all_steps_activity recognition_final_version_split_cycling_12_1_seconds

January 11, 2021

[2]:

```
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

[3]:

```
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_light', 'activity_cycling_heavy',␣
 ↪'activity_walking', 'activity_running', 'activity_standing',␣
 ↪'activity_sitten']
activities = ['fietsen licht', 'fietsen zwaar', 'lopen', 'rennen', 'staan',␣
 ↪'zitten']
```

```
test_users = ['BMR004', 'BMR034', 'BMR097']
segment_size = 12.1
number_of_trees = 93
```

[4]:
```
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)]

                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
```

[5]:

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

    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)
```

[6]:

```
features_dataset = extract_features_from_all_correspondents()
```

```
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

[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'] == 'staan'),␣
 ↪'activity_standing'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'zitten'),␣
 ↪'activity_sitten'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'fietsen licht'),␣
 ↪'activity_cycling_light'] = 1
features_dataset.loc[(features_dataset['activiteit'] == 'fietsen zwaar'),␣
 ↪'activity_cycling_heavy'] = 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  \
2019-10-14 09:44:12.100              0.473065 -0.807228              0.173984
2019-10-14 09:44:24.200              0.489663 -0.819975              0.187111
2019-10-14 09:44:36.300              0.489649 -0.844156              0.189527
2019-10-14 09:44:48.400              0.504203 -0.813262              0.195295
2019-10-14 09:45:00.500              0.483359 -0.839368              0.181478

                           mean_y  standard_deviation_z    mean_z  \
2019-10-14 09:44:12.100  0.099108              0.217114  0.804670
2019-10-14 09:44:24.200  0.115357              0.231906  0.786661
2019-10-14 09:44:36.300  0.115309              0.228367  0.785190
2019-10-14 09:44:48.400  0.110127              0.239517  0.782500
2019-10-14 09:45:00.500  0.118851              0.233384  0.791683

                         activity_cycling_light  activity_cycling_heavy  \
2019-10-14 09:44:12.100                       1                       0
2019-10-14 09:44:24.200                       1                       0
2019-10-14 09:44:36.300                       1                       0
2019-10-14 09:44:48.400                       1                       0
2019-10-14 09:45:00.500                       1                       0

                         activity_walking  activity_running  \
2019-10-14 09:44:12.100                 0                 0
2019-10-14 09:44:24.200                 0                 0
2019-10-14 09:44:36.300                 0                 0
2019-10-14 09:44:48.400                 0                 0
2019-10-14 09:45:00.500                 0                 0

                         activity_standing  activity_sitten
2019-10-14 09:44:12.100                  0                0
2019-10-14 09:44:24.200                  0                0
2019-10-14 09:44:36.300                  0                0
2019-10-14 09:44:48.400                  0                0
2019-10-14 09:45:00.500                  0                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]

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

2 Random tree forest

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

[9]:
```
RandomForestClassifier(n_estimators=93, random_state=0)
```

2.1 Validation result

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

Accuracy

[11]:
```
accuracy_score(y_valid, predictions)
```

[11]:
```
0.963855421686747
```

F1

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

[12]:
```
0.963855421686747
```

Precision

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

[13]:
```
0.963855421686747
```

Recall

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

[14]:
```
0.963855421686747
```

Classification report

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

```
                       precision    recall  f1-score   support

activity_cycling_light       0.90      0.94      0.92       107
activity_cycling_heavy       0.94      0.90      0.92       110
      activity_walking       0.97      1.00      0.99       113
```

```
 activity_running       1.00      0.95      0.98       109
activity_standing       0.98      0.99      0.98       122
  activity_sitten       0.99      0.99      0.99       103

        micro avg       0.96      0.96      0.96       664
        macro avg       0.96      0.96      0.96       664
     weighted avg       0.96      0.96      0.96       664
      samples avg       0.96      0.96      0.96       664
```

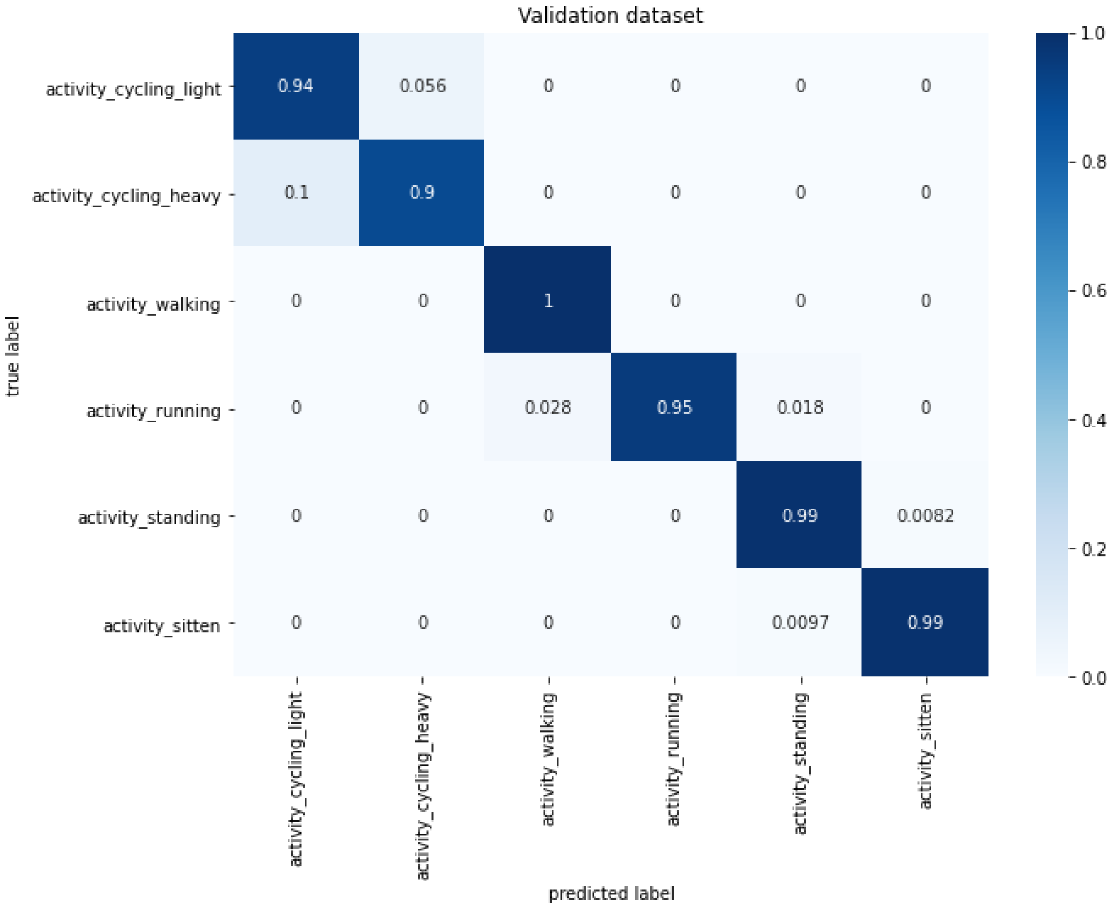
Confusion matrix

[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')
```



k-fold cross validation

[17]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold, KFold
import seaborn as sn
from sklearn.model_selection import cross_val_predict

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

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

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

```
Accuracy: 0.82 (+/- 0.05)
Precision: 0.84 (+/- 0.04)
Recall: 0.82 (+/- 0.05)
```

2.2 Test result

[18]:

```
test_dataset = extract_features_from_correspondents(test_users)

test_dataset[activity_columns] = 0

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_light'] = 1
test_dataset.loc[(test_dataset['activiteit'] == 'fietsen zwaar'),␣
 ↪'activity_cycling_heavy'] = 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]
```

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

[19]:

```
test_prediction_y = ftc.predict(x)
```

accuracy

[20]:

```
accuracy_score(y, test_prediction_y, normalize=True)
```

[20]:

```
0.8468271334792122
```

F1

[21]:

```
f1_score(y, test_prediction_y, average='micro')
```

[21]:

```
0.8486842105263157
```

Precision

[22]:

```
precision_score(y, test_prediction_y, average='micro')
```

[22]:

```
0.8505494505494505
```

Recall

[23]:

```
recall_score(y, test_prediction_y, average='micro')
```

[23]:

```
0.8468271334792122
```

[24]:

```
#### classification report
```

[25]:

```
print(classification_report(y,test_prediction_y, target_names=activity_columns,␣
 ↪zero_division=0))
```

```
                       precision    recall  f1-score   support

activity_cycling_light       0.55      0.77      0.64        75
activity_cycling_heavy       0.64      0.37      0.47        75
      activity_walking       0.96      1.00      0.98        76
      activity_running       1.00      0.93      0.96        81
     activity_standing       0.97      1.00      0.99        75
       activity_sitten       1.00      1.00      1.00        75

             micro avg       0.85      0.85      0.85       457
             macro avg       0.85      0.85      0.84       457
          weighted avg       0.86      0.85      0.84       457
           samples avg       0.85      0.85      0.85       457
```

