intensity_classification_model_backup

January 12, 2021

1 Intensity recognition

Activity intensity	MET range
Light	< 3.00
Moderate	3.00 - 5.99
Hard	6.00 - 8.99
Very hard	> 8.99

Activty count cut-points bu : Freedson et al.(1998)

[4]:

```
from helpers                 import pandas_helper as pdh, math_helper as mth
from utils                   import read_functions
from sensors.activpal        import *
from sensors.vyntus          import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model    import LogisticRegression
from sklearn.ensemble        import RandomForestClassifier
from sklearn.metrics         import confusion_matrix, classification_report,␣
 ↪accuracy_score, precision_score, recall_score, roc_auc_score, f1_score

import matplotlib.pyplot as plt
import pandas            as pd

activpal = Activpal()
vyntus = Vyntus()

respondents_df = pdh.read_csv_respondents()
test_users     = ['BMR032', 'BMR042', 'BMR098']
```

2 Preparing dataset

[5]:
```
def get_vyntus_df(correspondent, start, stop):
    intensity_intervals = [0, 3, 5.99, 8.99, 1000]
    intensity_labels    = ['Light', 'Moderate', 'Hard', 'Very Hard']

    vyntus_df = vyntus.read_data(correspondent, start, stop)

    if vyntus_df.empty:
        return pd.DataFrame()

    corr_number = int(correspondent.replace('BMR0', ''))

    weight = respondents_df['gewicht'][corr_number]
    length_m = respondents_df['lengte'][corr_number] / 100

    vyntus_df['vyn_VO2'] = [float(vo2.replace(',', '.')) if type(vo2) == str␣
 ↪else vo2 for vo2 in vyntus_df['vyn_VO2']]
    vyntus_df['met'] = mth.calculate_met(vyntus_df['vyn_VO2'], weight)
    vyntus_df['weight'] = weight
    vyntus_df['bmi'] = mth.calculate_bmi(weight, length_m)

    vyntus_df =  vyntus_df.resample('60s').mean()[:-1]

    vyntus_df['intensity'] = pd.cut(vyntus_df.met, intensity_intervals,␣
 ↪labels=intensity_labels)

    return vyntus_df;
```

[6]:
```
def get_activpal_df(correspondent, start, stop):
    activpal_df = activpal.read_data(correspondent, start, stop)

    activpal_df = activpal_df[['pal_accX', 'pal_accY', 'pal_accZ']].apply(mth.
 ↪convert_value_to_g)

    x_abs = abs(activpal_df['pal_accX'])
    y_abs = abs(activpal_df['pal_accY'])
    z_abs = abs(activpal_df['pal_accZ'])

    activpal_df['mag_acc'] = mth.to_mag_acceleration(x_abs, y_abs, z_abs)

    return activpal_df.resample('60s').sum()[:-1]
```

[7]:
```
def get_dataset_of_correspondent(correspondent):
    dataset_df = pd.DataFrame(columns=['sum_mag_of_acc', 'mean_met',␣
 ↪'intensity', 'activity'], index=pd.to_datetime([]))
```

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

    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 = get_activpal_df(correspondent, start_time, stop_time)
            vyntus_df = get_vyntus_df(correspondent, start_time, stop_time)

            if not vyntus_df.empty and not activpal_df.empty:
                activity_dataset_df = pd.DataFrame(index=activpal_df.index)

                activity_dataset_df['sum_mag_of_acc'] = activpal_df['mag_acc']
                activity_dataset_df['mean_met']       = vyntus_df['met']
                activity_dataset_df['weight']         = vyntus_df['weight']
                activity_dataset_df['bmi']            = vyntus_df['bmi']
                activity_dataset_df['intensity']      = vyntus_df['intensity']
                activity_dataset_df['activity']       = activity_name

                activity_dataset_df.dropna(how='any', inplace=True)

                dataset_df = pd.concat([activity_dataset_df, dataset_df])

    return dataset_df
```

[8]:

```
def create_dataset_from_correspondents(correspodents):
    dataset_df = pd.DataFrame(index=pd.to_datetime([]))

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

        correspondent_dataset_df = get_dataset_of_correspondent(correspodent)
        dataset_df = pd.concat([dataset_df, correspondent_dataset_df])


    print("Done creating dataset")

    return dataset_df

def create_dataset_from_all_correspondents(exclude_test_correspodent = True):
    exclude = ['output', 'throughput', 'Test data','.ipynb_checkpoints',␣
↪'BRM015', 'BMR035', 'BMR100', 'BMR051', 'BMR027']

