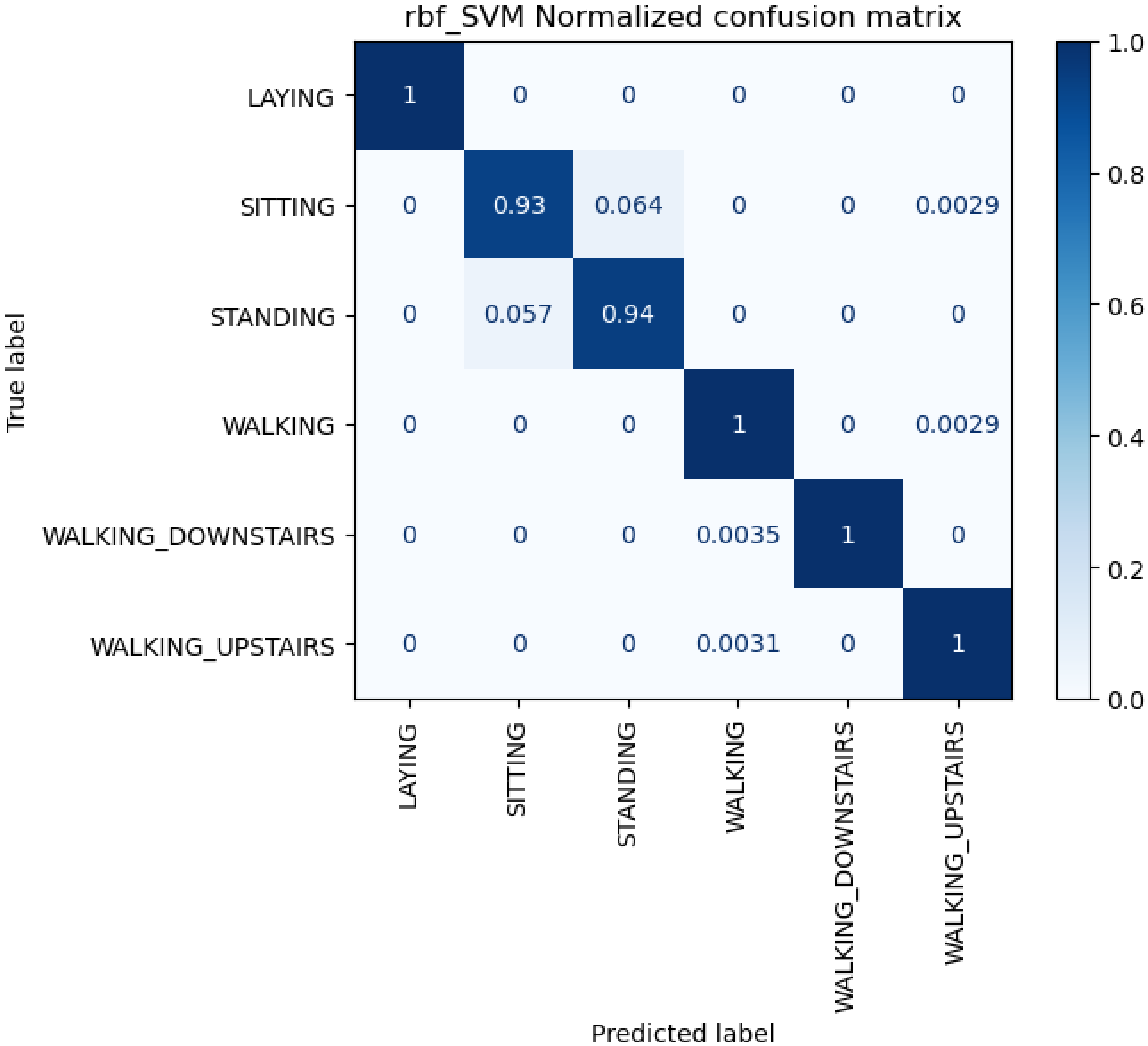
```
 [   0    0    0 343    1    0]
 [   0    0    0  99 165   25]
 [   0    0    0  53   22 250]]
```

```
D:\programs\anaconda\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureW
arning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mod
e` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will chang
e: the default value of `keepdims` will become False, the `axis` over which the statisti
c is taken will be eliminated, and the value None will no longer be accepted. Set `keepd
ims` to True or False to avoid this warning.
```

