extract_features_from_all_respodents

January 12, 2021

Adnan Akbas # Extracting activity features from all respodents

[147]:
```
from helpers import math_helper
from sensors.activpal import *
from utils import read_functions
from scipy import signal
import matplotlib.pyplot as plt

import pandas as pd
import statistics
import os
```

[148]:
```
activpal = Activpal()

features_columns = ['standard_deviation_x', 'mean_x', 'standard_deviation_y',␣
 ↪'mean_y','standard_deviation_z',
                    'mean_z','peak_distance_x', 'peak_distance_y',␣
 ↪'peak_distance_z', 'activiteit']
activities = ['lopen', 'rennen', 'springen', 'staan', 'traplopen', 'zitten']
segment_size = 6.4
```

[149]:
```
def calculate_peak_distance(activpal_segment, key):
    accelerations = activpal_segment[key]

    # todo: Think about what kind peaks we are looking for and what we want to␣
 ↪with it
    peak_index, _ = signal.find_peaks(accelerations)

    if len(peak_index) < 2:
        return 0

    peak_values = [accelerations[i] for i in peak_index]

    peak_values.sort(reverse=True)

    # There is a change there are is peak that shows up at multiple index
    # For this reason i am taking the index with highest value.
    highest_peak_index = activpal_segment[activpal_segment[key] ==␣
↪peak_values[0]].index.max()
    second_highest_peak_index = activpal_segment[activpal_segment[key] ==␣
↪peak_values[1]].index.max()

    diff_time = max(highest_peak_index, second_highest_peak_index) -␣
↪min(highest_peak_index, second_highest_peak_index)

    # It's better to use microseconds diveded by 1000 to get milliseconds. This␣
↪way you won't lose information
    # return diff_time.seconds * 1000
    return diff_time.microseconds / 1000
```

[150]:

```
def extract_features_from_correspondent(correspondent):
    features_df = pd.DataFrame(columns=features_columns, index=pd.
↪to_datetime([]))

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

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

                # features
                peak_distance_x = calculate_peak_distance(activpal_segment, 'x')
                peak_distance_y = calculate_peak_distance(activpal_segment, 'y')
                peak_distance_z = calculate_peak_distance(activpal_segment, 'z')

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

                #Matthew
                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, peak_distance_x, peak_distance_y,
                                                 peak_distance_z, activity_name]

    return features_df
```

[151]:

```
def extract_features_from_all_correspondents():
    all_features_df = pd.DataFrame(index=pd.to_datetime([]))

    for directory in os.walk('../../data'):
        if directory[0] == '../../data':
            for respDirect in directory[1]:
                print("Extracting " + respDirect)
                if respDirect not in ['output', 'throughput', 'Test data','.
↪ipynb_checkpoints']:
                    features_df =␣
↪extract_features_from_correspondent(respDirect)
                    all_features_df = pd.concat([all_features_df, features_df])

    print("Done extracting features")

    return all_features_df
```

[152]:

```
df = extract_features_from_all_correspondents()
```

```
Extracting BMR099
Extracting BMR025
Extracting BMR060
Extracting BMR012
Extracting BMR035
Extracting BMR030
Extracting BMR051
Extracting BMR044
Extracting BMR043
Extracting BMR004
Extracting BMR011
Extracting BMR098
Extracting BMR034
Extracting BMR014
Extracting BMR036
Extracting BMR052
Extracting throughput
Extracting BMR002
Extracting BMR031
Extracting BMR097
Extracting BMR008
Extracting BMR015
Extracting BMR033
Extracting output
Extracting BMR100
Extracting BMR064
Extracting BMR055
Extracting .ipynb_checkpoints
Extracting BMR027
Extracting BMR041
Extracting BMR053
Extracting BMR042
Extracting BMR018
Extracting BMR058
Extracting BMR040
Extracting BMR032
Done extracting features
```

[153]:

```
df.head()
```

[153]:

```
                         standard_deviation_x   mean_x standard_deviation_y  \
2019-09-12 10:58:57.400              0.316779 -1.03691              0.24107
2019-09-12 10:59:03.800              0.471659 -1.05928             0.373105
2019-09-12 10:59:10.200              0.472722 -1.03658             0.332441
2019-09-12 10:59:16.600              0.486818 -1.02629              0.32838
2019-09-12 10:59:23.000              0.516327 -1.04024             0.342754

                             mean_y standard_deviation_z    mean_z  \
2019-09-12 10:58:57.400   0.0243632             0.415354  0.180017
2019-09-12 10:59:03.800   0.0329861             0.562155  0.189236
2019-09-12 10:59:10.200   0.0405506             0.586077  0.157862
```

```
2019-09-12 10:59:16.600  0.0381944             0.545233  0.147817
2019-09-12 10:59:23.000  0.0301464             0.539769  0.154424

                        peak_distance_x peak_distance_y peak_distance_z  \
2019-09-12 10:58:57.400         150.002         150.001         300.003
2019-09-12 10:59:03.800         399.993         100.001               0
2019-09-12 10:59:10.200         250.004         199.998               0
2019-09-12 10:59:16.600         999.996         249.996               0
2019-09-12 10:59:23.000         349.997          50.002               0

                        activiteit
2019-09-12 10:58:57.400      lopen
2019-09-12 10:59:03.800      lopen
2019-09-12 10:59:10.200      lopen
2019-09-12 10:59:16.600      lopen
2019-09-12 10:59:23.000      lopen
```

replacing time range index with number because it's not useful anymore

[154]:

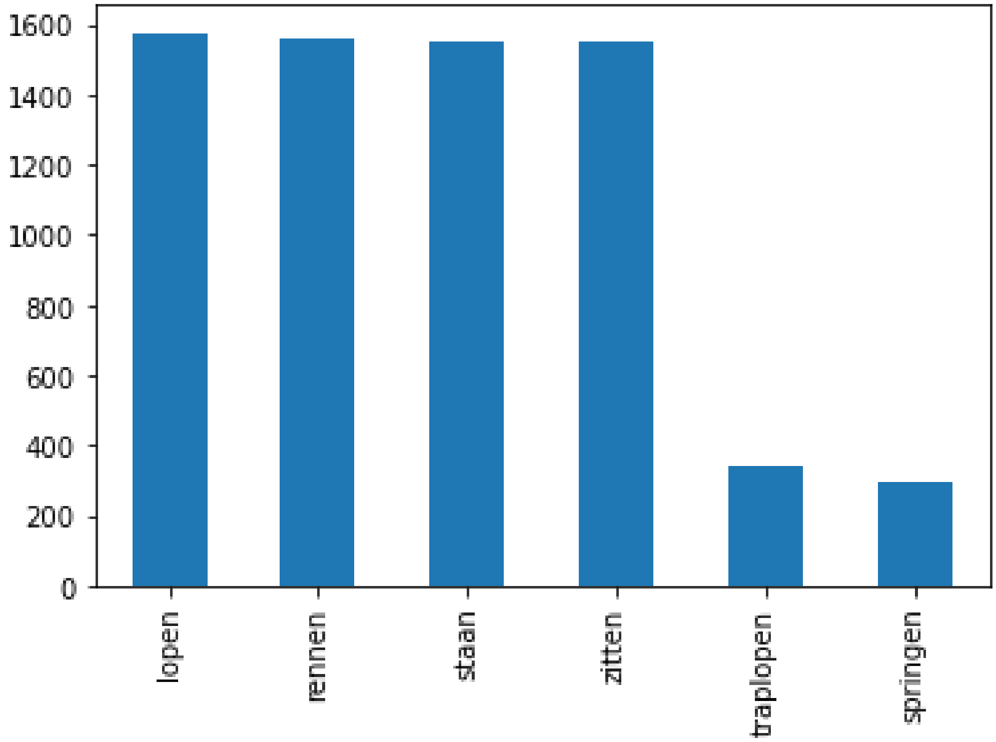
```
df.index = range(len(df.index))
```

[155]:

```
df['activiteit'].value_counts().plot.bar()
```

[155]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f239498d630>
```



[163]:

```
def balance_dataset_by_activity(dataset):
    highest_frequency  = dataset.groupby('activiteit').
 ↪count()['standard_deviation_x'].max()
    unbalanced_dataset = dataset.copy()

    for activity_name in unbalanced_dataset.activiteit.unique():
        activity_data = unbalanced_dataset[unbalanced_dataset['activiteit'] ==␣
 ↪activity_name]

        multiplier =  int(highest_frequency / len(activity_data)) - 1
        unbalanced_dataset = unbalanced_dataset.append([activity_data] *␣
 ↪multiplier, ignore_index=True)

        activity_amount = len(activity_data[ activity_data['activiteit'] ==␣
 ↪activity_name])
        missing_amount = highest_frequency - activity_amount
        unbalanced_dataset = unbalanced_dataset.append(activity_data[:
 ↪missing_amount], ignore_index=True)

    return unbalanced_dataset
```

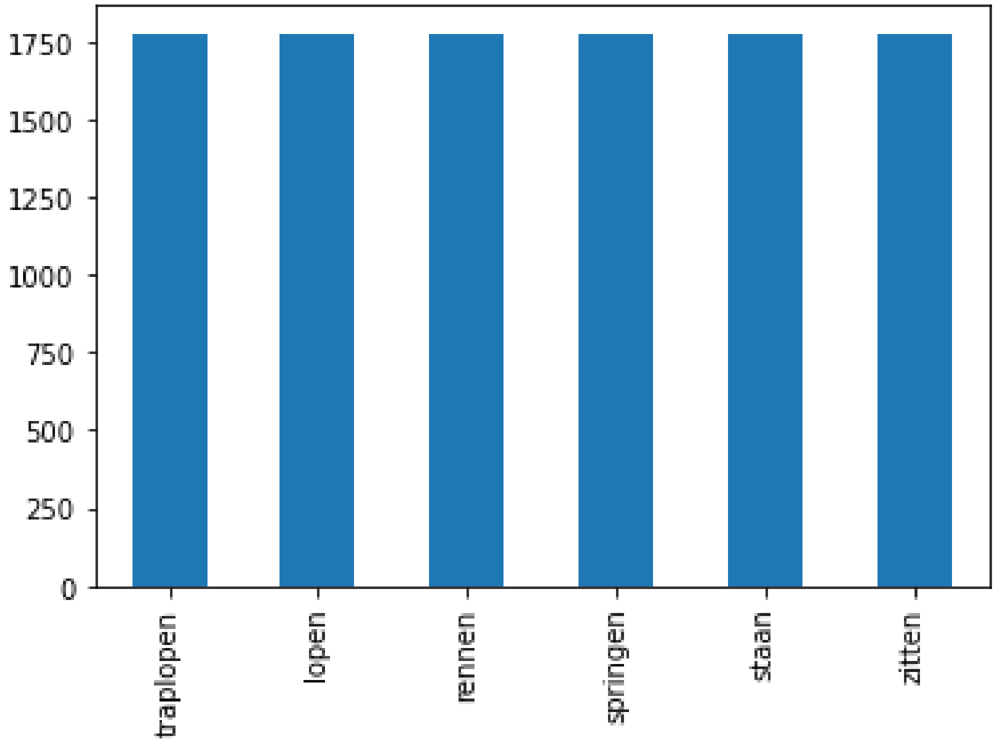
[165]:

```
df = balance_dataset_by_activity(df)
df['activiteit'].value_counts().plot.bar()
```

```
1776
0
0
0
0
0
0
```

[165]:

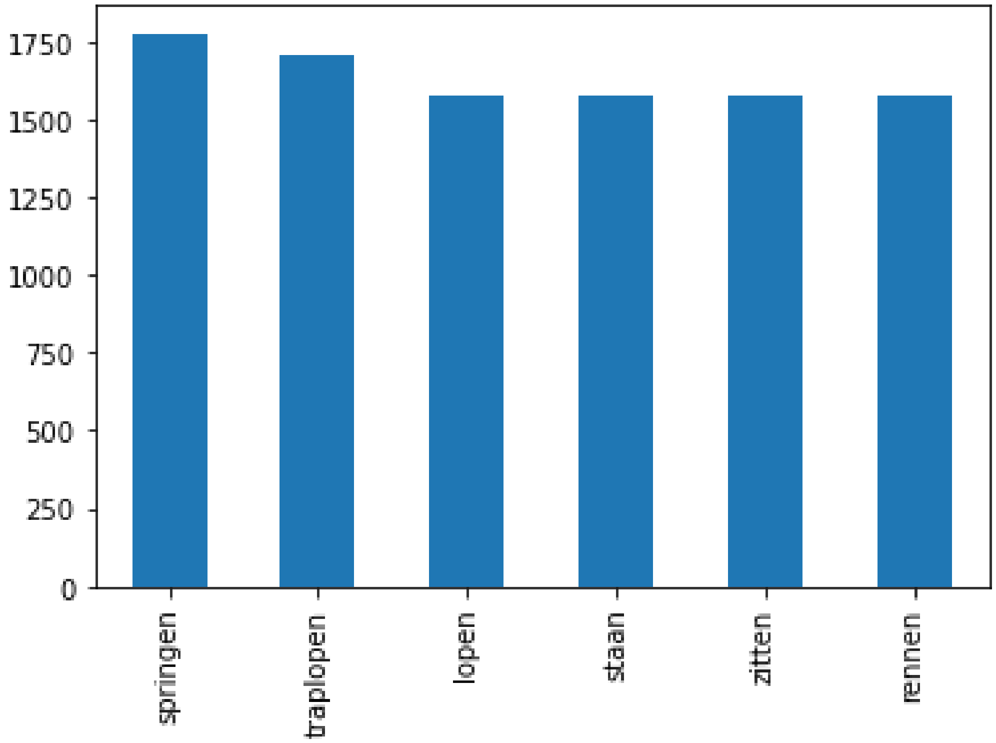
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f239437fe48>
```



[158]:
```
# Balacing dataset for jumping
#jumping = df[ df['activiteit'] == 'springen']
#multiplier =  int(len(walking) / len(jumping)) - 1

#df = df.append([jumping] * multiplier, ignore_index=True)

# filling left over missing values
#jumping_missing_values_amount = len(df[ df['activiteit'] == 'springen'])

#missing_amount = len(walking) - jumping_missing_values_amount

#df = df.append(jumping[:missing_amount], ignore_index=True)
```

[159]:
```
df['activiteit'].value_counts().plot.bar()
```

[159]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2394ac7710>
```



Saving extracted features to activity.csv in activpal/data

[160]:
```
df.to_csv('../../data/activity_features_2.csv')
print('saved')
```

```
saved
```