    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 create_dataset_from_correspondents(correspodents)
```

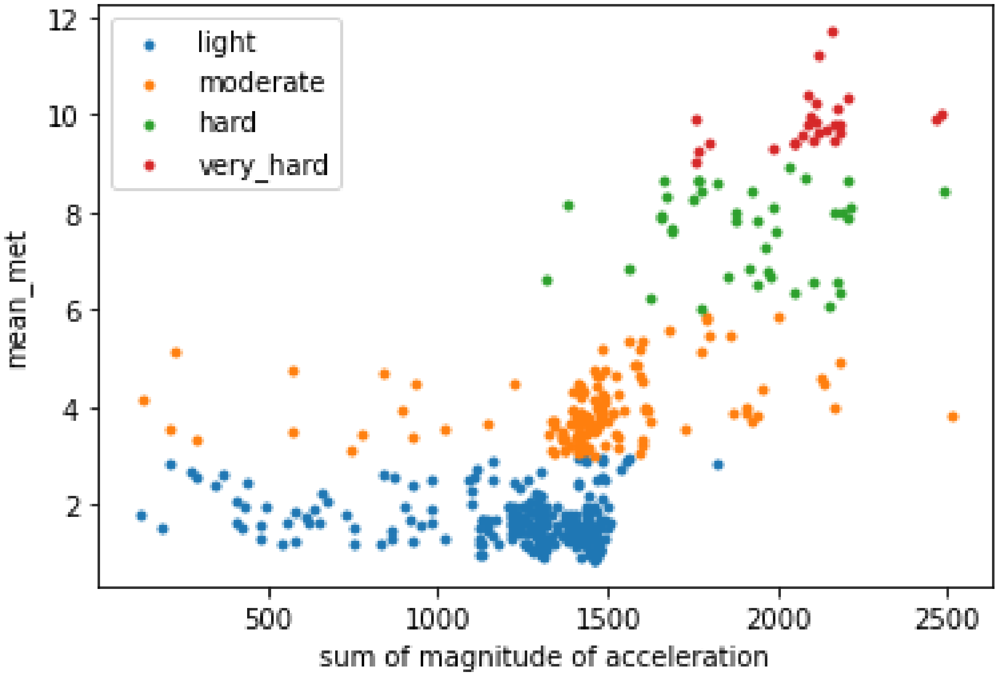
[9]:
```
dataset = create_dataset_from_all_correspondents()
```

```
Extracting BMR099
Extracting BMR025
Could not read file: ../../data/BMR025/vyntus.csv
Could not read file: ../../data/BMR025/vyntus.csv
Could not read file: ../../data/BMR025/vyntus.csv
Could not read file: ../../data/BMR025/vyntus.csv
Could not read file: ../../data/BMR025/vyntus.csv
Could not read file: ../../data/BMR025/vyntus.csv
Extracting BMR060
No data for respondnet: BMR060
No data for respondnet: BMR060
No data for respondnet: BMR060
No data for respondnet: BMR060
No data for respondnet: BMR060
No data for respondnet: BMR060
Extracting BMR012
Extracting BMR030
Extracting BMR044
Extracting BMR043
Extracting BMR004
Extracting BMR011
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 BMR018
Extracting BMR058
Extracting BMR040
Done creating dataset
```

2.1 Dataset analysis

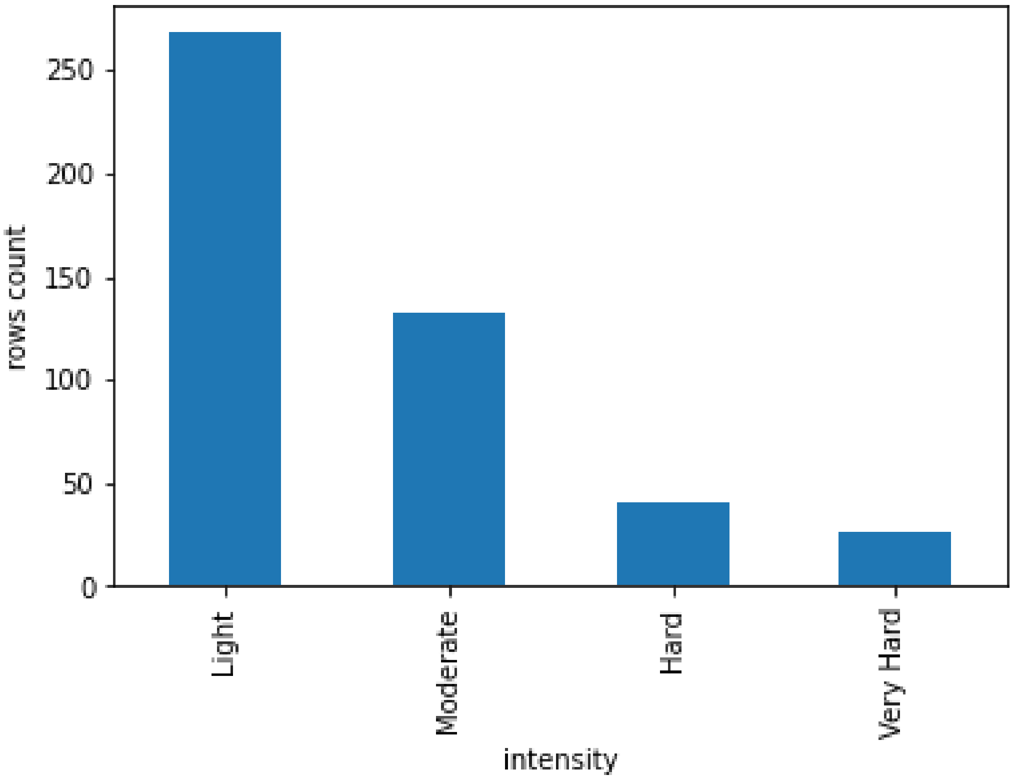
[10]:

