

```
In [1]: !wget https://ora.ox.ac.uk/objects/uuid:99d7c092-d865-4a19-b096-cc16440cd
--2024-11-06 06:24:02-- https://ora.ox.ac.uk/objects/uuid:99d7c092-d865-4a19-b096-cc16440cd001/files/rpr76f381b
Resolving ora.ox.ac.uk (ora.ox.ac.uk)... 129.67.246.216
Connecting to ora.ox.ac.uk (ora.ox.ac.uk)|129.67.246.216|:443... connecte
d.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'rpr76f381b'

rpr76f381b          [          <=> ]   6.43G  27.0MB/s   in 4m
11s

2024-11-06 06:28:14 (26.3 MB/s) - 'rpr76f381b' saved [6902652480]
```

```
In [2]: !unzip rpr76f381b -d data >> /dev/null
```

```
In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import scipy.signal as signal
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
```

```
In [4]: def extract_features(xyz, sample_rate=100):
''' Extract commonly used HAR time-series features. xyz is a window o

feats = {}

x, y, z = xyz.T

feats['xmin'], feats['xq25'], feats['xmed'], feats['xq75'], feats['xm
x, (0, .25, .5, .75, 1))
feats['ymin'], feats['yq25'], feats['ymed'], feats['yq75'], feats['ym
y, (0, .25, .5, .75, 1))
feats['zmin'], feats['zq25'], feats['zmed'], feats['zq75'], feats['zm
z, (0, .25, .5, .75, 1))

with np.errstate(divide='ignore', invalid='ignore'): # ignore div by
# xy, xy, zx correlation
feats['xycorr'] = np.nan_to_num(np.corrcoef(x, y)[0, 1])
feats['yzcorr'] = np.nan_to_num(np.corrcoef(y, z)[0, 1])
feats['zxcorr'] = np.nan_to_num(np.corrcoef(z, x)[0, 1])

v = np.linalg.norm(xyz, axis=1)

feats['min'], feats['q25'], feats['med'], feats['q75'], feats['max']
v, (0, .25, .5, .75, 1))

with np.errstate(divide='ignore', invalid='ignore'): # ignore div by
# 1s autocorrelation
feats['corr1s'] = np.nan_to_num(np.corrcoef(
v[:-sample_rate], v[sample_rate:]))[0, 1]
```

```

# Angular features
feats.update(angular_features(xyz, sample_rate))

# Spectral features
feats.update(spectral_features(v, sample_rate))

# Peak features
feats.update(peak_features(v, sample_rate))

return feats

def spectral_features(v, sample_rate):
    """ Spectral entropy, 1st & 2nd dominant frequencies """

    feats = {}

    # Spectrum using Welch's method with 3s segment length
    # First run without detrending to get the true spectrum
    freqs, powers = signal.welch(v, fs=sample_rate,
                                  nperseg=3 * sample_rate,
                                  noverlap=2 * sample_rate,
                                  detrend=False,
                                  average='median')

    with np.errstate(divide='ignore', invalid='ignore'): # ignore div by
        feats['pentropy'] = np.nan_to_num(stats.entropy(powers + 1e-16))

    # Spectrum using Welch's method with 3s segment length
    # Now do detrend to focus on the relevant freqs
    freqs, powers = signal.welch(v, fs=sample_rate,
                                  nperseg=3 * sample_rate,
                                  noverlap=2 * sample_rate,
                                  detrend='constant',
                                  average='median')

    peaks, _ = signal.find_peaks(powers)
    peak_powers = powers[peaks]
    peak_freqs = freqs[peaks]
    peak_ranks = np.argsort(peak_powers)[::-1]
    if len(peaks) >= 2:
        feats['f1'] = peak_freqs[peak_ranks[0]]
        feats['f2'] = peak_freqs[peak_ranks[1]]
        feats['p1'] = peak_powers[peak_ranks[0]]
        feats['p2'] = peak_powers[peak_ranks[1]]
    elif len(peaks) == 1:
        feats['f1'] = feats['f2'] = peak_freqs[peak_ranks[0]]
        feats['p1'] = feats['p2'] = peak_powers[peak_ranks[0]]
    else:
        feats['f1'] = feats['f2'] = 0
        feats['p1'] = feats['p2'] = 0

    return feats

def peak_features(v, sample_rate):
    """ Features of the signal peaks. A proxy to step counts. """

    feats = {}
    u = butterfilt(v, (.6, 5), fs=sample_rate)

