all_steps_activity recognition_v3_analysis_normalized_data

January 12, 2021

[1]:
```
from helpers import math_helper, pandas_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

[2]:
```
activpal = Activpal()

features_columns = ['standard_deviation_x', 'mean_x', 'standard_deviation_y',␣
↪'mean_y','standard_deviation_z', 'mean_z', 'activiteit']
activity_columns = ['cycling_light','cycling_hard', 'activity_walking',␣
↪'activity_running', 'activity_jumping', 'activity_standing',␣
↪'activity_traplopen', 'activity_sitten']

activities = ['fietsen licht', 'fietsen zwaar', 'lopen', 'rennen', 'springen',␣
↪'staan', 'traplopen', 'zitten']
test_users = ['BMR002', 'BMR004', 'BMR008']
segment_size = 12.8
```

[3]:
```
def extract_features_from_correspondent(correspondent):
    features_df = pd.DataFrame(columns=features_columns, index=pd.
↪to_datetime([]))
```

```
    # Getting dataset for a correspodent
    activities_df = pandas_helper.read_csv_activiteiten(correspondent)
    activpal_corr_df = pandas_helper.read_activpal_20_diceface(correspondent)

    #2019-09-12 09:51:55:000001
    #activpal_corr_df.index = pd.to_datetime(activpal_corr_df.index,␣
↪format='%Y-%m-%d %H:%M:%S:%f')

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

        if not activity.empty:
            start_time = activity.start
            stop_time = activity.stop
            activity_mask = (activpal_corr_df.index >= start_time) &␣
↪(activpal_corr_df.index <= stop_time)
            activpal_df = activpal_corr_df.loc[activity_mask].copy()

            # 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)]

                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
```

[4]:
```
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', 'BMR035', 'BMR100', 'BMR051', 'BMR027']:
                  #  if respDirect not in test_users:
                    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
```

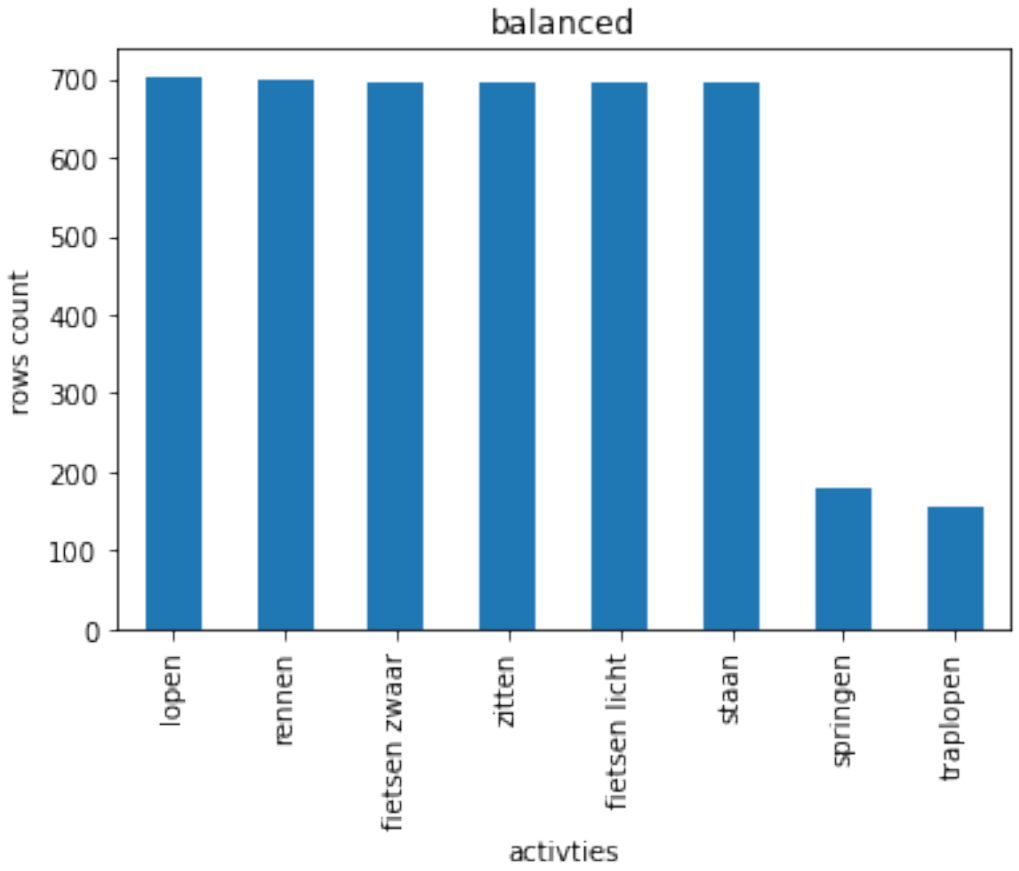
[5]:
```
features_dataset = extract_features_from_all_correspondents()
```

```
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
```

[6]:
```
features_dataset['activiteit'].value_counts().plot.bar(ylabel='rows␣
↪count',xlabel='activties',title='balanced')
```

[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f01f87cac18>



1 model preperation

[7]:
```
features_dataset[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'] == '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.loc[(features_dataset['activiteit'] == 'fietsen licht'),␣
↪'cycling_light'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'fietsen zwaar'),␣
↪'cycling_hard'] = 1



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

[7]:

	standard_deviation_x	mean_x	standard_deviation_y	mean_y	standard_deviation_z	mean_z	cycling_light	cycling_hard	activity_walking
2019-09-12 10:25:08.800	0.428530	-0.678992	0.156412	-0.848864	0.133432	0.133956	1	0	0
2019-09-12 10:25:21.600	0.434741	-0.669878	0.159230	-0.857884	0.121550	0.130443	1	0	0
2019-09-12 10:25:34.400	0.445535	-0.673389	0.167727	-0.856707	0.127858	0.131030	1	0	0
2019-09-12 10:25:47.200	0.443970	-0.674325	0.157079	-0.852017	0.141328	0.125440	1	0	0
2019-09-12 10:26:00.000	0.429386	-0.681855	0.152461	-0.860815	0.134197	0.121942			

```
2019-09-12 10:26:00.000              1             0                  0

                         activity_running  activity_jumping  \
2019-09-12 10:25:08.800                 0                 0
2019-09-12 10:25:21.600                 0                 0
2019-09-12 10:25:34.400                 0                 0
2019-09-12 10:25:47.200                 0                 0
2019-09-12 10:26:00.000                 0                 0

                         activity_standing  activity_traplopen  \
2019-09-12 10:25:08.800                  0                   0
2019-09-12 10:25:21.600                  0                   0
2019-09-12 10:25:34.400                  0                   0
2019-09-12 10:25:47.200                  0                   0
2019-09-12 10:26:00.000                  0                   0

                         activity_sitten
2019-09-12 10:25:08.800                0
2019-09-12 10:25:21.600                0
2019-09-12 10:25:34.400                0
2019-09-12 10:25:47.200                0
2019-09-12 10:26:00.000                0
```

1.1 Preparing feature dataset for learning

1.1.1 Splitting in x and y

[8]:
```
x = features_dataset[features_columns[:-1]]
y = features_dataset[activity_columns]
train_x, valid_x, train_y, valid_y = train_test_split(x,y, test_size=0.2,␣
 ↪random_state=23, stratify=y)

train_x.head()
```

[8]:
```
                         standard_deviation_x    mean_x  standard_deviation_y  \
2019-10-08 15:06:12.000              0.497198 -0.538318              0.192663
2019-09-30 14:41:44.000              0.426062 -1.040198              0.318967
2019-10-11 13:58:43.200              1.184394 -0.889310              0.941234
2019-09-25 12:37:36.800              0.420638 -0.796739              0.170242
2019-09-30 10:41:55.200              0.085699 -1.029754              0.122041

                           mean_y  standard_deviation_z    mean_z
2019-10-08 15:06:12.000 -0.833085              0.190893 -0.205915
2019-09-30 14:41:44.000  0.130489              0.265027  0.432076
2019-10-11 13:58:43.200 -0.185981              1.256313  0.115606
2019-09-25 12:37:36.800 -0.812365              0.151457  0.047125
```

```
2019-09-30 10:41:55.200 -0.056458              0.242803   0.324308
```

2 Random tree forest

[9]:
```
ftc = RandomForestClassifier(n_estimators=20, random_state=0)
ftc.fit(train_x, train_y)
```

[9]:
```
RandomForestClassifier(n_estimators=20, random_state=0)
```

2.1 Testing and results

[10]:
```
prediction_y = ftc.predict(valid_x)
```

2.1.1 Result

Accuracy

[11]:
```
accuracy_score(valid_y, prediction_y, normalize=True)
```

[11]:
```
0.9092920353982301
```

Classification report

[12]:
```
print(classification_report(valid_y,prediction_y,␣
↪target_names=activity_columns, zero_division=0))
```

```
                   precision    recall  f1-score   support

     cycling_light       0.87      0.88      0.88       139
      cycling_hard       0.90      0.86      0.88       139
  activity_walking       0.91      0.94      0.92       141
  activity_running       0.97      0.96      0.96       140
  activity_jumping       0.95      0.56      0.70        36
 activity_standing       0.94      0.98      0.96       139
activity_traplopen       0.92      0.71      0.80        31
   activity_sitten       0.98      0.97      0.97       139

         micro avg       0.93      0.91      0.92       904
         macro avg       0.93      0.86      0.88       904
      weighted avg       0.93      0.91      0.92       904
       samples avg       0.91      0.91      0.91       904
```

2.1.2 Confusion matrix

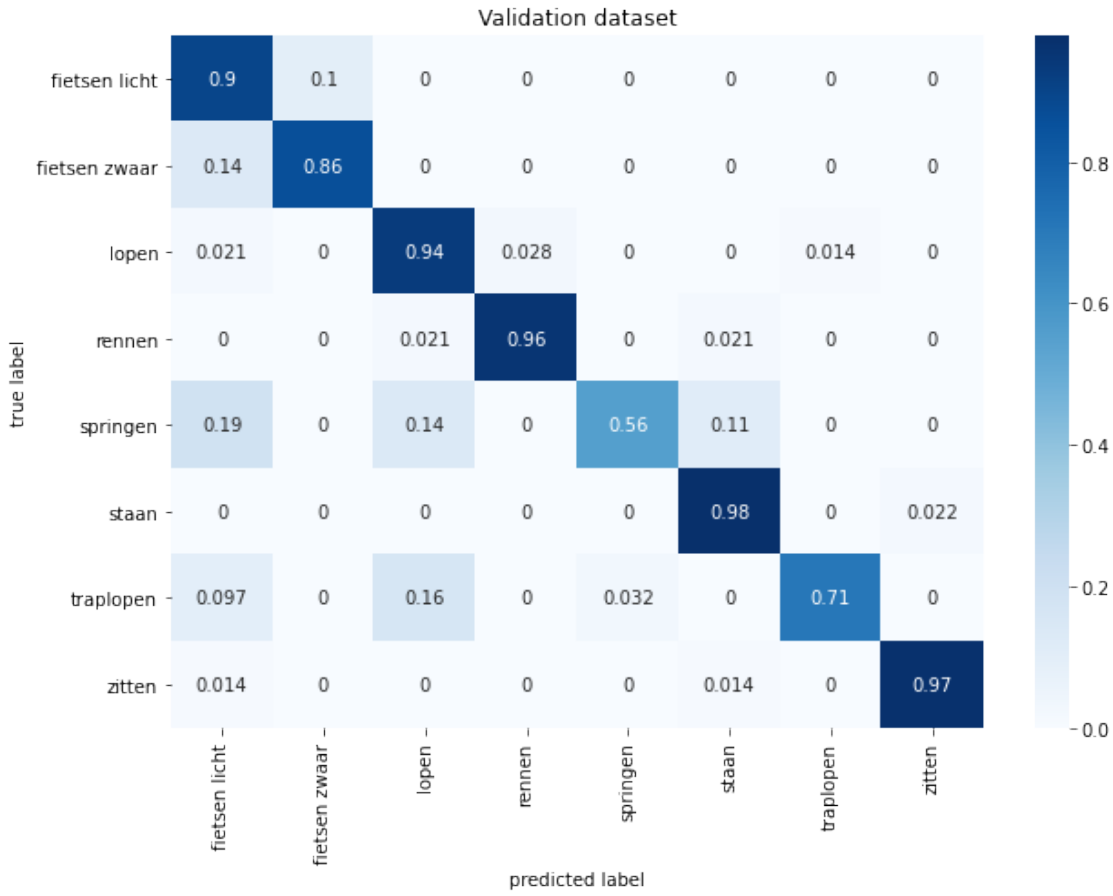
[13]:
```
import seaborn as sn

#confusion_matrix(valid_y, prediction_y)
cm = confusion_matrix(valid_y.values.argmax(axis=1), prediction_y.
 ↪argmax(axis=1), normalize='true')

df_cm = pd.DataFrame(cm, index=activities, columns=activities)
df_cm.head()
plt.figure(figsize = (10,7))
sn.heatmap(df_cm, annot=True, cmap='Blues')

plt.title("Validation dataset")
plt.xlabel("predicted label")
plt.ylabel("true label")
```

[13]:
```
Text(69.0, 0.5, 'true label')
```



3 Result Summary

Random seed: 23 n_estimators: 20

3.0.1 With balanced dataset

Features	
standard_deviation_x	mean_x
standard_deviation_y	mean_y
standard_deviation_z	mean_z

Time range	Accuracy	Precision	Recall	F1
0.4S	93%	96%	93%	95%
0.8S	95%	97%	95%	96%
1.0S	95%	98%	95%	96%
1.6S	95%	97%	95%	96%
2.0S	95%	97%	95%	96%
3.2S	96%	98%	96%	97%
4.0S	95%	98%	95%	96%
6.4S	97%	98%	97%	97%
8.0S	96%	98%	96%	97%
10.0S	96%	99%	96%	97%
12.8S	97%	98%	97%	97%

Best results:

Time range	Accuracy	Precision	Recall	F1
6.4S	97%	98%	97%	97%
12.8S	97%	98%	97%	97%

3.1 Diagnostics

3.1.1 Cross validation analysis

[14]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold, KFold
import seaborn as sn
from sklearn.model_selection import cross_val_predict

rfc = RandomForestClassifier(n_estimators=20, random_state=0)
pred_y = cross_val_predict(rfc, x, y)
```

[15]:
```
accuracy_scores = cross_val_score(rfc, x, y, scoring='accuracy')
recall_scores = cross_val_score(rfc, x, y , scoring='recall_micro')
precision_scores = cross_val_score(rfc, x, y , scoring='precision_micro')

print("Accuracy: %0.2f (+/- %0.2f)" % (accuracy_scores.mean(), accuracy_scores.
 ↪std() * 2))
print("Recall: %0.2f (+/- %0.2f)" % (recall_scores.mean(), recall_scores.std()␣
 ↪* 2))
print("Precision: %0.2f (+/- %0.2f)" % (precision_scores.mean(),␣
 ↪precision_scores.std() * 2))
```

```
Accuracy: 0.68 (+/- 0.09)
Recall: 0.68 (+/- 0.09)
Precision: 0.77 (+/- 0.05)
```

[16]:
```
skf = KFold(n_splits=5, shuffle=True)

accuracy_scores = np.array([])
recall_scores = np.array([])
precision_scores = np.array([])
```

[17]:
```
for train_index, test_index in skf.split(x, y):
    x_train, y_train = x.iloc[train_index], y.iloc[train_index]
    x_test, y_test = x.iloc[test_index], y.iloc[test_index]

    rfc = RandomForestClassifier(n_estimators=20, random_state=0)
    rfc.fit(x_train, y_train)

    y_prediction = rfc.predict(x_test)

    accuracy_scores  = np.append(accuracy_scores, accuracy_score(y_test,␣
 ↪y_prediction, normalize=True))
    recall_scores    = np.append(recall_scores, recall_score(y_test,␣
 ↪y_prediction, average='micro'))
    precision_scores = np.append(precision_scores, precision_score(y_test,␣
 ↪y_prediction, average='micro'))
```

[18]:
```
accuracy_scores.mean()
```

[18]: 0.8829646017699115

[19]:
```
recall_scores.mean()
```

[19]: 0.8829646017699115

[20]:
```
precision_scores.mean()
```

[20]: 0.920851054674236

[21]:
```
#grid search cv sklearn
```

[ ]:
```

```