ACC and GYR - physical activity detection

Usairim Isani - muisan@utu.fi

Joni Rajamaki - t09jraja@utu.fi

In [ ]:
```
%xmode Minimal
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import glob
import tsfel
import random

# Machine learning packages - scikit-learn
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeClassifier
```

```
Exception reporting mode: Minimal
```

In [ ]:
```
import warnings
warnings.filterwarnings(action='ignore')
```

In [ ]:
```
data_path = "./Data"
train_data_path = f'{data_path}/Train'
test_data_path = f'{data_path}/Test'
extracted_features_path = "./Features"
```

In [ ]:
```
train_dir = os.listdir(train_data_path)
```

Testing with single file

In [ ]:
```
test_df = pd.read_csv("./Data/Train/standing/sample_ID10_exp19_125.csv")
cfg_file = tsfel.get_features_by_domain(json_path="./features.json")
```

In [ ]:
```
tsfel.get_number_features(cfg_file)
extracted_features = tsfel.time_series_features_extractor(
    cfg_file, test_df, fs=50, window_size=250, verbose=0)
extracted_features.to_csv("./test.csv")
print(len(extracted_features.columns)/6)
extracted_features
```

Progress: 0% Complete

```
149.0
```

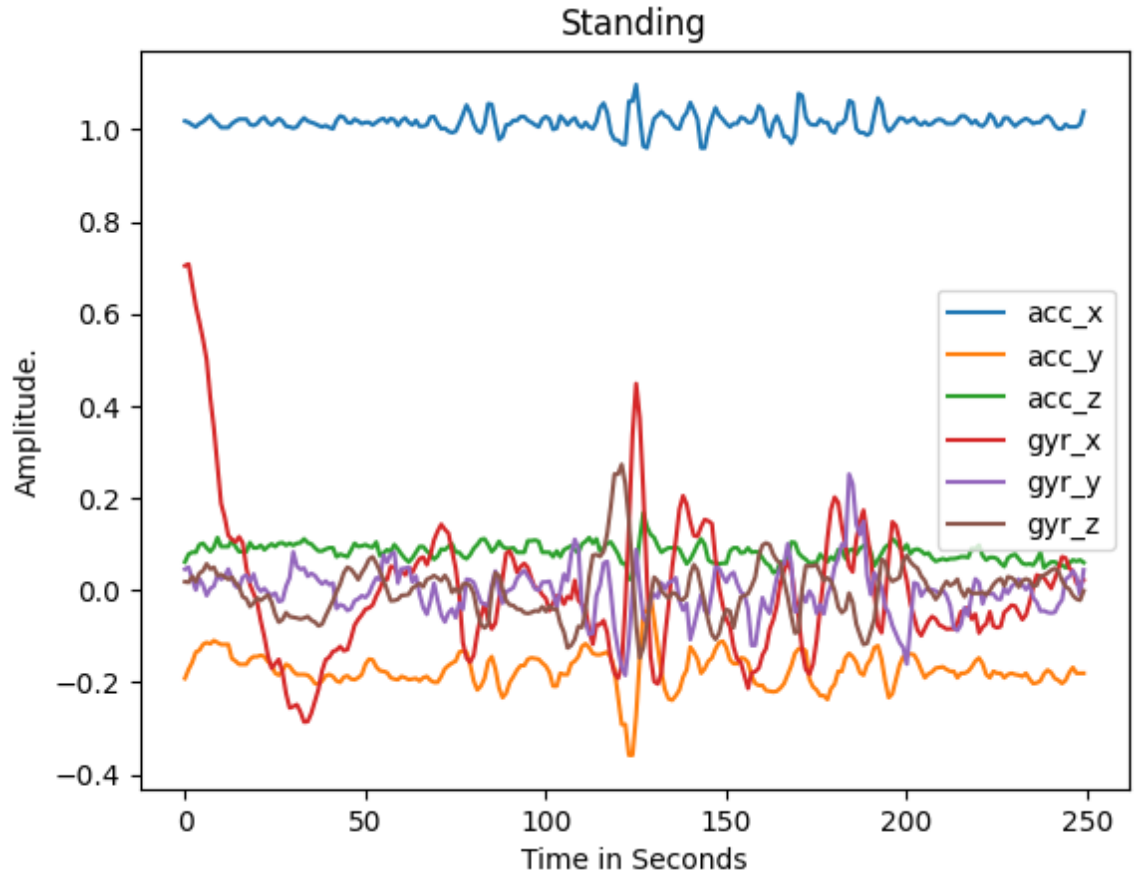
Out[ ]:

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Entropy	0_FFT mean coefficient_0	0_FFT mean coefficient_1	0_FFT m coefficien
0	257.983808	5.057764	257.983808	0.64478	9.963536e-08	0.000004	0.00(

