all_steps_activity recognition_final_version

January 12, 2021

[1]:
```
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 pickle
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 = ['activity_cycling', 'activity_walking',␣
↪'activity_running', 'activity_jumping', 'activity_standing',␣
↪'activity_traplopen', 'activity_sitten']
#activities = ['fietsen licht', 'fietsen zwaar', 'lopen', 'rennen', 'springen',␣
↪'staan', 'traplopen', 'zitten']

activity_columns = ['activity_cycling', 'activity_walking', 'activity_running',␣
↪'activity_standing', 'activity_sitten']
activities = ['fietsen licht', 'fietsen zwaar', 'lopen', 'rennen', 'staan',␣
↪'zitten']

test_users = ['BMR004', 'BMR034', 'BMR097']
segment_size = 9.4
```

[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 = 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)]

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

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

                #Adnan
                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_correspondents(correspodents):
    all_features_df = pd.DataFrame(index=pd.to_datetime([]))


    print(len(correspodents))
    for correspodent in correspodents:
        print("Extracting " + correspodent)

        features_df     = extract_features_from_correspondent(correspodent)
        all_features_df = pd.concat([all_features_df, features_df])

    print("Done extracting features")

    return all_features_df

def extract_features_from_all_correspondents(exclude_test_correspodent = True):

    exclude_directory = ['output', 'throughput', 'Test data','.
↪ipynb_checkpoints']
    exclude_respodents = ['BMR015','BMR025','BMR027', 'BMR035', 'BMR051',␣
↪'BMR054', 'BMR060', 'BMR099', 'BMR100']

    exclude = exclude_respodents + exclude_directory

    if (exclude_test_correspodent):
        exclude = exclude + test_users

    correspodents = []

    for directory in os.walk('../../data'):
        if directory[0] == '../../data':
            correspodents = directory[1]

    for exclude_item in exclude:
        if exclude_item in correspodents:
            correspodents.remove(exclude_item)

    return extract_features_from_correspondents(correspodents)
```

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

```
22
Extracting BMR012
Extracting BMR030
Extracting BMR044
Extracting BMR043
Extracting BMR011
Extracting BMR098
Extracting BMR014
Extracting BMR036
Extracting BMR052
Extracting BMR002
Extracting BMR031
Extracting BMR008
Extracting BMR033
Extracting BMR064
Extracting BMR055
Extracting BMR041
Extracting BMR053
Extracting BMR042
Extracting BMR018
Extracting BMR058
Extracting BMR040
Extracting BMR032
Done extracting features
```

1 model preperation

[6]:
```
features_dataset[activity_columns] = 0

#features_dataset.loc[(features_dataset['activiteit'] == 'springen'),␣
↪'activity_jumping'] = 1
#features_dataset.loc[(features_dataset['activiteit'] == 'traplopen'),␣
↪'activity_traplopen'] = 1
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.loc[(features_dataset['activiteit'] == 'fietsen licht'),␣
↪'activity_cycling'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'fietsen zwaar'),␣
↪'activity_cycling'] = 1
```

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

features_dataset.head()
```

[6]:

	standard_deviation_x	mean_x	standard_deviation_y	mean_y	standard_deviation_z	mean_z	activity_cycling	activity_walking	activity_running	activity_standing	activity_sitten
2019-10-14 09:44:09.400	0.469460	-0.799782	0.178449	0.101537	0.219465	0.803141	1	0	0	0	0
2019-10-14 09:44:18.800	0.475820	-0.835328	0.174250	0.113403	0.221300	0.798150	1	0	0	0	0
2019-10-14 09:44:28.200	0.504157	-0.822037	0.188928	0.108004	0.230099	0.783237	1	0	0	0	0
2019-10-14 09:44:37.600	0.489660	-0.822526	0.195269	0.109591	0.234475	0.780564	1	0	0	0	0
2019-10-14 09:44:47.000	0.501598	-0.843972	0.188185	0.120230	0.240469	0.792553	1	0	0	0	0

1.1 Preparing feature dataset for learning

1.1.1 Splitting in x and y

[7]:
```
x = features_dataset[features_columns[:-1]]
y = features_dataset[activity_columns]

## split
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.2,␣
↪random_state=0, stratify=y)
```

1.2 scale training and validation set for model

[8]:
```
x_train
```

[8]:
```
                         standard_deviation_x    mean_x  standard_deviation_y  \
2019-09-30 11:40:17.400              0.514813 -1.000000              0.492874
2019-10-14 12:10:54.400              0.000000 -0.174603              0.000000
2019-10-09 12:08:12.800              0.837709 -0.972982              0.591663
2019-10-02 11:49:21.200              0.460197 -0.743837              0.150597
2019-09-16 15:16:23.200              0.507802 -1.073793              0.318954
…                                         …       …                     …
2019-09-30 13:22:42.800              0.036894 -1.061543              0.122541
2019-10-10 12:58:42.200              0.005906 -0.330716              0.000000
2019-10-08 13:35:37.200              0.015342 -0.480412              0.023013
2019-10-02 10:49:04.400              0.488098 -1.092790              0.276464
2019-10-14 10:13:40.200              0.855066 -0.967663              0.347789

                           mean_y  standard_deviation_z    mean_z
2019-09-30 11:40:17.400  0.045255              0.413718  0.122678
2019-10-14 12:10:54.400  0.380952              0.001158 -0.888804
2019-10-09 12:08:12.800  0.029889              0.563101  0.259710
2019-10-02 11:49:21.200  0.194360              0.207433  0.820078
2019-09-16 15:16:23.200  0.025160              0.509101  0.148007
…                             …                     …       …
2019-09-30 13:22:42.800  0.049754              0.155300  0.078251
2019-10-10 12:58:42.200  0.158730              0.000000  1.126984
2019-10-08 13:35:37.200 -0.204745              0.023013  1.112715
2019-10-02 10:49:04.400  0.180682              0.594954  0.037150
2019-10-14 10:13:40.200  0.206265              0.679888 -0.124367

[3396 rows x 6 columns]
```

2 Random tree forest

[9]:
```
ftc = RandomForestClassifier(n_estimators=53, random_state=0)
ftc.fit(x_train, y_train)
```

[9]:
```
RandomForestClassifier(n_estimators=53, random_state=0)
```

2.1 Validation result

[10]:
```
predictions = ftc.predict(x_valid)
```

Accuracy

[11]:
```
accuracy_score(y_valid, predictions, normalize=True)
```

[11]: 0.9858823529411764

F1

[12]:
```
f1_score(y_valid, predictions, average='micro')
```

[12]: 0.9864626250735727

Recall

[13]:
```
recall_score(y_valid, predictions, average='micro')
```

[13]: 0.9858823529411764

Precision

[14]:
```
precision_score(y_valid, predictions, average='micro')
```

[14]: 0.9870435806831567

Classification report

[15]:
```
print(classification_report(y_valid, predictions,␣
 ↪target_names=activity_columns, zero_division=0))
```

```
                  precision    recall  f1-score   support

 activity_cycling       1.00      1.00      1.00       282
 activity_walking       0.99      0.98      0.99       145
 activity_running       0.98      0.97      0.98       141
activity_standing       0.96      0.99      0.97       141
  activity_sitten       1.00      0.98      0.99       141

        micro avg       0.99      0.99      0.99       850
        macro avg       0.99      0.98      0.98       850
     weighted avg       0.99      0.99      0.99       850
      samples avg       0.99      0.99      0.99       850
```

2.1.1 Confusion matrix

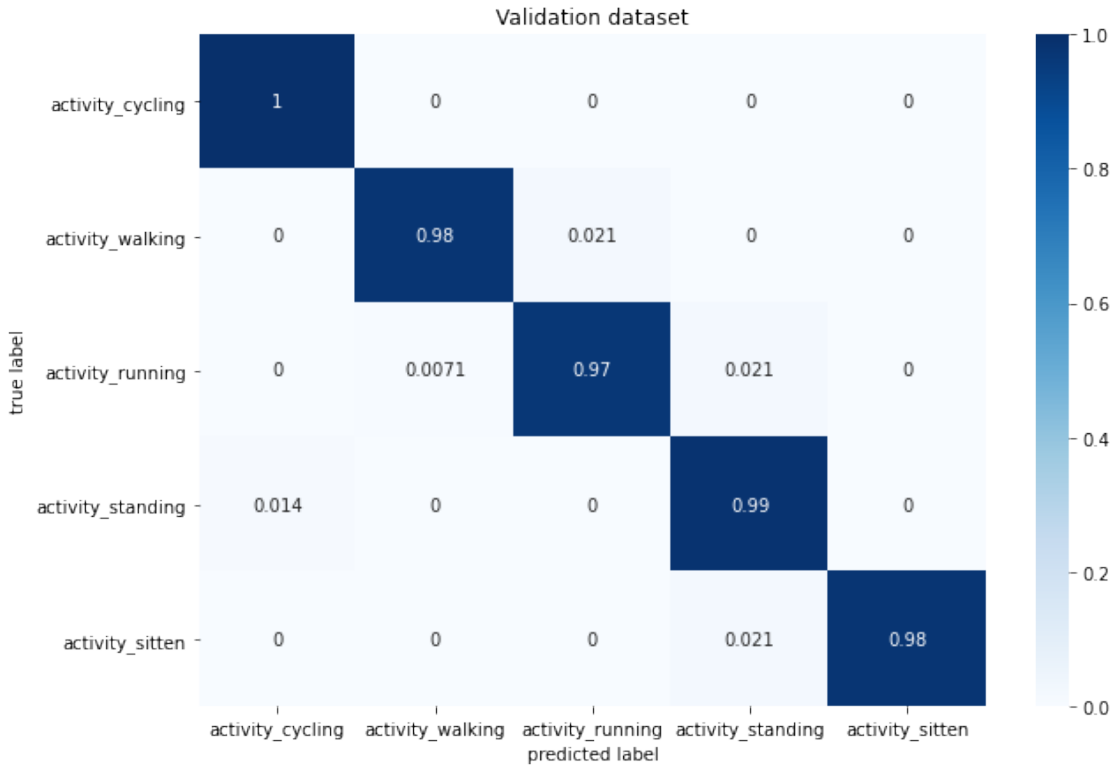
[16]:
```
import seaborn as sn

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

df_cm = pd.DataFrame(cm, index=activity_columns, columns=activity_columns)
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")
```