```
def plot_intensity_on_accel(dataset):
    light = dataset.loc[(dataset['intensity'] == 'Light') ]
    moderate = dataset.loc[(dataset['intensity'] == 'Moderate') ]
    hard = dataset.loc[(dataset['intensity'] == 'Hard') ]
    very_hard = dataset.loc[(dataset['intensity'] == 'Very Hard') ]

    plt.scatter(light['sum_mag_of_acc'],     light['mean_met'],    marker='.',␣
↪label='light')
    plt.scatter(moderate['sum_mag_of_acc'],  moderate['mean_met'], marker='.',␣
↪label='moderate')
    plt.scatter(hard['sum_mag_of_acc'],      hard['mean_met'],     marker='.',␣
↪label='hard')
    plt.scatter(very_hard['sum_mag_of_acc'], very_hard['mean_met'],marker='.',␣
↪label='very_hard')


    plt.ylabel('mean_met')
    plt.xlabel('sum of magnitude of acceleration')
    plt.legend()
    plt.show()

plot_intensity_on_accel(dataset)
```



```
[11]: dataset['intensity'].value_counts().plot.bar(ylabel='rows␣
       ↪count',xlabel='intensity')
```

```
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff0a16f20b8>
```



2.2 Defining categories

[12]:
```
intensity_columns          = ['intensity_light', 'intensity_moderate',␣
↪'intensity_hard', 'intensity_very_hard']
dataset[intensity_columns] = 0

dataset.loc[(dataset['intensity'] == 'Light'),    'intensity_light']      = 1
dataset.loc[(dataset['intensity'] == 'Moderate'), 'intensity_moderate']   = 1
dataset.loc[(dataset['intensity'] == 'Hard'),     'intensity_hard']       = 1
dataset.loc[(dataset['intensity'] == 'Very Hard'),'intensity_very_hard']  = 1

dataset.dropna(how='any', inplace=True)
```

2.3 Splitting dataset

[13]:
```
x = dataset[['sum_mag_of_acc', 'bmi']]
y = dataset[intensity_columns]
```

```
train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=23, stratify=y)

len(x)
```

[13]:
```
467
```

2.4 Training dataset

[25]:
```
rfc = RandomForestClassifier(n_estimators=23,random_state=0)
rfc.fit(train_x, train_y)
```

[25]:
```
RandomForestClassifier(n_estimators=23, random_state=0)
```

[26]:
```
prediction_y = rfc.predict(valid_x)
```

2.5 Result analysis

2.5.1 Validation

[27]:
```
accuracy_score(valid_y, prediction_y, normalize=True)
```

[27]:
```
0.7446808510638298
```

[28]:
```
print(classification_report(valid_y, prediction_y,␣
 ↪target_names=intensity_columns, zero_division=0))
```

```
                    precision    recall  f1-score   support

   intensity_light       0.87      0.96      0.91        54
intensity_moderate       0.80      0.44      0.57        27
    intensity_hard       0.40      0.50      0.44         8
intensity_very_hard       0.40      0.40      0.40         5

         micro avg       0.78      0.74      0.76        94
         macro avg       0.62      0.58      0.58        94
      weighted avg       0.78      0.74      0.75        94
       samples avg       0.74      0.74      0.74        94
```

2.5.2 Test

[29]:

```
test_dataset = create_dataset_from_correspondents(test_users)

test_dataset[intensity_columns] = 0

test_dataset.loc[(test_dataset['intensity'] == 'Light'),    'intensity_light'] ␣
 ↪      = 1
test_dataset.loc[(test_dataset['intensity'] == 'Moderate'),␣
 ↪'intensity_moderate']     = 1
test_dataset.loc[(test_dataset['intensity'] == 'Hard'),     'intensity_hard']  ␣
 ↪      = 1
test_dataset.loc[(test_dataset['intensity'] == 'Very␣
 ↪Hard'),'intensity_very_hard']    = 1

test_dataset.dropna(how='any', inplace=True)

x = test_dataset[['sum_mag_of_acc', 'bmi']]
y = test_dataset[intensity_columns]
```