[26]:

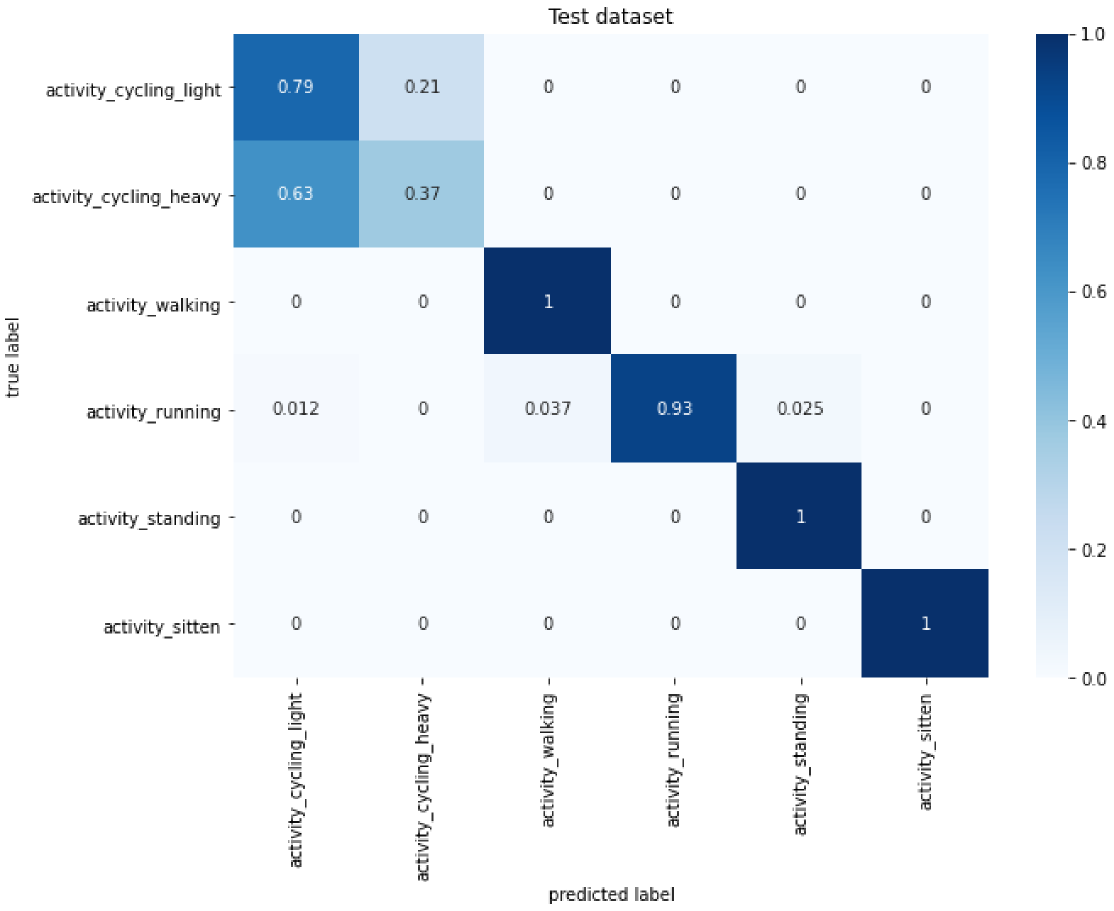
```
#### Confusion matrix
```

[27]:

```
import seaborn as sn
```

```
cm = confusion_matrix(y.values.argmax(axis=1), test_prediction_y.
 ↪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("Test dataset")
plt.xlabel("predicted label")
plt.ylabel("true label")
```

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



3 save model

[28]:

```
#from joblib import dump

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