1 rows × 894 columns

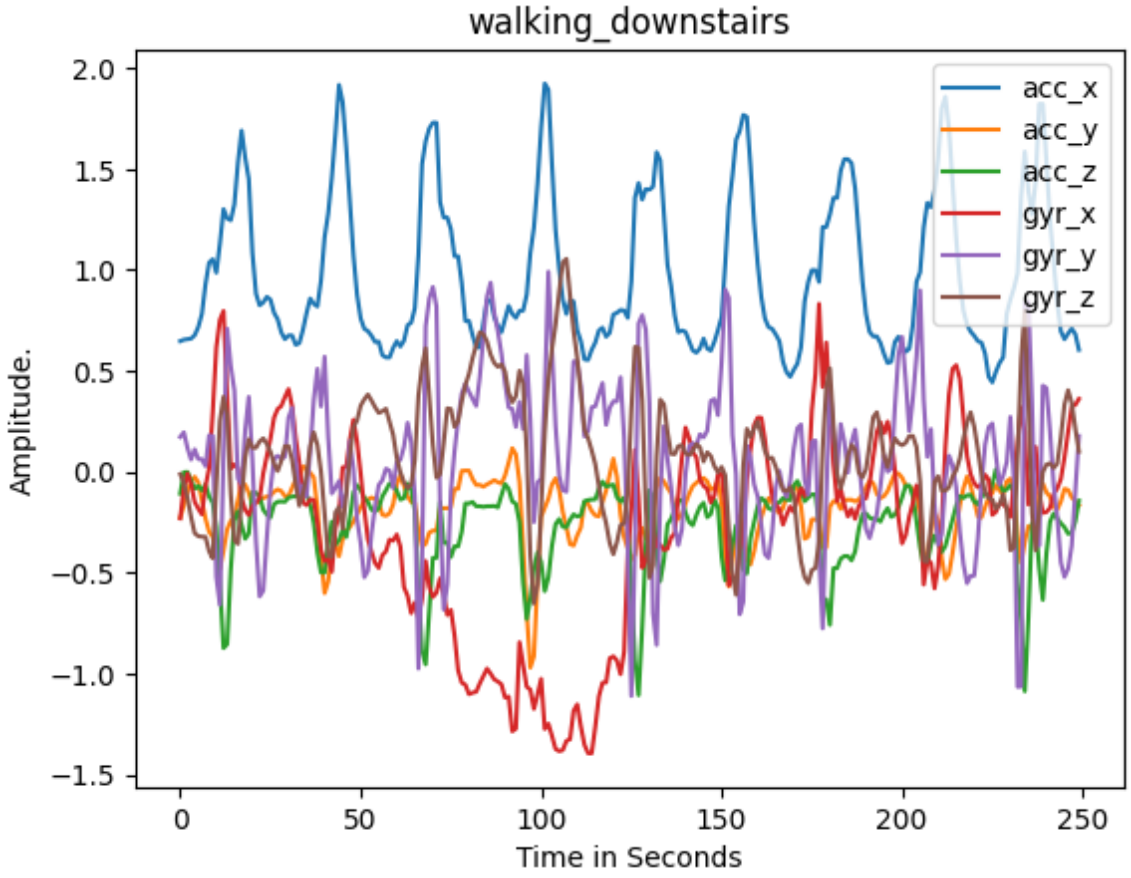
In [ ]:
```
ax = test_df.plot.line()
ax.set_title("Standing")
ax.set_xlabel("Time in Seconds")
ax.set_ylabel("Amplitude.")
```

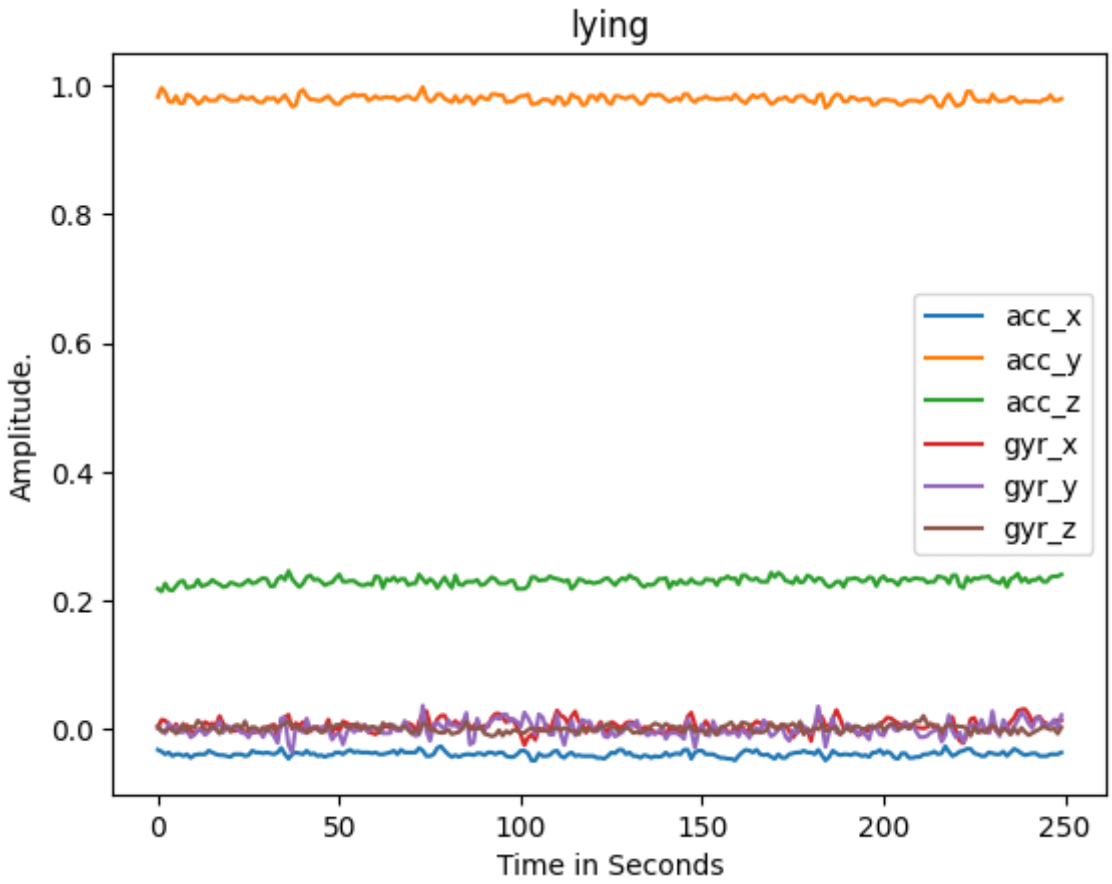
Out[ ]:
```
Text(0, 0.5, 'Amplitude.')
```

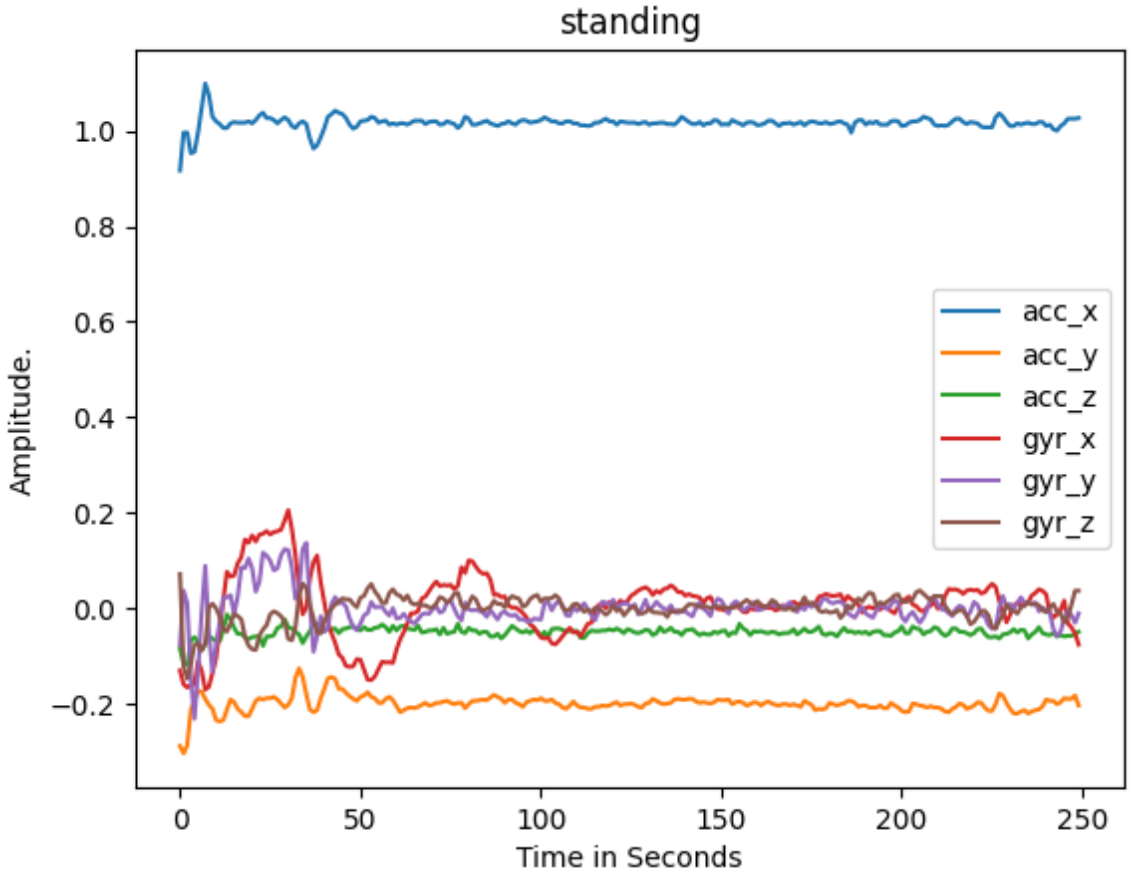


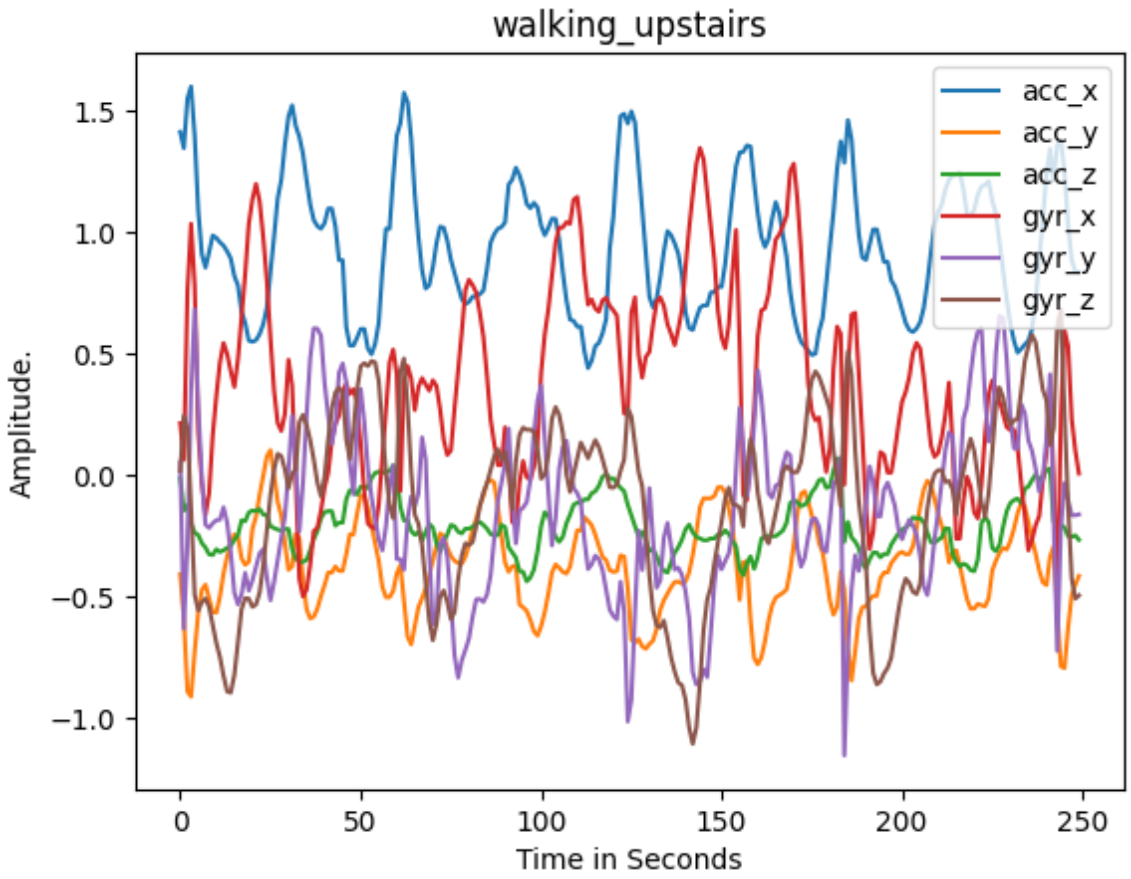
Plot for all activities in the dataset.