```
Extracting BMR032
Extracting BMR042
Extracting BMR098
Done creating dataset
```

[30]:

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

[31]:

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

[31]:

```
0.6610169491525424
```

[32]:

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

```
                     precision    recall  f1-score   support

    intensity_light       0.79      0.79      0.79        33
 intensity_moderate       0.50      0.39      0.44        18
     intensity_hard       0.60      0.60      0.60         5
intensity_very_hard       1.00      1.00      1.00         3

          micro avg       0.71      0.66      0.68        59
          macro avg       0.72      0.69      0.71        59
       weighted avg       0.69      0.66      0.68        59
        samples avg       0.66      0.66      0.66        59
```

2.6 Tune hyperparameteres

[22]:

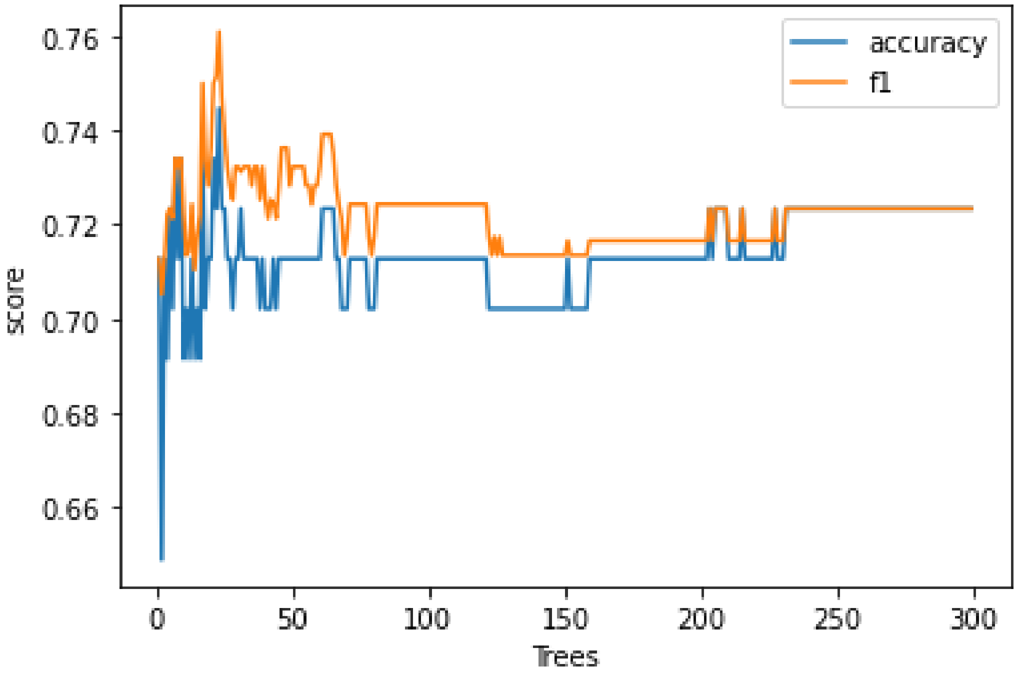
```
#### Quick analysis
accuracy_scores = []
f1_scores = []
#precision_scores = []

n_estimator_numbers = range(1,300)

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

    predictions = rfc_t.predict(valid_x)

    accuracy_scores.append(accuracy_score(valid_y, predictions, normalize=True))
    f1_scores.append(f1_score(valid_y, predictions, average='micro' ))
    #precision_scores.append(precision_score(valid_y, predictions,␣
 ↪average='micro'))
```

[23]:

```
plt.plot(n_estimator_numbers, accuracy_scores, label='accuracy')
plt.plot(n_estimator_numbers, f1_scores, label='f1')

plt.xlabel('Trees')
plt.ylabel('score')
#plt.plot(n_estimator_numbers, precision_scores, label='precision')

plt.legend()
```

[23]:

```
<matplotlib.legend.Legend at 0x7ff0a182ac18>
```



[24]:
```
np_accuracy_scores = np.array(accuracy_scores)
np_f1_scores = np.array(f1_scores)

best_accuracy_index = np.argmax(np_accuracy_scores)
best_f1_index = np.argmax(np_f1_scores)

print('accuracy: ', n_estimator_numbers[best_accuracy_index])
print('f1: ', n_estimator_numbers[best_f1_index])
```

```
accuracy:  23
f1:  23
```

[ ]: