activity_recognition_demo

January 12, 2021

[27]:
```
from helpers import math_helper
from sensors.activpal import *
from utils import read_functions
from scipy import signal
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.metrics import f1_score, plot_confusion_matrix, confusion_matrix,␣
↪accuracy_score, precision_score, recall_score, confusion_matrix,␣
↪classification_report
from sklearn.ensemble import RandomForestClassifier

import pandas as pd
import numpy as np
import statistics
import os

import matplotlib.pyplot as plt
```

Adnan Akbas # Feature Extraction

[75]:
```
class ActivityRecognitionModel:

    features_columns = ['standard_deviation_x', 'mean_x',␣
↪'standard_deviation_y', 'mean_y','standard_deviation_z', 'mean_z',␣
↪'activiteit']
    activity_columns = ['activity_walking', 'activity_running',␣
↪'activity_standing', 'activity_sitten']
    activities = ['lopen', 'rennen', 'staan', 'zitten']
    segment_size = 12.8

    def __init__(self, segment_size):
        self.segment_size = segment_size

    def get_model(self):
        features_dataset = self.
↪extract_features_from_all_correspondents_lab_dataset()
```

```
        features_dataset[self.activity_columns] = 0

        features_dataset.loc[(features_dataset['activiteit'] == 'lopen'),␣
↪'activity_walking'] = 1
        features_dataset.loc[(features_dataset['activiteit'] == 'rennen'),␣
↪'activity_running'] = 1
        features_dataset.loc[(features_dataset['activiteit'] == 'staan'),␣
↪'activity_standing'] = 1
        features_dataset.loc[(features_dataset['activiteit'] == 'zitten'),␣
↪'activity_sitten'] = 1

        features_dataset.drop('activiteit', axis=1, inplace=True)
        features_dataset.dropna(how='any', inplace=True)

        x = features_dataset[self.features_columns[:-1]]
        y = features_dataset[self.activity_columns]

        ftc = RandomForestClassifier(n_estimators=20, random_state=0)
        ftc.fit(x, y)

        return ftc

    def extract_features_from_all_correspondents_lab_dataset(self):
        all_features_df = pd.DataFrame(index=pd.to_datetime([]))

        for directory in os.walk('../../data'):
            if directory[0] == '../../data':
                for respDirect in directory[1]:
                    if respDirect not in ['output', 'throughput', 'Test data','.
↪ipynb_checkpoints', 'BMR035', 'BMR100', 'BMR051', 'BMR027']:
                      #  if respDirect not in test_users:
                        print("Extracting " + respDirect)
                        features_df = self.
↪extract_features_from_correspondent_lab_dataset(respDirect)
                        all_features_df = pd.concat([all_features_df,␣
↪features_df])

        print("Done extracting features")

        return all_features_df

    def extract_features_from_correspondent_lab_dataset(self, correspondent):
        activpal = Activpal()
        features_df = pd.DataFrame(columns=self.features_columns, index=pd.
↪to_datetime([]))
```

```
        # Getting dataset for a correspodent
        activities_df = read_functions.read_activities(correspondent)


        for activity_name in self.activities:
            activity = activities_df.loc[activity_name]

            if not activity.empty:
                start_time = activity.start
                stop_time = activity.stop
                activpal_df = activpal.read_data(correspondent, start_time,␣
↪stop_time)

                # denormalizing dataset
                activpal_df['x'] = math_helper.
↪convert_value_to_g(activpal_df['pal_accX'])
                activpal_df['y'] = math_helper.
↪convert_value_to_g(activpal_df['pal_accY'])
                activpal_df['z'] = math_helper.
↪convert_value_to_g(activpal_df['pal_accZ'])

                date_range = pd.date_range(start_time, stop_time, freq=str(self.
↪segment_size) + 'S')

                for time in date_range:
                    segment_time = time + pd.DateOffset(seconds=self.
↪segment_size)
                    activpal_segment = activpal_df[(activpal_df.index >= time)␣
↪& (activpal_df.index <= segment_time)]

                    stdev_x =  statistics.stdev(activpal_segment['x']) if␣
↪len(activpal_segment['x']) >= 2 else 0
                    mean_x = activpal_segment['x'].mean()

                    stdev_y =  statistics.stdev(activpal_segment['y']) if␣
↪len(activpal_segment['y']) >= 2 else 0
                    mean_y = activpal_segment['y'].mean()

                    stdev_z =  statistics.stdev(activpal_segment['z']) if␣
↪len(activpal_segment['z']) >= 2 else 0
                    mean_z = activpal_segment['z'].mean()


                    features_df.loc[segment_time] = [stdev_x, mean_x, stdev_y,␣
↪mean_y, stdev_z, mean_z, activity_name]
```

```
        return features_df

    def extract_features_from_correspondent_all_data(self, correspondent):
        activpal = Activpal()
        features_df = pd.DataFrame(columns=self.features_columns[:-1], index=pd.
↪to_datetime([]))
        activpal_df = activpal.read_data(correspondent)

        start_time = activpal_df.index.min()
        stop_time = activpal_df.index.max()

        # descaling dataset
        activpal_df['x'] = math_helper.
↪convert_value_to_g(activpal_df['pal_accX'])
        activpal_df['y'] = math_helper.
↪convert_value_to_g(activpal_df['pal_accY'])
        activpal_df['z'] = math_helper.
↪convert_value_to_g(activpal_df['pal_accZ'])

        date_range = pd.date_range(start_time, stop_time, freq=str(self.
↪segment_size) + 'S')

        print("total_rows: ", len(activpal_df.index))
        count = 0
        rows = 0
        for time in date_range:
            segment_time = time + pd.DateOffset(seconds=self.segment_size)
            activpal_segment = activpal_df[(activpal_df.index >= time) &␣
↪(activpal_df.index <= segment_time)]

            stdev_x =  statistics.stdev(activpal_segment['x']) if␣
↪len(activpal_segment['x']) >= 2 else 0
            mean_x = activpal_segment['x'].mean()

            stdev_y =  statistics.stdev(activpal_segment['y']) if␣
↪len(activpal_segment['y']) >= 2 else 0
            mean_y = activpal_segment['y'].mean()

            stdev_z =  statistics.stdev(activpal_segment['z']) if␣
↪len(activpal_segment['z']) >= 2 else 0
            mean_z = activpal_segment['z'].mean()

            features_df.loc[segment_time] = [stdev_x, mean_x, stdev_y, mean_y,␣
↪stdev_z, mean_z]

            count = count + 1
```

```
            rows = rows + len(activpal_segment)

            if(count % 1000 == 0):
                print("count:", count)
                print("rows_processed", rows )

        return features_df
```

