all_steps_activity recognition_v2_analysis

January 12, 2021

[ ]:
```
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 matplotlib.pyplot as plt
```

1 Feature Extraction

[ ]:
```
activpal = Activpal()

features_columns = ['standard_deviation_x', 'mean_x', 'standard_deviation_y',␣
↪'mean_y','standard_deviation_z', 'mean_z', 'activiteit']
activities = ['lopen', 'rennen', 'springen', 'staan', 'traplopen', 'zitten']
activity_columns = ['activity_walking', 'activity_running', 'activity_jumping',␣
↪'activity_standing', 'activity_traplopen', 'activity_sitten']

segment_size = 6.4
```

[ ]:
```
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
```

[ ]:
```
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']:
                    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

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

    for subject in test_subject:
        print("Extracting " + subject)
        features_df = extract_features_from_correspondent(subject)
        all_features_df = pd.concat([all_features_df, features_df])

    print("Done extracting features from test users")

    return all_features_df
```

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

2 Balancing dataset

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

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

    return unbalanced_dataset

#features_dataset = balance_dataset_by_activity(features_dataset)
```

[ ]:

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

3 model preperation

[ ]:
```
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.drop('activiteit', axis=1, inplace=True)
```

[ ]:
```
#fill_na_columns = ['standard_deviation_x', 'mean_x', 'standard_deviation_y',␣
↪'mean_y','standard_deviation_z',
#                    'mean_z']

#for column in fill_na_columns:
#    features_dataset[column].fillna(0, inplace=True)
features_dataset.dropna(how='any', inplace=True)
```

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

3.1 Preparing feature dataset for learning

3.1.1 Splitting in x and y

[ ]:
```
x = features_dataset[features_columns[:-1]]
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)
```

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

4 Random tree forest

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

4.1 Testing and results

[ ]:
```
prediction_y = rfc.predict(valid_x)
```

4.1.1 Result

Accuracy

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

Classification report

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

[ ]:
```
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)
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")
```

5 Result Summary

Random seed: 23 n_estimators: 20

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%

5.1 Diagnostics

5.1.1 Cross validation analysis

[ ]:
```
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)
```

[ ]:
```
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))
```

[ ]:
```
import seaborn as sn
from sklearn.model_selection import cross_val_predict

cm = confusion_matrix(y.values.argmax(axis=1), pred_y.argmax(axis=1),␣
 ↪normalize='true')

df_cm = pd.DataFrame(cm, index=activities, columns=activities)

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")
```

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

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

[ ]:
```
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'))
```

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

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

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

6 hyper parameter optimalization

6.1 n_estimator optimalization

[ ]:
```
accuracy_scores = []
recall_scores = []
precision_scores = []

n_estimator_numbers = range(1,200)

for i in n_estimator_numbers:
    rfc = RandomForestClassifier(n_estimators=i, random_state=0)
    rfc.fit(train_x, train_y)

    predictions = rfc.predict(valid_x)

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

[ ]:
```
plt.plot(n_estimator_numbers, accuracy_scores, label='accuracy')
plt.plot(n_estimator_numbers, recall_scores, label='recall')
plt.plot(n_estimator_numbers, precision_scores, label='precision')

plt.legend()
```

6.2 Test set analysis

[ ]:
```
test_set_features = extract_features_from_all_test_correspondents()
```

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

test_set_features[['activity_walking_running', 'activity_jumping',␣
↪'activity_standing', 'activity_traplopen',
                   'activity_sitten']] = 0

test_set_features.loc[(test_set_features['activiteit'] == 'lopen'),␣
↪'activity_walking_running'] = 1
test_set_features.loc[(test_set_features['activiteit'] == 'rennen'),␣
↪'activity_walking_running'] = 1
test_set_features.loc[(test_set_features['activiteit'] == 'springen'),␣
↪'activity_jumping'] = 1
test_set_features.loc[(test_set_features['activiteit'] == 'staan'),␣
↪'activity_standing'] = 1
test_set_features.loc[(test_set_features['activiteit'] == 'traplopen'),␣
↪'activity_traplopen'] = 1
test_set_features.loc[(test_set_features['activiteit'] == 'zitten'),␣
↪'activity_sitten'] = 1

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

test_set_features
```

[ ]:
```
test_x = test_set_features[['standard_deviation_x', 'mean_x',␣
↪'standard_deviation_y', 'mean_y','standard_deviation_z', 'mean_z']]
test_y = test_set_features[['activity_walking_running', 'activity_jumping',␣
↪'activity_standing', 'activity_traplopen', 'activity_sitten']]

prediction_y = rfc.predict(test_x)
```

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

[ ]:
```
print(classification_report(test_y,prediction_y,␣
↪target_names=['activity_walking_running' 'activity_jumping',␣
↪'activity_standing', 'activity_traplopen',
                   'activity_sitten'], zero_division=0))
```

[ ]:
```

```