[16]:
```
Text(68.09375, 0.5, 'true label')
```



2.2 Test result

[17]:
```
test_dataset = extract_features_from_correspondents(test_users)

test_dataset[activity_columns] = 0

#features_dataset.loc[(features_dataset['activiteit'] == 'springen'),␣
↪'activity_jumping'] = 1
#features_dataset.loc[(features_dataset['activiteit'] == 'traplopen'),␣
↪'activity_traplopen'] = 1
test_dataset.loc[(test_dataset['activiteit'] == 'lopen'), 'activity_walking'] =␣
↪1
test_dataset.loc[(test_dataset['activiteit'] == 'rennen'), 'activity_running']␣
↪= 1
test_dataset.loc[(test_dataset['activiteit'] == 'staan'), 'activity_standing']␣
↪= 1
test_dataset.loc[(test_dataset['activiteit'] == 'zitten'), 'activity_sitten'] =␣
↪1
test_dataset.loc[(test_dataset['activiteit'] == 'fietsen licht'),␣
↪'activity_cycling'] = 1
test_dataset.loc[(test_dataset['activiteit'] == 'fietsen zwaar'),␣
↪'activity_cycling'] = 1

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

x = test_dataset[features_columns[:-1]]
y = test_dataset[activity_columns]
```

```
3
Extracting BMR004
Extracting BMR034
Extracting BMR097
Done extracting features
```

[18]:
```
test_prediction_y = ftc.predict(x)
```

accuracy

[19]:
```
accuracy_score(y, test_prediction_y, normalize=True)
```

[19]:
```
0.9846153846153847
```

F1

[20]:
```
f1_score(y, test_prediction_y, average='micro')
```

[20]: 0.9854576561163388

Recall

[21]:
```
recall_score(y, test_prediction_y, average='micro')
```

[21]: 0.9846153846153847

Precision

[22]:
```
precision_score(y, test_prediction_y, average='micro')
```

[22]: 0.9863013698630136

[23]:
```
print(classification_report(y,test_prediction_y, target_names=activity_columns,␣
↪zero_division=0))
```

```
                  precision    recall  f1-score   support

 activity_cycling       1.00      1.00      1.00       192
 activity_walking       0.95      1.00      0.97        97
 activity_running       1.00      0.92      0.96       104
activity_standing       0.97      0.99      0.98        96
  activity_sitten       1.00      1.00      1.00        96

        micro avg       0.99      0.98      0.99       585
        macro avg       0.98      0.98      0.98       585
     weighted avg       0.99      0.98      0.99       585
      samples avg       0.98      0.98      0.98       585
```

2.3 k-fold cross validation

[24]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold, KFold
import seaborn as sn
from sklearn.model_selection import cross_val_predict

test_respondents = ['BMR004', 'BMR034', 'BMR097']

dataset = features_dataset

x = dataset[features_columns[:-1]]
y = dataset[activity_columns]

accuracy_scores = cross_val_score( RandomForestClassifier(n_estimators=53,␣
↪random_state=0), x, y, cv=5, scoring='accuracy')
```

```
recall_scores = cross_val_score( RandomForestClassifier(n_estimators=53,␣
 ↪random_state=0), x, y , cv=5, scoring='recall_micro')
precision_scores = cross_val_score( RandomForestClassifier(n_estimators=53,␣
 ↪random_state=0), x, y , cv=5, scoring='precision_micro')

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

```
Accuracy: 0.96 (+/- 0.04)
Recall: 0.96 (+/- 0.04)
Precision: 0.97 (+/- 0.03)
```

3 save model

[25]:
```
from joblib import dump

dump(ftc, 'activity.dat')
```

[25]:
```
['activity.dat']
```