[76]:
```
activity_recognition_model = ActivityRecognitionModel(12.8)
```

[77]:
```
model = activity_recognition_model.get_model()
```

```
Extracting BMR099
Extracting BMR025
Extracting BMR060
Extracting BMR012
Extracting BMR030
Extracting BMR044
Extracting BMR043
Extracting BMR004
Extracting BMR011
Extracting BMR098
Extracting BMR034
Extracting BMR014
Extracting BMR036
Extracting BMR052
Extracting BMR002
Extracting BMR031
Extracting BMR097
Extracting BMR008
Extracting BMR015
Extracting BMR033
Extracting BMR064
Extracting BMR055
Extracting BMR041
Extracting BMR053
Extracting BMR042
Extracting BMR018
Extracting BMR058
Extracting BMR040
Extracting BMR032
Done extracting features
                         activity_walking  activity_running  \
2019-09-12 10:59:03.800                 1                 0
2019-09-12 10:59:16.600                 1                 0
2019-09-12 10:59:29.400                 1                 0
```

```
2019-09-12 10:59:42.200                 1                 0
2019-09-12 10:59:55.000                 1                 0

                         activity_standing  activity_sitten
2019-09-12 10:59:03.800                  0                0
2019-09-12 10:59:16.600                  0                0
2019-09-12 10:59:29.400                  0                0
2019-09-12 10:59:42.200                  0                0
2019-09-12 10:59:55.000                  0                0
```