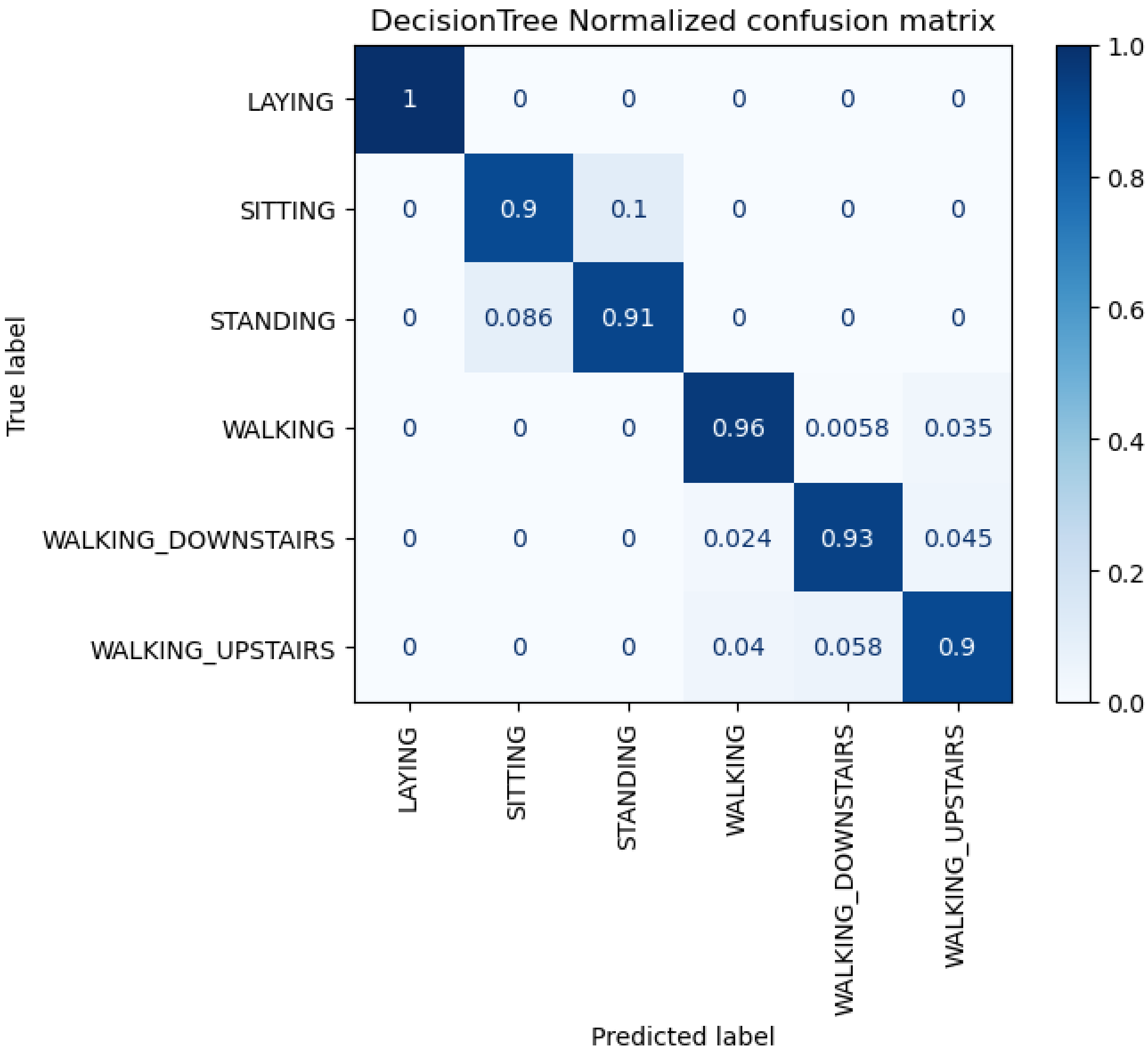


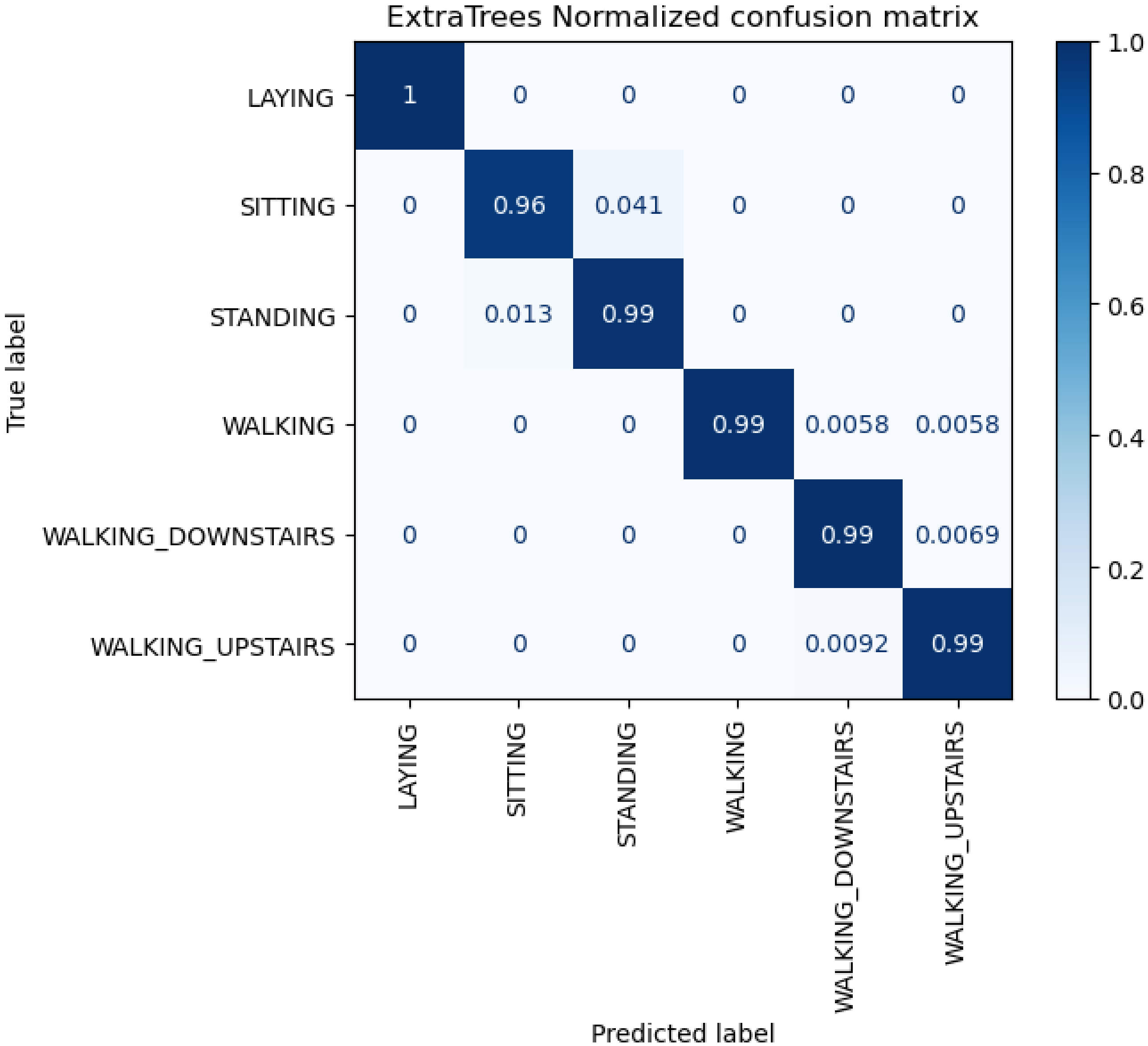

Linear_SVM Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1 0 0 0 0 0
0 0.96 0.038 0 0 0
0 0.029 0.97 0 0 0
0 0 0 1 0 0.0029
0 0 0 0 1 0
0 0 0 0 0 1
1.0
0.8
0.6
0.4
0.2
0.0

