feature_selection

January 12, 2021

[215]:
```
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 accuracy_score, precision_score, recall_score,␣
↪confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier

import pandas as pd
import statistics
import os
```

1 Feature Extraction

[216]:
```
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
```

[217]:
```
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
```

[218]:
```
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()

                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
```

[219]:
```
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]:
                if respDirect not in ['output', 'throughput', 'Test data','.
↪ipynb_checkpoints']:
                    print("Extracting " + respDirect)
                    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
```

[220]:
```
features_dataset = extract_features_from_all_correspondents()
```

```
Extracting BMR025
Extracting BMR060
Extracting BMR012
Extracting BMR035
Extracting BMR030
Extracting BMR051
Extracting BMR044
Extracting BMR043
Extracting BMR004
Extracting BMR011
Extracting BMR034
Extracting BMR014
Extracting BMR036
Extracting BMR052
Extracting BMR002
Extracting BMR031
Extracting BMR008
Extracting BMR015
Extracting BMR033
Extracting BMR055
Extracting BMR027
Extracting BMR041
Extracting BMR053
Extracting BMR042
Extracting BMR018
Extracting BMR058
Extracting BMR040
Extracting BMR032
Done extracting features
```

2 model preperation

[221]:
```
activities = ['lopen', 'rennen', 'springen', 'staan', 'traplopen', 'zitten']

features_dataset[['activity_walking', 'activity_running', 'activity_jumping',␣
↪'activity_standing', 'activity_traplopen',
                  'activity_sitten']] = 0
```

[222]:
```
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'] == 'springen'),␣
↪'activity_jumping'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'staan'),␣
↪'activity_standing'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'traplopen'),␣
↪'activity_traplopen'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'zitten'),␣
↪'activity_sitten'] = 1
features_dataset.drop('activiteit', axis=1, inplace=True)
```

[223]:
```
fill_na_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']

for column in fill_na_columns:
    features_dataset[column].fillna(0, inplace=True)
```

[224]:
```
features_dataset.head()
```

[224]:

	standard_deviation_x	mean_x	standard_deviation_y	mean_y	standard_deviation_z	mean_z	peak_distance_x	peak_distance_y	peak_distance_z	activity_walking	activity_running	activity_jumping
2019-10-07 15:09:18.400	0.455428	-1.105903	0.350250	0.141493	0.594701	-0.010045	150.009	950.003	0.000	1	0	0
2019-10-07 15:09:24.800	0.517525	-1.078135	0.307128	0.120217	0.596488	-0.088963	349.995	949.995	549.995	1	0	0
2019-10-07 15:09:31.200	0.499803	-1.082093	0.325409	0.135541	0.570476	-0.059028	200.001	150.003	250.001	1	0	0
2019-10-07 15:09:37.600	0.549347	-1.112361	0.356190	0.140607	0.530967	-0.019748	200.000	399.997	650.005	1	0	0
2019-10-07 15:09:44.000	0.563355	-1.087674	0.370789	0.160838	0.632978	-0.038070	549.994	50.001	0.000			

```
2019-10-07 15:09:44.000                1                  0                   0

                         activity_standing  activity_traplopen  \
2019-10-07 15:09:18.400                  0                   0
2019-10-07 15:09:24.800                  0                   0
2019-10-07 15:09:31.200                  0                   0
2019-10-07 15:09:37.600                  0                   0
2019-10-07 15:09:44.000                  0                   0

                         activity_sitten
2019-10-07 15:09:18.400                0
2019-10-07 15:09:24.800                0
2019-10-07 15:09:31.200                0
2019-10-07 15:09:37.600                0
2019-10-07 15:09:44.000                0
```

2.1 Preparing feature dataset for learning

2.1.1 Splitting in x and y

[225]:
```
x = features_dataset[['standard_deviation_x', 'mean_x', 'standard_deviation_y',␣
↪'mean_y','standard_deviation_z',
                      'mean_z','peak_distance_x', 'peak_distance_y',␣
↪'peak_distance_z']]
y = features_dataset[['activity_walking', 'activity_running',␣
↪'activity_jumping', 'activity_standing', 'activity_traplopen',␣
↪'activity_sitten']]
train_x, valid_x, train_y, valid_y = train_test_split(x,y, test_size=0.2,␣
↪random_state=23, stratify=y)
```