[56]:
```
corr_002 = activity_recognition_model.
 ↪extract_features_from_correspondent_all_data('BMR002')
```

```
total_rows:  13823915
count: 1000
rows_processed 256107
count: 2000
rows_processed 512218
count: 3000
rows_processed 768329
count: 4000
rows_processed 1024441
count: 5000
rows_processed 1280550
count: 6000
rows_processed 1536659
count: 7000
rows_processed 1792769
count: 8000
rows_processed 2048880
count: 9000
rows_processed 2304985
count: 10000
rows_processed 2561096
count: 11000
rows_processed 2817207
count: 12000
rows_processed 3073317
count: 13000
rows_processed 3329426
count: 14000
rows_processed 3585534
count: 15000
rows_processed 3841648
count: 16000
rows_processed 4097757
count: 17000
rows_processed 4353865
```

```
count: 18000
rows_processed 4609976
count: 19000
rows_processed 4866081
count: 20000
rows_processed 5122192
count: 21000
rows_processed 5378303
count: 22000
rows_processed 5634414
count: 23000
rows_processed 5890523
count: 24000
rows_processed 6146634
count: 25000
rows_processed 6402743
count: 26000
rows_processed 6658849
count: 27000
rows_processed 6914960
count: 28000
rows_processed 7171067
count: 29000
rows_processed 7427178
count: 30000
rows_processed 7683289
count: 31000
rows_processed 7939401
count: 32000
rows_processed 8195510
count: 33000
rows_processed 8451619
count: 34000
rows_processed 8707729
count: 35000
rows_processed 8963840
count: 36000
rows_processed 9219945
count: 37000
rows_processed 9476056
count: 38000
rows_processed 9732167
count: 39000
rows_processed 9988277
count: 40000
rows_processed 10244386
count: 41000
rows_processed 10500494
```

```
count: 42000
rows_processed 10756608
count: 43000
rows_processed 11012717
count: 44000
rows_processed 11268825
count: 45000
rows_processed 11524936
count: 46000
rows_processed 11781041
count: 47000
rows_processed 12037152
count: 48000
rows_processed 12293263
count: 49000
rows_processed 12549374
count: 50000
rows_processed 12805483
count: 51000
rows_processed 13061594
count: 52000
rows_processed 13317703
count: 53000
rows_processed 13573809
count: 54000
rows_processed 13829834
```

[57]:

```
corr_002.head()
```

[57]:

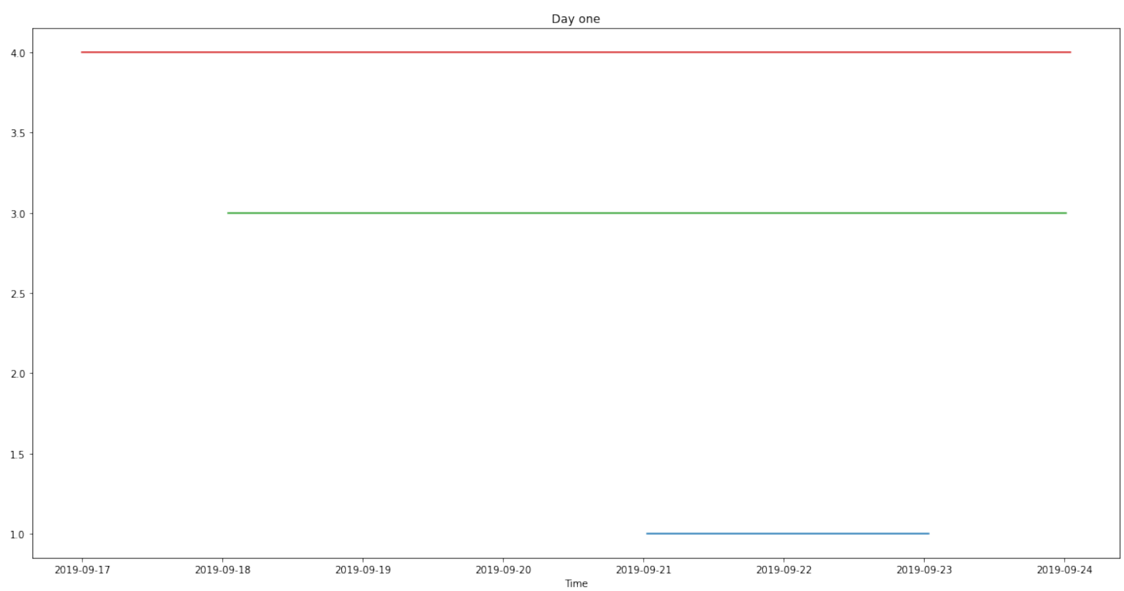
```
                            standard_deviation_x    mean_x  \
2019-09-16 12:45:19.799999              0.420104 -0.245846
2019-09-16 12:45:32.599999              0.160126 -0.037243
2019-09-16 12:45:45.399999              0.307604 -0.444507
2019-09-16 12:45:58.199999              0.265924  0.213328
2019-09-16 12:46:10.999999              0.513550 -0.301774

                            standard_deviation_y    mean_y  \
2019-09-16 12:45:19.799999              0.270373  0.042535
2019-09-16 12:45:32.599999              0.149759  0.122784
2019-09-16 12:45:45.399999              0.275764  0.031373
2019-09-16 12:45:58.199999              0.260005 -0.391267
2019-09-16 12:46:10.999999              0.197028 -0.179085

                            standard_deviation_z    mean_z
2019-09-16 12:45:19.799999              0.234449  1.092448
2019-09-16 12:45:32.599999              0.144499  1.194861
2019-09-16 12:45:45.399999              0.177549  1.045129
```

```
2019-09-16 12:45:58.199999             0.259520  0.955716
2019-09-16 12:46:10.999999             0.836738  0.283971
```