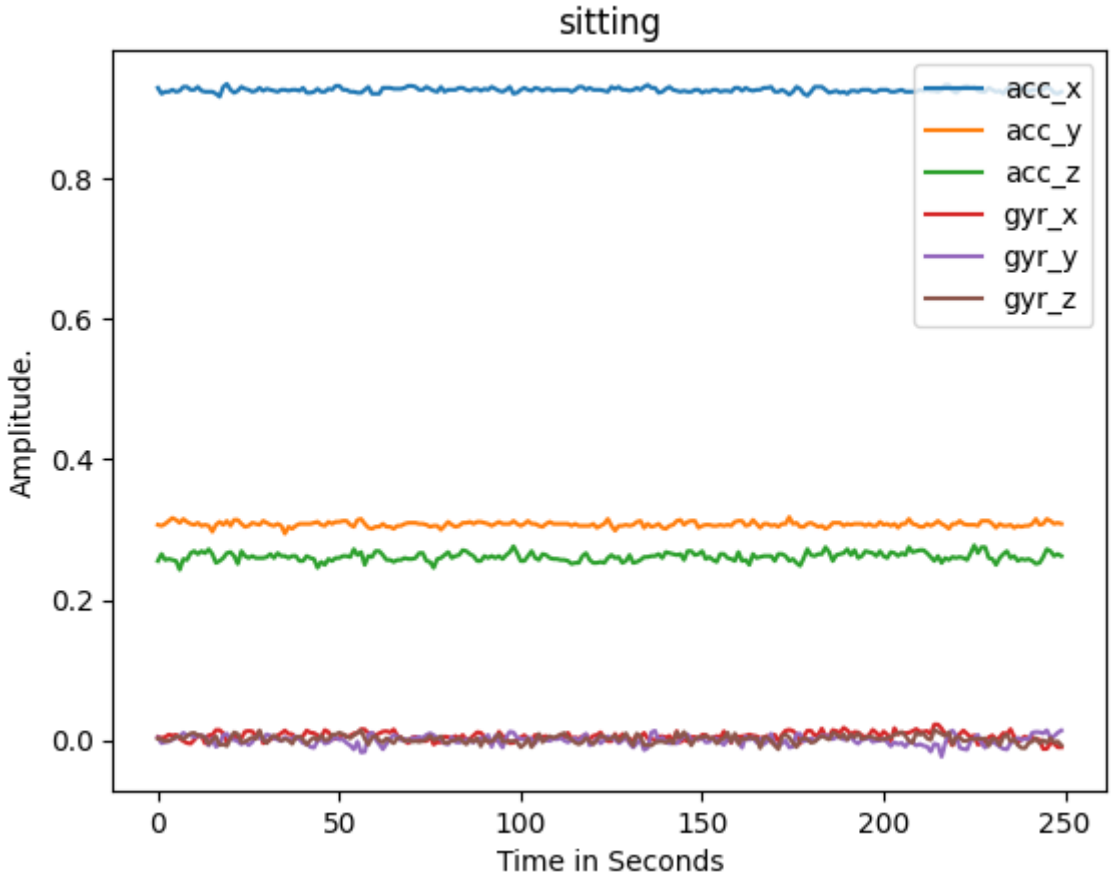
In [ ]:
```
samples = []
for folder in os.listdir(train_data_path):
    file = random.choice(os.listdir(f'{train_data_path}/{folder}'))
    sample_df = pd.read_csv(f'{train_data_path}/{folder}/{file}')
    ax = sample_df.plot.line()
    ax.set_title(folder)
    ax.set_xlabel("Time in Seconds")
    ax.set_ylabel("Amplitude.")
```

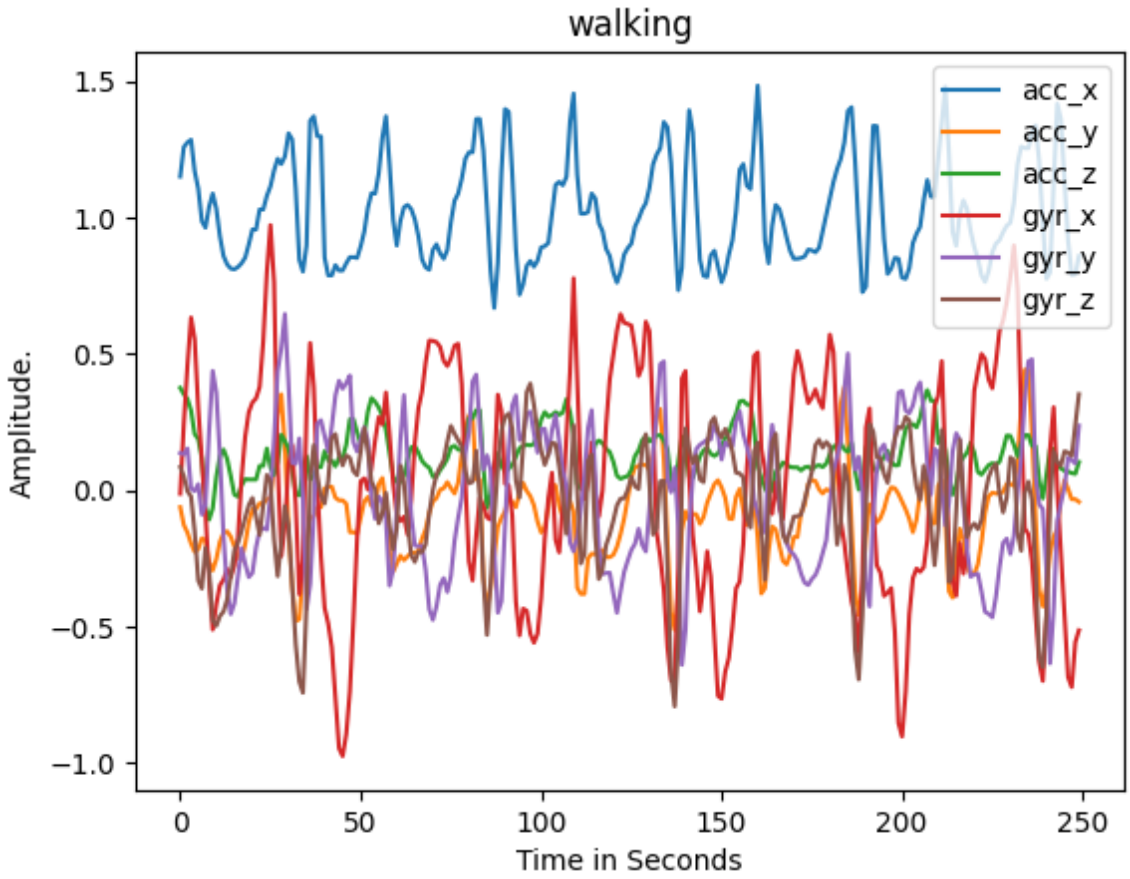

walking_downstairs
Amplitude.
Time in Seconds
acc_x
acc_y
acc_z
gyr_x
gyr_y
gyr_z
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
50
100
150
200
250


lying
Amplitude.
Time in Seconds
acc_x
acc_y
acc_z
gyr_x
gyr_y
gyr_z
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250


standing
Amplitude.
Time in Seconds
acc_x
acc_y
acc_z
gyr_x
gyr_y
gyr_z


walking_upstairs
Amplitude.
Time in Seconds
acc_x
acc_y
acc_z
gyr_x
gyr_y
gyr_z





In [ ]:
```
def read_data(data_path):
    folders = {}
    for folder in os.listdir(data_path):
        samples = {}
        for file in os.listdir(f'{data_path}/{folder}'):
```

```
            with open(f'{data_path}/{folder}/{file}') as f:
                sample_df = pd.read_csv(f)
                samples[file]=sample_df
        folders[folder] = samples
    return folders
```

In [ ]:
```
train_data = read_data(train_data_path)
test_data = read_data(test_data_path)
```

In [ ]:
```
cfg_file = tsfel.get_features_by_domain(json_path="./features.json")
tsfel.get_number_features(cfg_file)
```

Out[ ]: 279

Selected Features

The features are selected by setting the flag " `use` " `yes` or `no` in the `features.json` file.

- Spectral
  - FFT mean coefficient
  - Fundamental frequency
  - Max power spectrum (Max PSD)
  - Power bandwidth (PSD Bandwidth)
  - Spectral entropy
  - Spectral variation
  - Wavelet absolute mean (CWT absolute mean value of each wavelet scale)
  - Wavelet energy (CWT energy of each wavelet scale)
  - Wavelet entropy (CWT entropy of each wavelet scale)
- Statistical
  - Absolute energy
  - Entropy
  - Inter-quartile range
  - Kurtosis
  - Mean
  - Mean absolute deviation
  - Root mean square
  - Variance
- Temporal
  - Area under the curve
  - Autocorrelation
  - Mean absolute diff
  - Zero crossing rate

Extracting features for all the subjects and saving into a `train` and `test` csv for better usage.

In [ ]:
```
def extract_features(data, out):
    for label in data:
        for subject in data[label]:
            subject_df = data[label][subject]
            extracted_features = tsfel.time_series_features_extractor(cfg
            extracted_features.insert(0,"Label", label)
            extracted_features.insert(0,"Subject",subject)
            extracted_features.to_csv(f'{extracted_features_path}/{out}/{
```

In [ ]:
```
extract_features(train_data, "Train")
extract_features(test_data, "Test")
```

In [ ]:
```
def data_to_csv(path, name):
    all_files = glob.glob(f'{extracted_features_path}/{path}/*.csv')
    extracted_features_df = pd.concat((pd.read_csv(f) for f in all_files)
    extracted_features_df.to_csv(f'{extracted_features_path}/{name}.csv')
```

In [ ]:
```
data_to_csv("Train", "training")
data_to_csv("Test", "testing")
```

In [ ]:
```
training_features = pd.read_csv(f'{extracted_features_path}/training.csv'
testing_features = pd.read_csv(f'{extracted_features_path}/testing.csv')
```

In [ ]:
```
training_features.sample(10)
```

Out[ ]:

	Unnamed: 0.1	Unnamed: 0	Subject	Label	0_Absolute energy	0_A ui cu
78	78	0	sample_ID13_exp27_181.csv	walking	262.315040	4.953
1069	1069	0	sample_ID18_exp37_217.csv	walking_upstairs	227.293936	4.650
547	547	0	sample_ID16_exp33_218.csv	walking	261.584783	4.997
1034	1034	0	sample_ID5_exp9_48.csv	walking_downstairs	299.643598	5.131
658	658	0	sample_ID3_exp6_47.csv	walking	262.292354	4.960
772	772	0	sample_ID3_exp5_41.csv	walking	257.036338	4.980
157	157	0	sample_ID7_exp14_92.csv	standing	258.061801	5.059
73	73	0	sample_ID5_exp9_68.csv	walking	267.054478	5.072
809	809	0	sample_ID18_exp36_252.csv	standing	259.462139	5.073
1000	1000	0	sample_ID12_exp24_163.csv	walking	283.951015	5.109