```

```

peaks, peak_props = signal.find_peaks(
    u, distance=0.2 * sample_rate, prominence=0.25)
feats['numPeaks'] = len(peaks)
if len(peak_props['prominences']) > 0:
    feats['peakPromin'] = np.median(peak_props['prominences'])
else:
    feats['peakPromin'] = 0

return feats

def angular_features(xyz, sample_rate):
    """ Roll, pitch, yaw.
    Hip and Wrist Accelerometer Algorithms for Free-Living Behavior
    Classification, Ellis et al.
    """

    feats = {}

    # Raw angles
    x, y, z = xyz.T

    roll = np.arctan2(y, z)
    pitch = np.arctan2(x, z)
    yaw = np.arctan2(y, x)

    feats['avgroll'] = np.mean(roll)
    feats['avgpitch'] = np.mean(pitch)
    feats['avgyaw'] = np.mean(yaw)
    feats['sdroll'] = np.std(roll)
    feats['sdpitch'] = np.std(pitch)
    feats['sdyaw'] = np.std(yaw)

    # Gravity angles
    xyz = butterfilt(xyz, 0.5, fs=sample_rate)

    x, y, z = xyz.T

    roll = np.arctan2(y, z)
    pitch = np.arctan2(x, z)
    yaw = np.arctan2(y, x)

    feats['rollg'] = np.mean(roll)
    feats['pitchg'] = np.mean(pitch)
    feats['yawg'] = np.mean(yaw)

    return feats

def butterfilt(x, cutoffs, fs, order=10, axis=0):
    nyq = 0.5 * fs
    if isinstance(cutoffs, tuple):
        hicut, lowcut = cutoffs
        if hicut > 0:
            btype = 'bandpass'
            Wn = (hicut / nyq, lowcut / nyq)
        else:
            btype = 'low'
            Wn = lowcut / nyq
    else:

```

```

        btype = 'low'
        Wn = cutoffs / nyq
        sos = signal.butter(order, Wn, btype=btype, analog=False, output='sos')
        y = signal.sosfiltfilt(sos, x, axis=axis)
        return y

def get_feature_names():
    """ Hacky way to get the list of feature names """

    feats = extract_features(np.zeros((1000, 3)), 100)
    return list(feats.keys())

```

```

In [5]: data = pd.read_csv(
        "data/capture24/P001.csv.gz", compression="gzip",
        index_col="time", parse_dates=["time"],
        dtype={"x": "f4", "y": "f4", "z": "f4", "annotation": "string"}
    )
    data

```

```

Out [5]:

```

	x	y	z	annotation
time				
2016-11-13 02:18:00.000	-0.466690	-0.533341	0.658472	7030 sleeping;MET 0.95
2016-11-13 02:18:00.010	-0.466690	-0.533341	0.658472	7030 sleeping;MET 0.95
2016-11-13 02:18:00.020	-0.466690	-0.533341	0.658472	7030 sleeping;MET 0.95
2016-11-13 02:18:00.030	-0.466690	-0.533341	0.658472	7030 sleeping;MET 0.95
2016-11-13 02:18:00.040	-0.466690	-0.533341	0.658472	7030 sleeping;MET 0.95
...	...	...	...	...
2016-11-14 06:07:59.960	0.049416	-0.797846	0.565700	7030 sleeping;MET 0.95
2016-11-14 06:07:59.970	0.049416	-0.782285	0.565700	7030 sleeping;MET 0.95
2016-11-14 06:07:59.980	0.049416	-0.782285	0.565700	7030 sleeping;MET 0.95
2016-11-14 06:07:59.990	0.049416	-0.782285	0.565700	7030 sleeping;MET 0.95
2016-11-14 06:08:00.000	0.049416	-0.782285	0.565700	7030 sleeping;MET 0.95