[226]:
```
train_x.head()
```

[226]:
```
                         standard_deviation_x    mean_x  standard_deviation_y  \
2019-10-10 13:46:52.800              0.750314 -1.058078              0.464737
2019-10-08 10:16:42.000              0.000000 -0.015873              0.000000
2019-10-08 10:17:33.200              0.000000 -0.015873              0.000000
2019-10-03 11:58:14.800              0.000000 -0.952381              0.000000
2019-10-10 12:56:46.400              0.006396 -0.220472              0.009684

                           mean_y  standard_deviation_z    mean_z  \
2019-10-10 13:46:52.800  0.036422              0.711707  0.109142
2019-10-08 10:16:42.000  0.238095              0.000000  1.190476
2019-10-08 10:17:33.200  0.238095              0.000000  1.190476
2019-10-03 11:58:14.800  0.269841              0.000000 -0.238095
2019-10-10 12:56:46.400  0.147107              0.006520  1.171229
```

	peak_distance_x	peak_distance_y	peak_distance_z
2019-10-10 13:46:52.800	699.996	750.002	0.0
2019-10-08 10:16:42.000	0.000	0.000	0.0
2019-10-08 10:17:33.200	0.000	0.000	0.0
2019-10-03 11:58:14.800	0.000	0.000	0.0
2019-10-10 12:56:46.400	849.994	949.993	0.0

3 Decision Tree model

[250]:
```
dtc = tree.DecisionTreeClassifier()
dtc = dtc.fit(train_x, train_y)
```

3.1 results

Random seed: 23

Features |Features|| |——| —-| | 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|

Time range	Accuracy	Precision	Recall
0.4S	80%	86%	80%
0.8S	82%	90%	82%
1.0S	83%	90%	83%
1.6S	83%	90%	83%
2.0S	83%	90%	83%
3.2S	83%	89.5%	83%
4.0S	83%	90.5%	82%
6.4S	84%	90%	85%
8.0S	85%	91%	85%
12.8S	87%	92%	87%

12.8S gives the best result

[251]:
```
prediction_y = dtc.predict(valid_x)
```

Accuracy

[252]:
```
accuracy_score(valid_y, prediction_y)
```

[252]: 0.8344709897610921

Precision

[253]:
```
precision_score(valid_y, prediction_y, average='micro')
```

[253]:
```
0.8923357664233577
```

Recall

[254]:
```
recall_score(valid_y, prediction_y , average='micro')
```

[254]:
```
0.8344709897610921
```

4 Random tree forest

[255]:
```
#from sklearn.preprocessing import StandardScaler

#sc = StandardScaler()
#train_x = sc.fit_transform(train_x)
#valid_x = sc.transform(valid_x)
```

[256]:
```
regressor = RandomForestClassifier(n_estimators=20, random_state=0)
regressor.fit(train_x, train_y)
```

[256]:
```
RandomForestClassifier(n_estimators=20, random_state=0)
```

4.1 Result

[257]:
```
prediction_y = dtc.predict(valid_x)
```

4.1.1 classification_report

Accuracy

[258]:
```
accuracy_score(valid_y, prediction_y, normalize=True)
```

[258]:
```
0.8344709897610921
```

[260]:
```
print(classification_report(valid_y,prediction_y,␣
↪target_names=['activity_walking', 'activity_running', 'activity_jumping',␣
↪'activity_standing', 'activity_traplopen',
                  'activity_sitten'], zero_division=0))
```

```
                   precision    recall  f1-score   support

  activity_walking       0.87      0.83      0.85       270
```

activity_running	0.91	0.89	0.90	265
activity_jumping	0.57	0.59	0.58	51
activity_standing	0.92	0.84	0.88	263
activity_traplopen	0.61	0.42	0.50	60
activity_sitten	0.98	0.92	0.95	263
micro avg	0.89	0.83	0.86	1172
macro avg	0.81	0.75	0.78	1172
weighted avg	0.89	0.83	0.86	1172
samples avg	0.83	0.83	0.83	1172