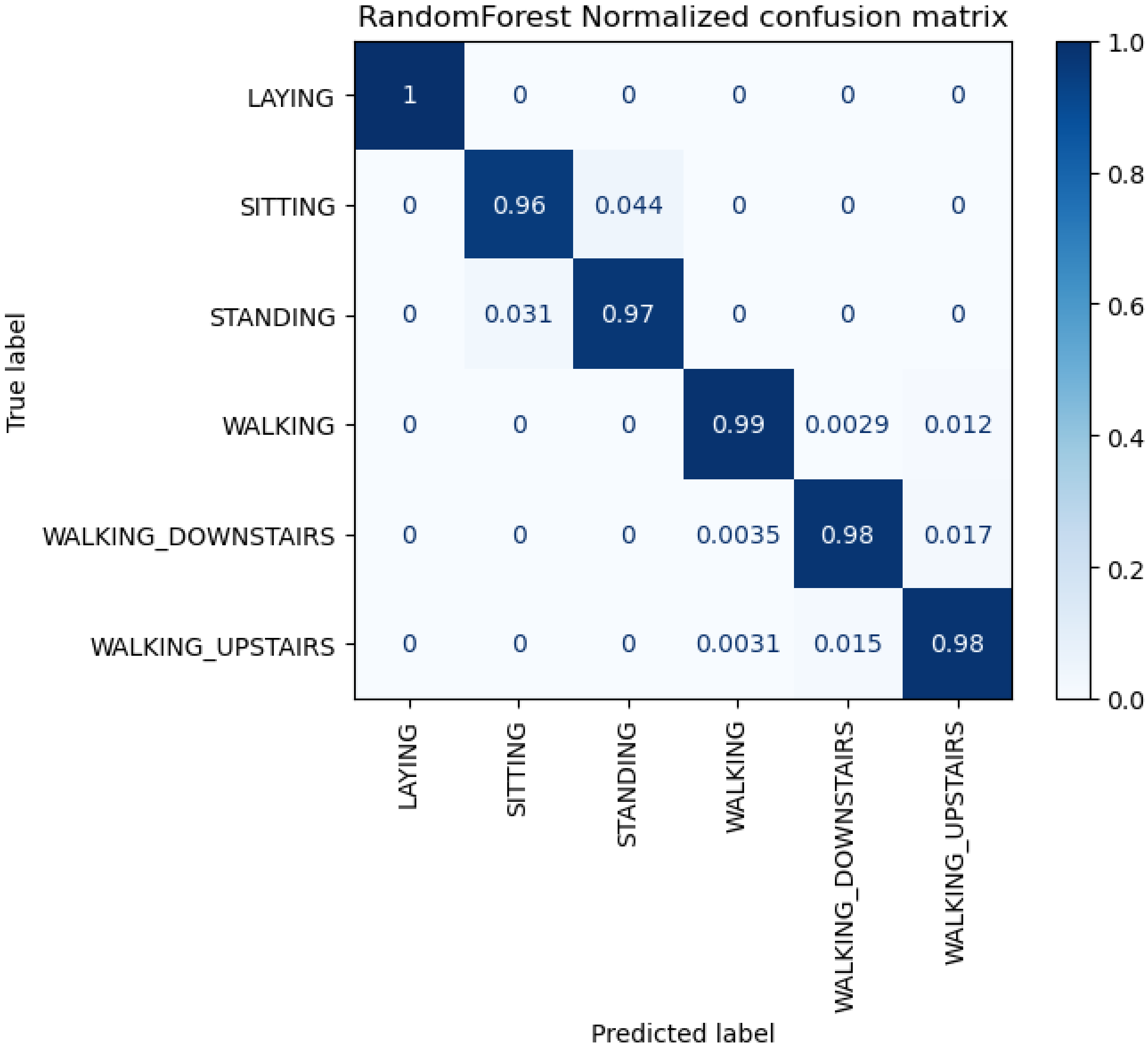

Poly_SVM Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1 0 0 0 0 0
0 0.98 0.023 0 0 0
0 0.018 0.98 0 0 0
0 0 0 0.99 0.012 0
0 0 0 0 1 0
0 0 0 0 0 1
1.0
0.8
0.6
0.4
0.2
0.0