10020001 rows x 4 columns

```

In [6]: annot = pd.read_csv("data/capture24/annotation-label-dictionary.csv", index_col=0)
    annot

```

Out [6]:

label:WilletttsSpecific2018label:WilletttsMET2018

annotation			
7030 sleeping;MET 0.95	sleep		sleep
occupation;office and administrative support;11580 office/computer work general;MET 1.5	sitting		sitstand+lowactivity
home activity;household chores;preparing meals/cooking/washing dishes;5035 kitchen activity general cooking/washing/dishes/cleaning up;MET 3.3	household-chores		sitstand+activity
occupation;office and administrative support;11580 office wok/computer work general;MET 1.5	sitting		sitstand+lowactivity
home activity;miscellaneous;sitting;9060 sitting/lying reading or without observable/identifiable activities;MET 1.3	sitting		sitstand+lowactivity
...	...		..
transportation;walking;17250 walking as the single means to a destination not to work or class;MET 3.0	mixed-activity		walking
transportation;walking;17270 walking as the single means to work or class (not from);MET 3.5	walking		walking
transportation;public transportation;16016 riding in a bus or train;MET 1.3	vehicle		vehicle
household-chores;sitstand+lowactivity;MET 2.8	household-chores		sitstand+lowactivity
vehicle;MET 1.3	vehicle		vehicle

206 rows × 6 columns

In [7]:

```
# using Willettts2018
annot["label:Willettts2018"]
```

Out [7]:

label:Willetts2018

annotation	
7030 sleeping;MET 0.95	sleep
occupation;office and administrative support;11580 office/computer work general;MET 1.5	sit-stand
home activity;household chores;preparing meals/cooking/washing dishes;5035 kitchen activity general cooking/washing/dishes/cleaning up;MET 3.3	mixed
occupation;office and administrative support;11580 office wok/computer work general;MET 1.5	sit-stand
home activity;miscellaneous;sitting;9060 sitting/lying reading or without observable/identifiable activities;MET 1.3	sit-stand
...	...
transportation;walking;17250 walking as the single means to a destination not to work or class;MET 3.0	mixed
transportation;walking;17270 walking as the single means to work or class (not from);MET 3.5	walking
transportation;public transportation;16016 riding in a bus or train;MET 1.3	vehicle
household-chores;sitstand+lowactivity;MET 2.8	mixed
vehicle;MET 1.3	vehicle

206 rows × 1 columns

dtype: object

```
In [8]: # mapping data to low dimension annotation
data["label"] = annot["label:Willetts2018"].reindex(data["annotation"]).t
data
```

Out [8]:

	x	y	z	annotation	label
time					
2016-11-13 02:18:00.000	-0.466690	-0.533341	0.658472	sleeping;MET 0.95	7030 sleep
2016-11-13 02:18:00.010	-0.466690	-0.533341	0.658472	sleeping;MET 0.95	7030 sleep
2016-11-13 02:18:00.020	-0.466690	-0.533341	0.658472	sleeping;MET 0.95	7030 sleep
2016-11-13 02:18:00.030	-0.466690	-0.533341	0.658472	sleeping;MET 0.95	7030 sleep
2016-11-13 02:18:00.040	-0.466690	-0.533341	0.658472	sleeping;MET 0.95	7030 sleep
...	...	...	...	...	...
2016-11-14 06:07:59.960	0.049416	-0.797846	0.565700	sleeping;MET 0.95	7030 sleep
2016-11-14 06:07:59.970	0.049416	-0.782285	0.565700	sleeping;MET 0.95	7030 sleep
2016-11-14 06:07:59.980	0.049416	-0.782285	0.565700	sleeping;MET 0.95	7030 sleep
2016-11-14 06:07:59.990	0.049416	-0.782285	0.565700	sleeping;MET 0.95	7030 sleep
2016-11-14 06:08:00.000	0.049416	-0.782285	0.565700	sleeping;MET 0.95	7030 sleep

10020001 rows x 5 columns

In [9]: `data.label.unique()`Out[9]: `array(['sleep', nan, 'mixed', 'walking', 'vehicle', 'sit-stand'],  
 dtype=object)`

```
In [10]: def window(data, size = "10s"):
X, Y = [], []
for time, d in data.resample(size, origin='start'):
    if d.isna().any().any() or len(d) != 1000:
        continue

    x = d[["x", "y", "z"]].to_numpy()
    y = d["label"].mode(dropna=False).item()

    X.append(x)
    Y.append(y)

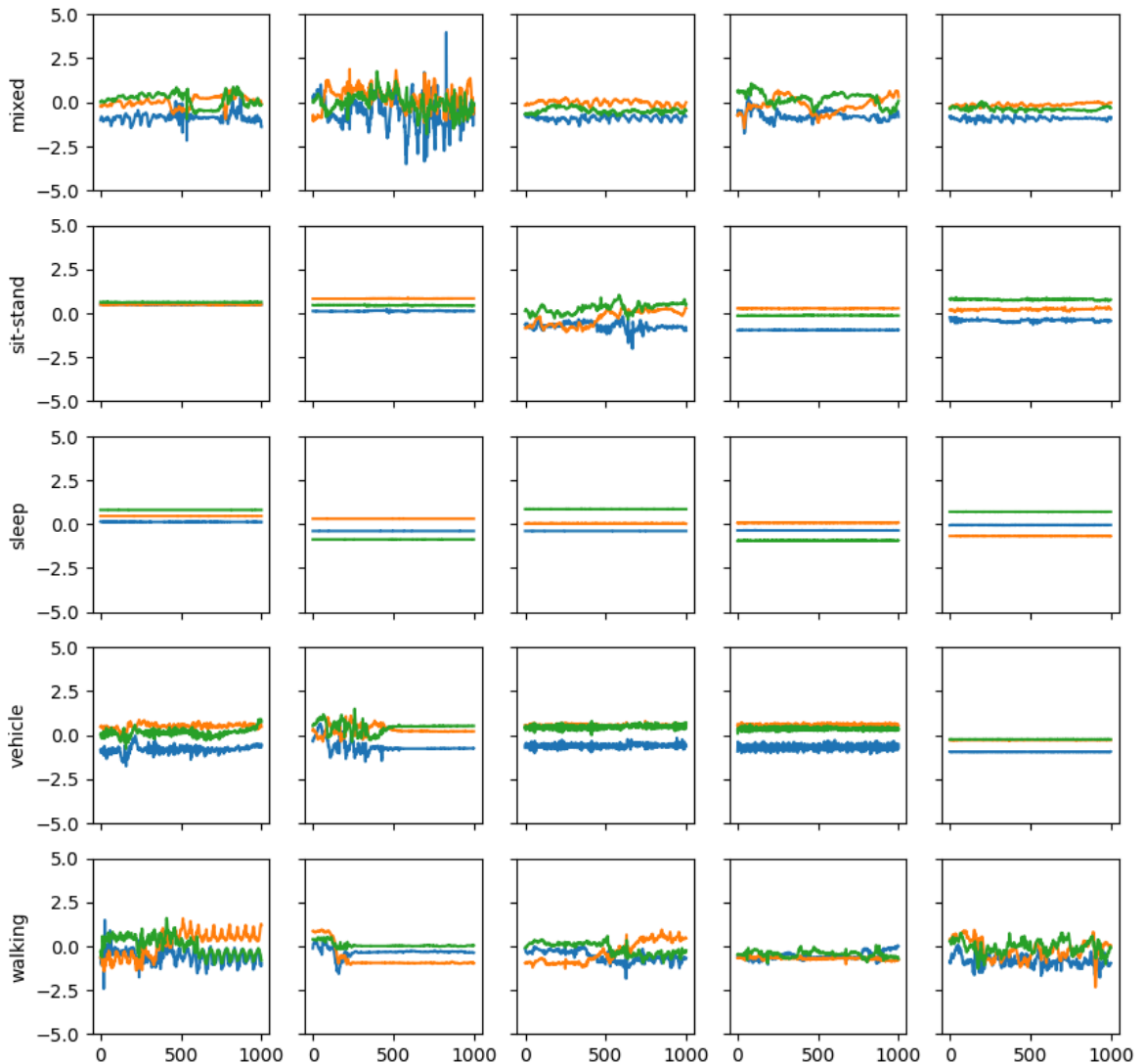
X = np.stack(X)
Y = np.stack(Y)

return X, Y
```

In [11]: `# creating the windows`

```
X, Y = window(data)
```

```
In [12]: # plotting the activities
plots = 5
unique_labels = np.unique(Y)
fig, ax = plt.subplots(len(unique_labels), plots, sharex = True, sharey =
for label, row in zip(unique_labels, ax):
    i = np.random.choice(np.where(Y == label)[0], size = plots)
    row[0].set_ylabel(label)
    for feature, x in zip(X[i], row):
        x.plot(feature)
        x.set_ylim(-5, 5)
fig.show()
```



```
In [13]: # training model to predict activity
Xf = pd.DataFrame([extract_features(x) for x in X])
Xf
```



Out [13]:

	xmin	xq25	xmed	xq75	xmax	ymin	yq
0	-0.482334	-0.466690	-0.466690	-0.466690	-0.466690	-0.548902	-0.5489
1	-0.482334	-0.466690	-0.466690	-0.466690	-0.466690	-0.548902	-0.5489
2	-0.482334	-0.482334	-0.466690	-0.466690	-0.466690	-0.548902	-0.5333
3	-0.482398	-0.482334	-0.466690	-0.466690	-0.466690	-0.548902	-0.5333
4	-0.482398	-0.482334	-0.466690	-0.466690	-0.466626	-0.548902	-0.5333
...	...	...	...	...	...	...	...
7347	0.033708	0.033772	0.049416	0.049416	0.049416	-0.813407	-0.7978
7348	0.033708	0.033772	0.049416	0.049416	0.065059	-0.813407	-0.7978
7349	0.033772	0.049416	0.049416	0.049416	0.049416	-0.797846	-0.7978
7350	0.033708	0.033772	0.049352	0.049416	0.049416	-0.797846	-0.7978
7351	0.033708	0.033772	0.049416	0.049416	0.049416	-0.797846	-0.7978

7352 rows x 40 columns

In [14]:

```
# training model
rfc = RandomForestClassifier(
    n_estimators=100,
    n_jobs=4,
    random_state=42
)
rfc.fit(Xf, Y)
```

Out [14]:

RandomForestClassifier

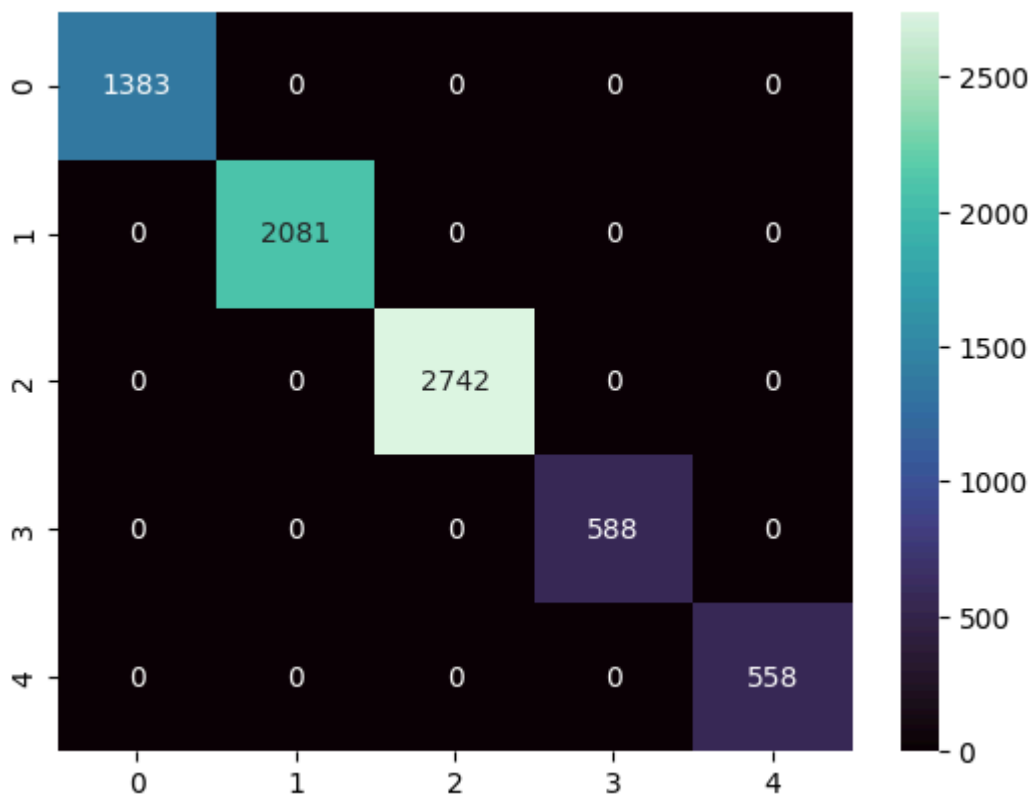
RandomForestClassifier(n\_jobs=4, random\_state=42)

In [15]:

```
print(classification_report(Y, rfc.predict(Xf), zero_division=0))
```

	precision	recall	f1-score	support
mixed	1.00	1.00	1.00	1383
sit-stand	1.00	1.00	1.00	2081
sleep	1.00	1.00	1.00	2742
vehicle	1.00	1.00	1.00	588
walking	1.00	1.00	1.00	558
accuracy			1.00	7352
macro avg	1.00	1.00	1.00	7352
weighted avg	1.00	1.00	1.00	7352

```
In [16]: Yp = rfc.predict(Xf)
sns.heatmap(confusion_matrix(Y, Yp), cmap = "mako", annot = True, fmt = "
plt.show()
```



```
In [17]: # trying to extend model to another person
data = pd.read_csv(
    "data/capture24/P002.csv.gz", compression="gzip",
    index_col="time", parse_dates=["time"],
    dtype={"x": "f4", "y": "f4", "z": "f4", "annotation": "string"}
)
data["label"] = annot["label:Willetts2018"].reindex(data["annotation"]).t
X, Y = window(data)
Xf = pd.DataFrame([extract_features(x) for x in X])
Xf
```

Out [17]:

	xmin	xq25	xmed	xq75	xmax	ymin	yq25
--	------	------	------	------	------	------	------

0	-0.394386	-0.378620	-0.378620	-0.362854	-0.331323	0.444236	0.459960
1	-0.410152	-0.378620	-0.378620	-0.378620	-0.347089	0.444236	0.459960
2	-0.410152	-0.394386	-0.378620	-0.378620	-0.347089	0.444236	0.459960
3	-0.410152	-0.394386	-0.378620	-0.378620	-0.347089	0.444236	0.459960
4	-0.410152	-0.394386	-0.378620	-0.378620	-0.347089	0.444236	0.459960
...	...	...	...	...	...	...	...
5581	-0.360339	-0.344573	-0.344573	-0.344573	-0.328807	-0.732336	-0.716600
5582	-0.344933	-0.344933	-0.344573	-0.344573	-0.328807	-0.717062	-0.701300
5583	-0.344933	-0.344933	-0.344573	-0.344573	-0.328807	-0.717062	-0.701300
5584	-0.344933	-0.344933	-0.344933	-0.344933	-0.329167	-0.717062	-0.701300
5585	-0.344933	-0.344933	-0.344933	-0.344933	-0.329167	-0.717062	-0.701300

5586 rows x 40 columns

```
In [18]: # checking metrics
print(classification_report(Y, rfc.predict(Xf), zero_division=0))
```

	precision	recall	f1-score	support
bicycling	0.00	0.00	0.00	230
mixed	0.40	0.77	0.52	724
sit-stand	0.69	0.52	0.59	2073
sleep	0.93	0.78	0.85	2160
vehicle	0.00	0.00	0.00	0
walking	0.52	0.24	0.33	399
accuracy			0.61	5586
macro avg	0.42	0.39	0.38	5586
weighted avg	0.70	0.61	0.64	5586

```
In [19]: # plotting prediction heatmap
Yp = rfc.predict(Xf)
sns.heatmap(confusion_matrix(Y, Yp), cmap = "mako", annot = True, fmt = "
plt.show()
```