[131]:
```
#activities = ['lopen', 'rennen', 'staan', 'zitten']
import matplotlib.dates as mdates

#hours = mdates.MinuteLocator(interval = 60)
#h_fmt = mdates.DateFormatter('%H')


def label(pred):
    if(pred[0] == 1):
        return 'lopen'
    elif(pred[1] == 1):
        return 'rennen'
    elif(pred[2] == 1):
        return 'staan'
    elif(pred[3] == 1):
        return 'zitten'
    else:
        return ''

for group in corr_002.groupby(corr_002.index.hour):
    day_df = group[1]

    x = day_df[['standard_deviation_x', 'mean_x', 'standard_deviation_y',␣
 ↪'mean_y','standard_deviation_z', 'mean_z']]

    y_predictions = model.predict(x)

#     day_df['activity'] = [label(pred) for pred in y_predictions]


#     lopen = day_df[day_df['activity'] == 'lopen']
#     rennen = day_df[day_df['activity'] == 'rennen']
#     staan = day_df[day_df['activity'] == 'staan']
#     zitten = day_df[day_df['activity'] == 'zitten']


#     fig, ax = plt.subplots(figsize=(20,10))

#     plt.plot(lopen.index, [1] * len(lopen.index), label='lopen')
#     plt.plot(rennen.index, [2] * len(rennen.index), label='rennen')
#     plt.plot(staan.index, [3] * len(staan.index), label='staan')
#     plt.plot(zitten.index, [4] * len(zitten.index), label='zitten')
```

```
#     plt.title("Day one")
#     plt.xlabel('Time')

#     print(day_df.index.min())
#     print(day_df.index.max())

#   #  ax.xaxis.set_major_locator(hours)
#    # ax.xaxis.set_major_formatter(h_fmt)

#     plt.show()

    break
```

```
2019-09-17 00:00:06.199999
2019-09-24 00:59:50.199999
```



[96]:
```
model.classes_
```

[96]:
```
[array([0, 1]), array([0, 1]), array([0, 1]), array([0, 1])]
```

[146]:
```
x = corr_002[['standard_deviation_x', 'mean_x', 'standard_deviation_y',␣
 ↪'mean_y','standard_deviation_z', 'mean_z']]

y_predictions = model.predict(x)
```

```
corr_002['activity'] = [label(pred) for pred in y_predictions]
corr_002['activity_category']=0


corr_002.loc[corr_002['activity'] == 'lopen', 'activity_category'] = 1
corr_002.loc[corr_002['activity'] == 'rennen', 'activity_category'] = 2
corr_002.loc[corr_002['activity'] == 'staan', 'activity_category'] = 3
corr_002.loc[corr_002['activity'] == 'zitten', 'activity_category']  = 4
```