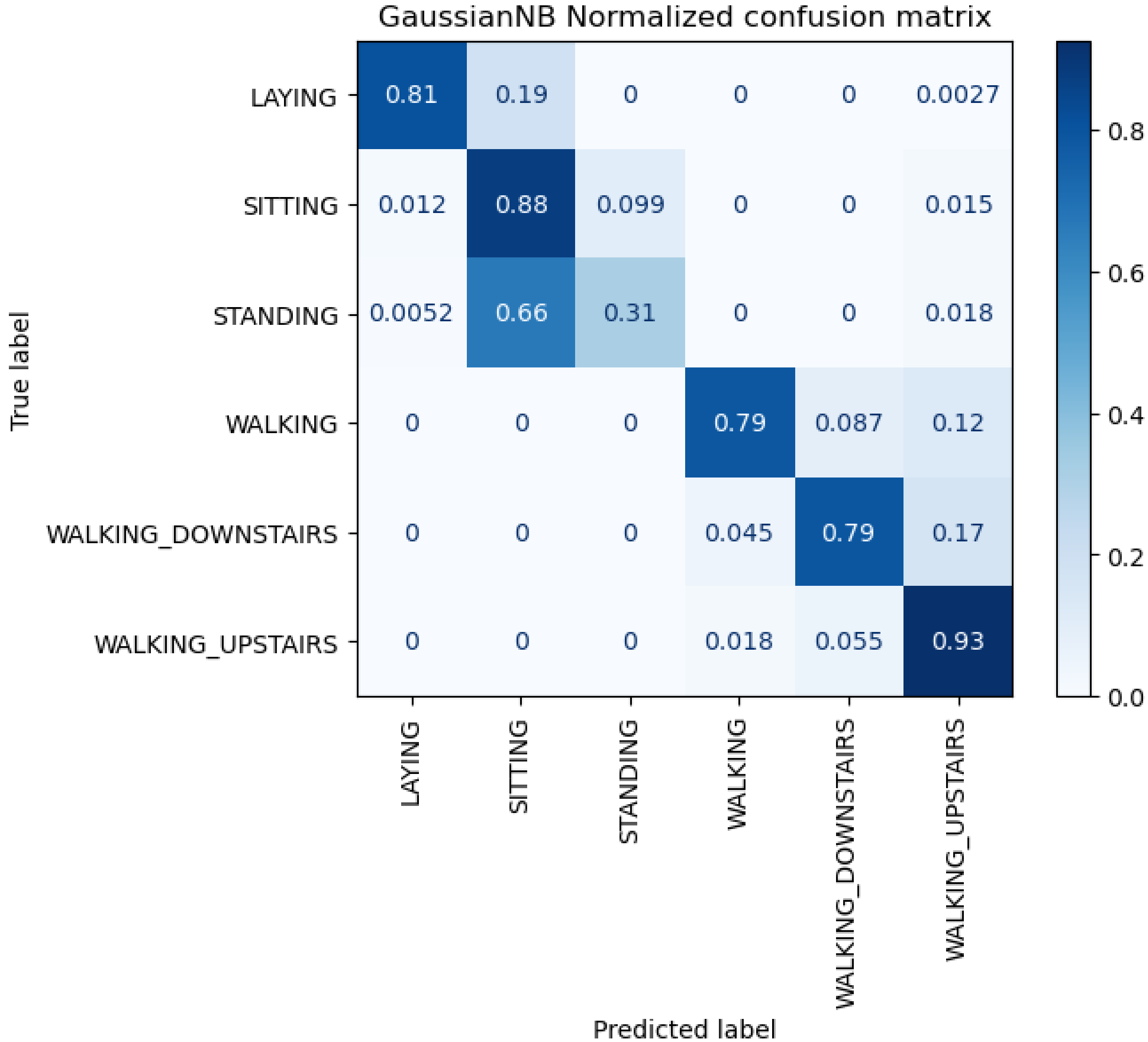


rbf_SVM Normalized confusion matrix
True label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1 0 0 0 0 0
0 0.93 0.064 0 0 0.0029
0 0.057 0.94 0 0 0
0 0 0 1 0 0.0029
0 0 0 0.0035 1 0
0 0 0 0.0031 0 1
1.0
0.8
0.6
0.4
0.2
0.0
Predicted label

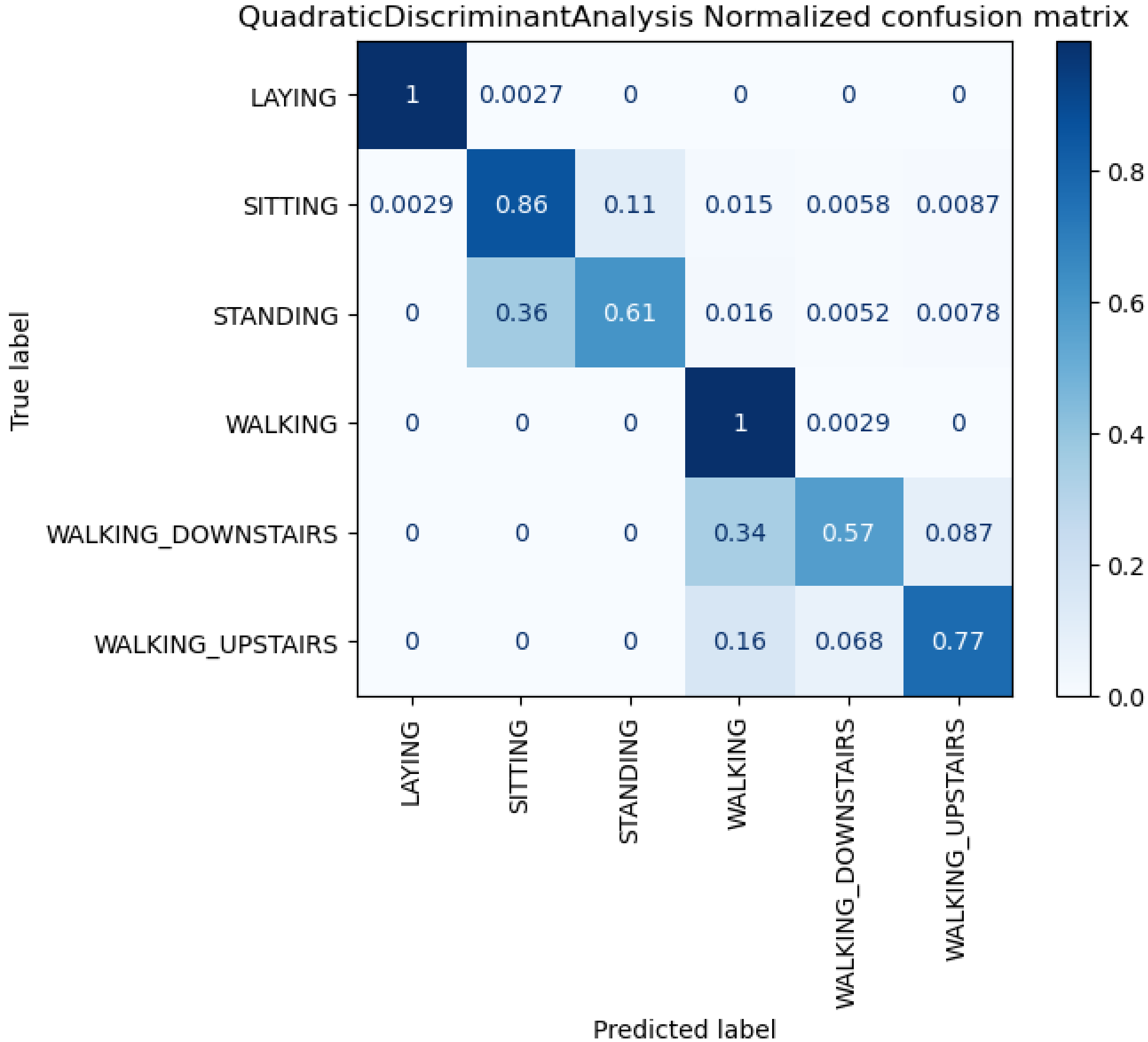

DecisionTree Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1 0 0 0 0 0
0 0.9 0.1 0 0 0
0 0.086 0.91 0 0 0
0 0 0 0.96 0.0058 0.035
0 0 0 0.024 0.93 0.045
0 0 0 0.04 0.058 0.9
1.0
0.8
0.6
0.4
0.2
0.0


ExtraTrees Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1 0 0 0 0 0
0 0.96 0.041 0 0 0
0 0.013 0.99 0 0 0
0 0 0 0.99 0.0058 0.0058
0 0 0 0 0.99 0.0069
0 0 0 0 0.0092 0.99
1.0
0.8
0.6
0.4
0.2
0.0


RandomForest Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1 0 0 0 0 0
0 0.96 0.044 0 0 0
0 0.031 0.97 0 0 0
0 0 0 0.99 0.0029 0.012
0 0 0 0.0035 0.98 0.017
0 0 0 0.0031 0.015 0.98
1.0
0.8
0.6
0.4
0.2
0.0



GaussianNB Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
0.81 0.19 0 0 0 0.0027
0.012 0.88 0.099 0 0 0.015
0.0052 0.66 0.31 0 0 0.018
0 0 0 0.79 0.087 0.12
0 0 0 0.045 0.79 0.17
0 0 0 0.018 0.055 0.93
0.0
0.2
0.4
0.6
0.8




```
knn accuracy : 97.43 %
lin accuracy : 98.79 %
pol accuracy : 99.08 %
rbf accuracy : 97.67 %
dtc accuracy : 93.45 %
etc accuracy : 98.64 %
rfc accuracy : 97.86 %
gnb accuracy : 73.98 %
qda accuracy : 80.68 %
```

```
First a not normalized version of the confusion graph is made, it's hard to see
the different accuracies in the not normalized graph. This is why a plot is
made of every normalized (meaning 100% correct = 1) algorithm. The perfect
model should have a score of 1 when all predictions are correct.
The accuracy can be seen underneath the last graph. This shows lineair, poly
and etc are the top 3 most accurate algorithms.
```

6. Fine-tune your models and combine them into a great solution.

Can you get better performance within a model? e.g if you use a KNN classifier how does it behave if you change K (k=3 vs k=5 vs k=?). Which parameters are here to tune in the chosen models?

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X,Y, stratify=Y, test_size=0.2, random_state=42
)

svclassifier_lin = SVC(kernel='linear', shrinking=False, tol=0.0001, class_weight=None, 
svclassifier_lin.fit(X_train, y_train)
y_pred_lin = svclassifier_lin.predict(X_test)

y_train_pred = svclassifier_lin.predict(X_train)
y_test_pred = svclassifier_lin.predict(X_test)

lin_train_accuracy = accuracy_score(y_train, y_train_pred)


# Test set performance
lin_test_accuracy = accuracy_score(y_test,y_test_pred)


print('Model performance for Training set')
print('- Accuracy: %s' % round(lin_train_accuracy*100,2), "%")

print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % round(lin_test_accuracy*100,2), "%")
```

```
Model performance for Training set
- Accuracy: 99.22 %
----------------------------------
Model performance for Test set
- Accuracy: 98.93 %
```

After altering the linear parameters the model has an accuracy of almost 99% this is more than the non optimized algorithm.

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X,Y, stratify=Y, test_size=0.2, random_state=42
)

svclassifier_pol = SVC(kernel='poly', gamma = 10, C = 1, degree=5)
svclassifier_pol.fit(X_train, y_train)
y_pred_pol = svclassifier_pol.predict(X_test)

y_train_pred = svclassifier_pol.predict(X_train)
y_test_pred = svclassifier_pol.predict(X_test)

pol_train_accuracy = accuracy_score(y_train, y_train_pred)


# Test set performance
pol_test_accuracy = accuracy_score(y_test,y_test_pred)


print('Model performance for Training set')
print('- Accuracy: %s' % round(pol_train_accuracy*100,2), "%")

print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % round(pol_test_accuracy*100,2), "%")
```

```
Model performance for Training set
- Accuracy: 100.0 %
----------------------------------
```

```
Model performance for Test set
- Accuracy: 99.22 %
```

"Gamma" is a parameter for non linear hyperplanes. The higher the gamma value it tries to exactly fit the training data set

"C" is the penalty parameter of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly.

"Degree" is a parameter used when kernel is set to ‘poly’. It’s basically the degree of the polynomial used to find the hyperplane to split the data.

This algorithm increases the accuracy at from 99.08% to 99.22%

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X,Y, stratify=Y, test_size=0.2, random_state=42
)

etc = ExtraTreesClassifier(n_estimators=500, min_samples_split=3, max_depth=500)
etc.fit(X_train, y_train)
y_pred_etc = etc.predict(X_test)

y_train_pred = etc.predict(X_train)
y_test_pred = etc.predict(X_test)

etc_train_accuracy = accuracy_score(y_train, y_train_pred)


# Test set performance
etc_test_accuracy = accuracy_score(y_test,y_test_pred)


print('Model performance for Training set')
print('- Accuracy: %s' % round(etc_train_accuracy*100,2), "%")

print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % round(etc_test_accuracy*100,2), "%")
```

```
Model performance for Training set
- Accuracy: 100.0 %
----------------------------------
Model performance for Test set
- Accuracy: 98.93 %
```

n_estimators is the number of trees to be used in the forest algorithm.

min_samples_split represents the minimum number of samples required to split an internal node. This can vary between considering at least one sample at each node to considering all of the samples at each node. When we increase this parameter, the tree becomes more constrained as it has to consider more samples at each node. Here we will vary the parameter from 10% to 100% of the samples

The first parameter to tune is max_depth. This indicates how deep the tree can be. The deeper the tree, the more splits it has and it captures more information about the data. We fit a decision tree with depths ranging from 1 to 32 and plot the training and test auc scores.

The accuracy of the extra trees algorithm increased from 98.64% to 98.93%

In [11]:
```python
estimator_list = [
    ('lin',svclassifier_lin),
    ('pol',svclassifier_pol),
    ('etc',etc) ]
stack_model = StackingClassifier(
    estimators=estimator_list, final_estimator=LogisticRegression()
)
stack_model.fit(X_train, y_train)
y_train_pred = stack_model.predict(X_train)
y_test_pred = stack_model.predict(X_test)


stack_model_train_accuracy = accuracy_score(y_train, y_train_pred)
stack_model_test_accuracy = accuracy_score(y_test, y_test_pred)


print('Model performance for Training set')
print('- Accuracy: %s' % round(stack_model_train_accuracy*100,2), "%")

print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % round(stack_model_test_accuracy*100,2), "%")
```

```
D:\programs\anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:814: Convergenc
eWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
Model performance for Training set
- Accuracy: 100.0 %
----------------------------------
Model performance for Test set
- Accuracy: 99.66 %
```

```
combining 3 algorithms into one model to have the highest accuracy.
```

7. Present your solution.

Explain why you would choose for a specific model

```
Total code of the assignment is shown above

This model is chosen to get the highest accuracy. combining the 3 most acurate
algorithms together will be sufficient to have an accuracy of 99.66%
```

8. Launch, monitor, and maintain your system.

Deployment we will do in the next assignment!

9. Additional Questions

- Explain which classes should be easy / challenging to classify based on your 2/3D plots the data?
- Explain what specifics you did to this dataset for preparing your data?
- Explain why you think that your chosen algorithm outperforms the rest?

```
Based on the 2d and 3d plots, standing and sitting should be hard to identify.
Its very likely that the movement will be the same. laying on the other hand
will be very easy to identify among the others because of its seporated group.

We are splitting the data in input and output while also randomizing this data.
This causes the ai to discard the idea to replay data in the same order.

Our model should perform better than a singular algorithm. this is also proven
by having the highest accuracy of all separate algorithms. The SVC models
should be the best at identifying groups of similar data. This is also
interpreted in our model.
```