10 rows × 898 columns

In [ ]:
```
testing_features.sample(10)
```

Out[ ]:

	Unnamed: 0.1	Unnamed: 0	Subject	Label	0_Absolute energy	0_A un cu
417	417	0	sample_ID30_exp60_359.csv	walking_upstairs	263.587479	5.0073
542	542	0	sample_ID27_exp55_308.csv	walking_downstairs	268.535091	4.8203
463	463	0	sample_ID25_exp51_393.csv	standing	259.006916	5.0689
276	276	0	sample_ID30_exp61_418.csv	walking	273.402432	5.0910
345	345	0	sample_ID23_exp46_251.csv	walking_downstairs	285.466532	5.0063
376	376	0	sample_ID22_exp45_338.csv	standing	256.629995	5.0450
785	785	0	sample_ID23_exp46_341.csv	lying	1.415922	0.3731
526	526	0	sample_ID25_exp50_337.csv	walking	242.741896	4.8617
217	217	0	sample_ID28_exp56_315.csv	walking_downstairs	279.757577	4.9021
588	588	0	sample_ID28_exp56_441.csv	standing	254.887912	5.0270

10 rows × 898 columns

In [ ]:
```
# Separate the labels to their own dataframe
testing_labels = testing_features["Label"]
training_labels = training_features["Label"]

# Drop labels from rest of data
testing_features_no_labels = testing_features.drop(["Label", "Subject", "
training_features_no_labels = training_features.drop(["Label","Subject","

# Check data
testing_features_no_labels.head(10)
training_features_no_labels.head(10)
```

Out[ ]:

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Entropy	0_FFT mean coefficient_0	0_FFT mean coefficient_1	0_FFT m coefficien
0	258.546808	5.064347	258.546808	0.469982	8.930358e-07	0.000001	1.804824
1	248.411002	4.845931	248.411002	0.915121	1.130804e-05	0.000018	5.730127
2	308.463970	4.970222	308.463970	0.947624	5.951605e-06	0.000046	6.574936
3	262.718817	4.963459	262.718817	0.935080	2.204985e-04	0.000032	2.183648
4	175.813698	4.175834	175.813698	0.460512	2.385487e-06	0.000002	2.727445
5	260.968683	5.087459	260.968683	0.638950	5.746562e-06	0.000006	1.755860
6	279.114259	4.888556	279.114259	0.954295	1.805066e-03	0.000115	9.058675
7	266.083682	4.967153	266.083682	0.932510	1.365879e-04	0.000333	3.249240
8	0.114606	0.072583	0.114606	0.612388	1.726944e-05	0.000168	5.577407
9	1.527849	0.384806	1.527849	0.578014	7.811763e-06	0.000175	4.145391

10 rows × 894 columns

In [ ]:
```
# Standardize data
scaler = StandardScaler()
training_features_scaled = scaler.fit_transform(training_features_no_labe
testing_features_scaled = scaler.transform(testing_features_no_labels)
```

In [ ]:
```
training_features.shape
```

Out[ ]:
```
(1170, 898)
```

In [ ]:
```
training_labels.shape
```

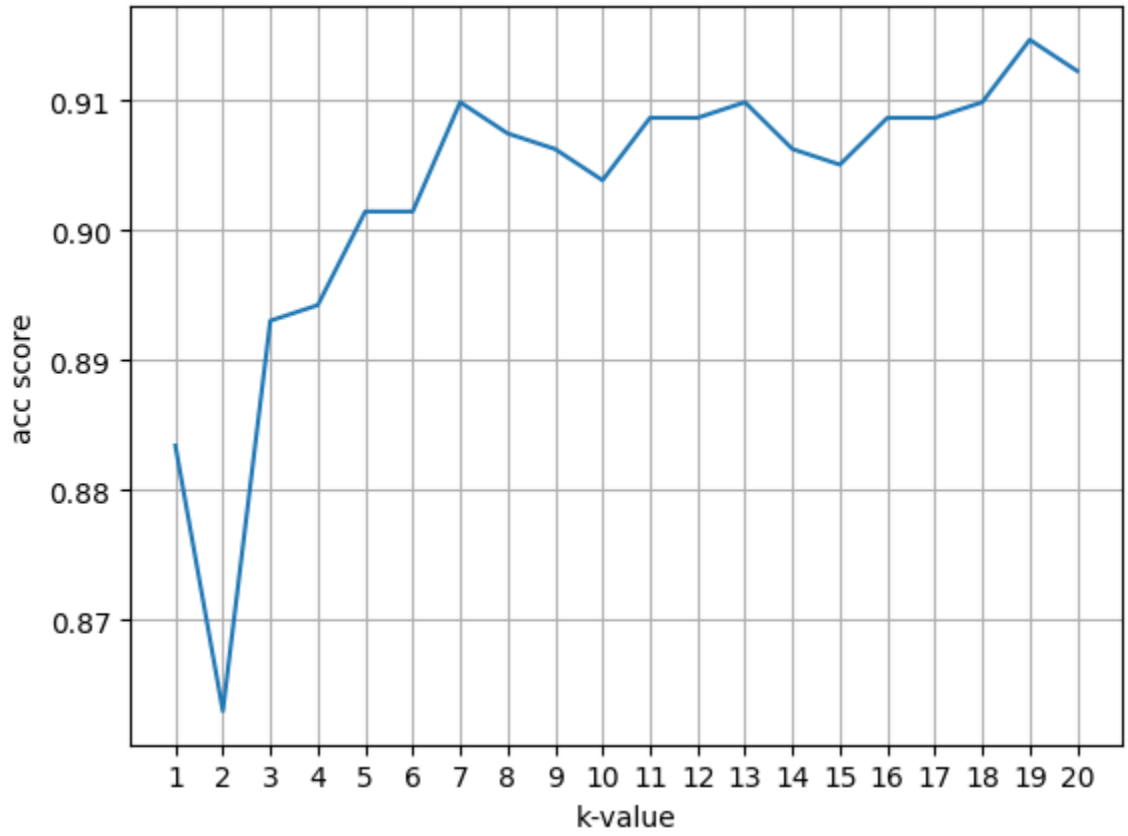
Out[ ]:
```
(1170,)
```

Accuracy pair group which parts was it better Compare methods ML, affects of hyperparameter on results

In [ ]:
```
# Trying different k-values (0-20)
accuracies = []
for k in range(1,21):
    train_knn = KNeighborsClassifier(n_neighbors=k) #define the model
    train_knn.fit(training_features_scaled, training_labels) #train/fit m
    predictions_knn = train_knn.predict(testing_features_scaled) #predict

    #print(metrics.confusion_matrix(testing_labels, predictions_knn)) #pr
    acc = metrics.accuracy_score(testing_labels, predictions_knn) #get ac
    #print("accuracy:",acc) #print accuracy score
    accuracies.append(acc)
```

```
plt.plot(range(1,21),accuracies,)
plt.ylabel('acc score')
plt.xlabel('k-value')
plt.xticks(range(1,21))
plt.grid()
```



In [ ]:
```
train_knn = KNeighborsClassifier(n_neighbors=19) #define the model
train_knn.fit(training_features_scaled, training_labels) #train/fit model
predictions_knn = train_knn.predict(testing_features_scaled) #predictions

confusion_matrix = metrics.confusion_matrix(testing_labels, predictions_k
acc = metrics.accuracy_score(testing_labels, predictions_knn) #get accura

print("Confusion Matrix:")
print(confusion_matrix) #print confusion matrix with labels_train vs. the
print("Accuracy: ",acc) #print accuracy score
```

```
Confusion Matrix:
[[ 81   3   0   0   0   0]
 [  0 155  14   0   0   0]
 [  0  21 167   0   0   0]
 [  0   0   5 139   0   0]
 [  0   0   0   9 102   8]
 [  0   0   0   2   9 117]]
Accuracy:  0.9146634615384616
```

In [ ]:
```
# Train RidgeClassifier
ridge = RidgeClassifier()
ridge.fit(training_features_scaled, training_labels)
prediction_ridge = ridge.predict(testing_features_scaled)

# confusion matrix with labels_train vs. the predictions
confusion_matrix = metrics.confusion_matrix(testing_labels, prediction_ri
```

```
acc = metrics.accuracy_score(
    testing_labels, prediction_ridge)  # get accuracy score

print("Confusion Matrix:")

print(confusion_matrix)
print("Accuracy: ", acc)  # print accuracy score
```

```
Confusion Matrix:
[[ 81   3   0   0   0   0]
 [  0 163   6   0   0   0]
 [  0  12 176   0   0   0]
 [  0  12  12 117   1   2]
 [  0   5  35   4  67   8]
 [  0  16  15   0   0  97]]
Accuracy:  0.8425480769230769
```

In [ ]:
```
print("F1 Score for Ridge Regression : ", metrics.f1_score(
    testing_labels, prediction_ridge, average='macro'))

print("F1 Score for KNN : ", metrics.f1_score(
    testing_labels, predictions_knn, average='macro'))
```

```
F1 Score for Ridge Regression :  0.8466093427333665
F1 Score for KNN :  0.9205178077773293
```

Conclusion

By selecting and extracting appropriate features from the activity sensors we can correctly predict the activity done. And KNN Classifier with k=19, performs the best with accuracy of 91% and an F1 Score of 0.92.

References

TSFEL was published in Barandas, Marília and Folgado, Duarte, et al. “TSFEL: Time Series Feature Extraction Library.” SoftwareX 11 (2020). https://doi.org/10.1016/j.softx.2020.100456