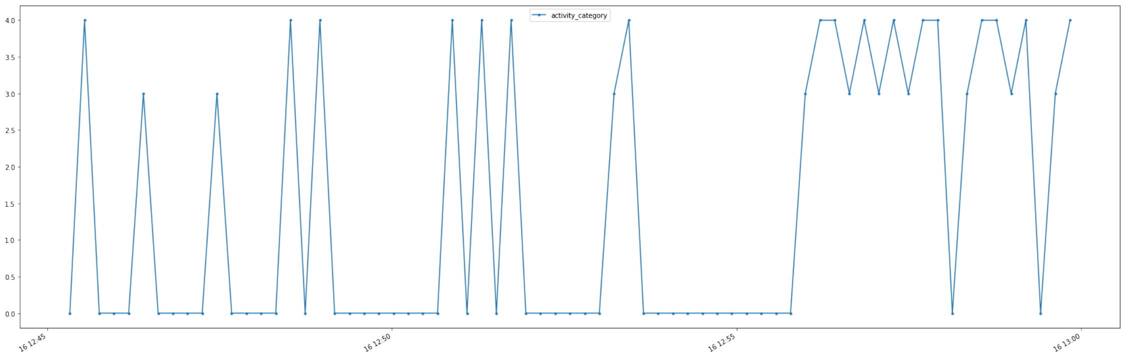
[147]:
```
corr_002.head()
```

[147]:
```
                            standard_deviation_x    mean_x  \
2019-09-16 12:45:19.799999              0.420104 -0.245846
2019-09-16 12:45:32.599999              0.160126 -0.037243
2019-09-16 12:45:45.399999              0.307604 -0.444507
2019-09-16 12:45:58.199999              0.265924  0.213328
2019-09-16 12:46:10.999999              0.513550 -0.301774

                            standard_deviation_y    mean_y  \
2019-09-16 12:45:19.799999              0.270373  0.042535
2019-09-16 12:45:32.599999              0.149759  0.122784
2019-09-16 12:45:45.399999              0.275764  0.031373
2019-09-16 12:45:58.199999              0.260005 -0.391267
2019-09-16 12:46:10.999999              0.197028 -0.179085

                            standard_deviation_z    mean_z activity  \
2019-09-16 12:45:19.799999              0.234449  1.092448
2019-09-16 12:45:32.599999              0.144499  1.194861   zitten
2019-09-16 12:45:45.399999              0.177549  1.045129
2019-09-16 12:45:58.199999              0.259520  0.955716
2019-09-16 12:46:10.999999              0.836738  0.283971

                            activity_category
2019-09-16 12:45:19.799999                  0
2019-09-16 12:45:32.599999                  4
2019-09-16 12:45:45.399999                  0
2019-09-16 12:45:58.199999                  0
2019-09-16 12:46:10.999999                  0
```

[169]:
```
plt.figure(figsize=(30, 10))

mask = (corr_002.index >= '2019-09-16 12:45') & (corr_002.index <= '2019-09-16␣
↪13:00')

corr_002[mask].plot(y='activity_category', marker='.', figsize=(30, 10))
```

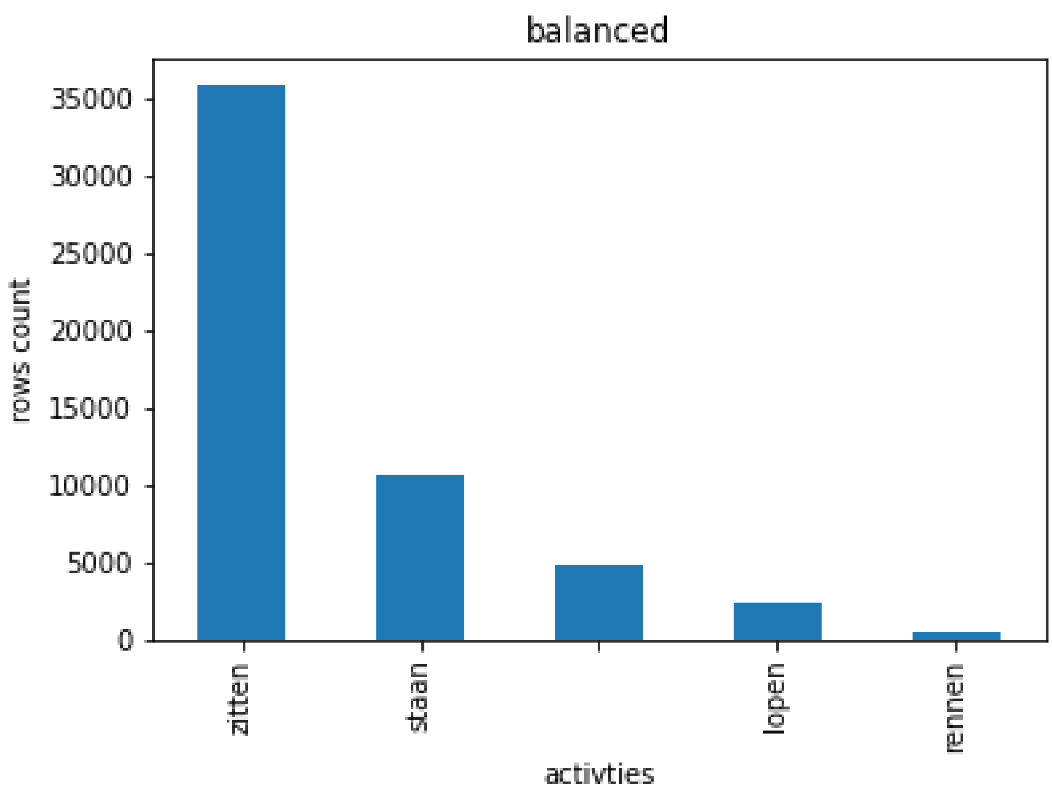
[169]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3227754c50>
```

<Figure size 2160x720 with 0 Axes>



[170]:
```python
corr_002['activity'].value_counts().plot.bar(ylabel='rows count',xlabel='activties',title='balanced')
```

[170]: <matplotlib.axes._subplots.AxesSubplot at 0x7f322c0dcda0>



[ ]: