intensity_classification_model

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)

[2]:

```
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

[3]:

```
def get_vyntus_df(correspondent, start, stop):
    intensity_intervals = [0, 3, 5.99, 1000]
    intensity_labels    = ['Light', 'Moderate', 'Hard and 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_kg'] = weight
    vyntus_df['length_m'] = length_m
    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;
```

[4]:

```
def get_speed(magnitude_accelerations):
    activpal_time = 0.05

    return [sum(magnitude_accelerations[:i]) * activpal_time for i in␣
 ↪range(len(magnitude_accelerations))]


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)
    activpal_df['speed']   = get_speed(activpal_df['mag_acc'])

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

[5]:
```
def get_dataset_of_correspondent(correspondent):
    dataset_df = pd.DataFrame(columns=['sum_mag_of_acc', 'mean_met',␣
 ↪'intensity', 'activity'], index=pd.to_datetime([]))
    activities         = ['fietsen licht', 'fietsen zwaar', '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['speed']          = activpal_df['speed']
                activity_dataset_df['mean_met']       = vyntus_df['met']
                activity_dataset_df['weight_kg']      = vyntus_df['weight_kg']
                activity_dataset_df['length_m']       = vyntus_df['length_m']
                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
```

[6]:
```
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','BMR025','BMR060', '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)
```

[7]:

```
dataset = create_dataset_from_all_correspondents()
```

```
Extracting BMR099
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
```

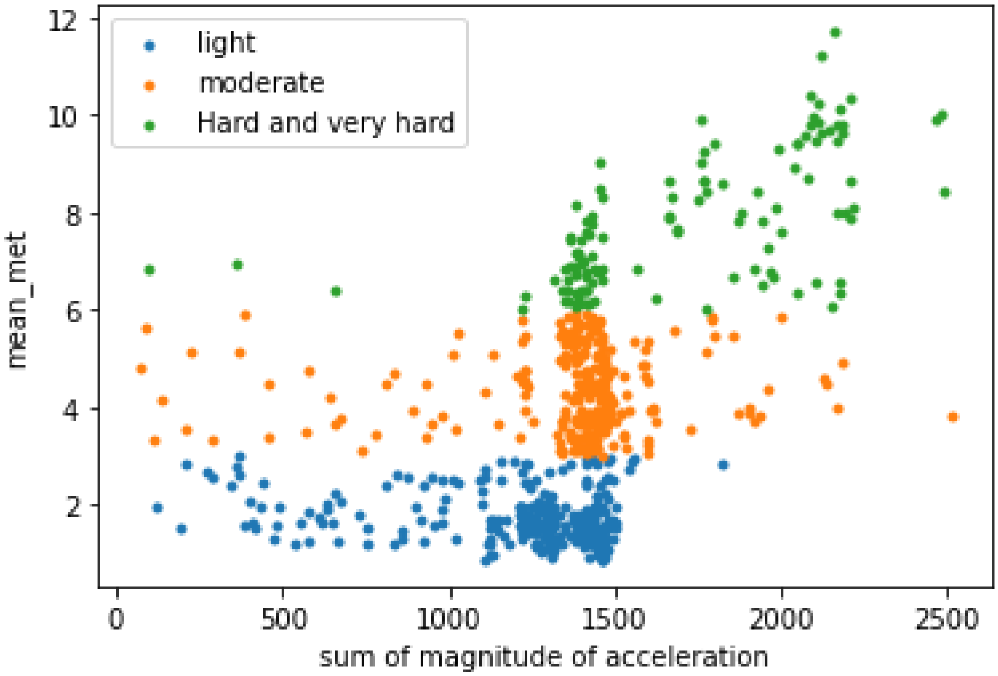
[8]:

```
dataset = dataset[~(dataset['activity'] == 'traplopen')]
```

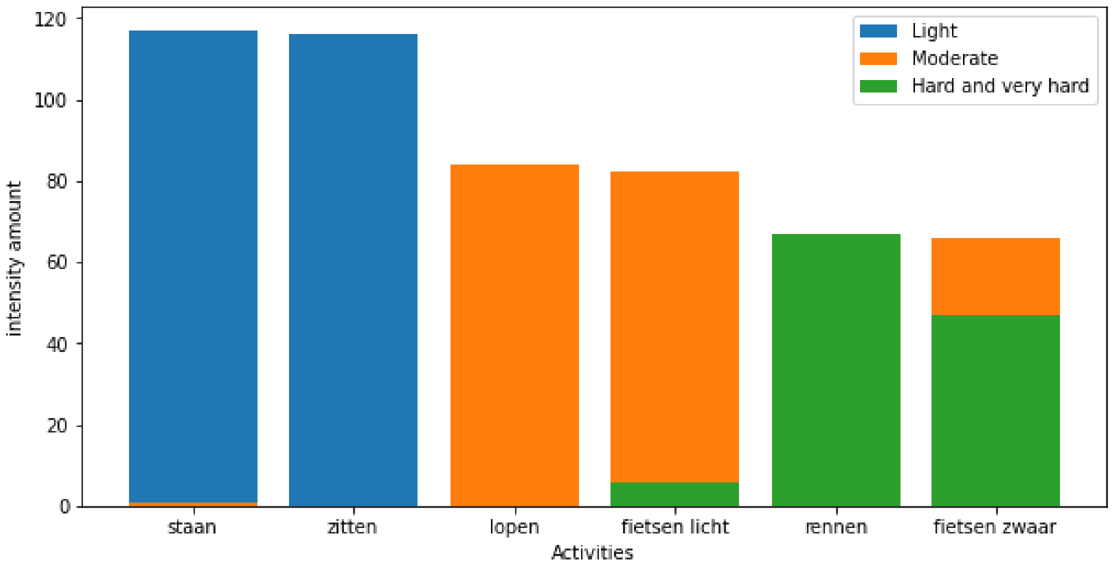
2.1 Dataset analysis

[9]:

```
def plot_intensity_on_accel(dataset):
    light = dataset.loc[(dataset['intensity'] == 'Light') ]
    moderate = dataset.loc[(dataset['intensity'] == 'Moderate') ]
    hard_and_very_hard = dataset.loc[(dataset['intensity'] == 'Hard and 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_and_very_hard['sum_mag_of_acc'],  ␣
 ↪hard_and_very_hard['mean_met'], marker='.', label='Hard and very hard')


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

plot_intensity_on_accel(dataset)
```

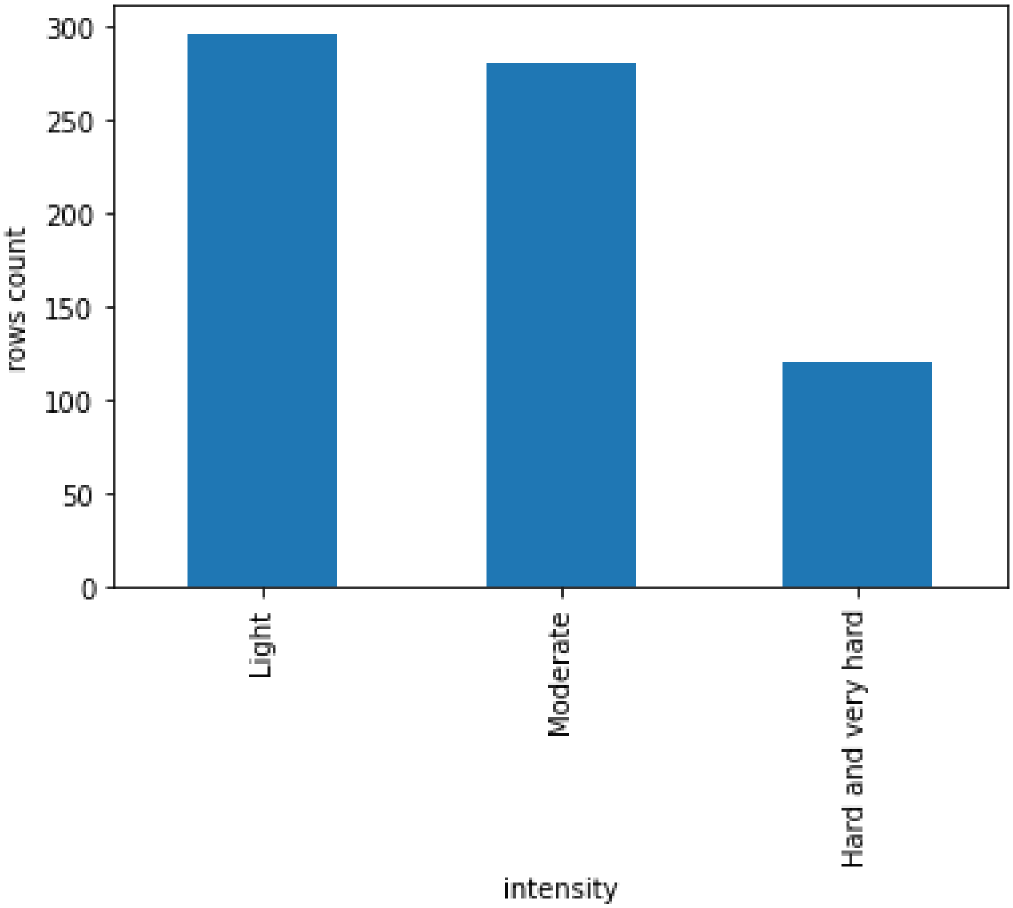


[10]:
```python
def plot_intensity_on_activity(dataset):
    light = dataset.loc[(dataset['intensity'] == 'Light') ]
    moderate = dataset.loc[(dataset['intensity'] == 'Moderate') ]
    hard = dataset.loc[(dataset['intensity'] == 'Hard and very hard') ]

    activities_light = light['activity'].value_counts()
    activities_moderate = moderate['activity'].value_counts()
    activities_hard = hard['activity'].value_counts()

    plt.figure(figsize=(10,5))

    plt.bar(activities_light.index, activities_light.tolist(), label='Light')
    plt.bar(activities_moderate.index, activities_moderate.tolist(),␣
 ↪label='Moderate')
    plt.bar(activities_hard.index, activities_hard.tolist(), label='Hard and␣
 ↪very hard')

    plt.ylabel('intensity amount')
    plt.xlabel('Activities')
    plt.legend()
    plt.show()

plot_intensity_on_activity(dataset)
```



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

```
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fadf6c1deb8>
```



2.2 Defining categories

[12]:
```
intensity_columns          = ['intensity_light', 'intensity_moderate',␣
 ↪'intensity_hard_and_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 and very hard'),␣
 ↪'intensity_hard_and_very_hard']    = 1

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

[13]:
```
dataset.head()
```

```
[13]:                      sum_mag_of_acc          speed  mean_met  weight_kg  \
      2019-09-12 10:38:00      977.310095   19984.040802  1.658801       64.0
      2019-09-12 10:39:00     1429.448879  101474.124770  1.505580       64.0
      2019-09-12 10:40:00     1419.886845  187111.197538  1.548363       64.0
      2019-09-12 10:41:00     1419.545634  271905.246767  1.508185       64.0
      2019-09-12 10:42:00     1426.501272  357912.594762  1.438616       64.0

                           length_m        bmi intensity activity  intensity_light  \
      2019-09-12 10:38:00     1.749  20.921863     Light   zitten                1
      2019-09-12 10:39:00     1.749  20.921863     Light   zitten                1
      2019-09-12 10:40:00     1.749  20.921863     Light   zitten                1
      2019-09-12 10:41:00     1.749  20.921863     Light   zitten                1
      2019-09-12 10:42:00     1.749  20.921863     Light   zitten                1

                           intensity_moderate  intensity_hard_and_very_hard
      2019-09-12 10:38:00                   0                             0
      2019-09-12 10:39:00                   0                             0
      2019-09-12 10:40:00                   0                             0
      2019-09-12 10:41:00                   0                             0
      2019-09-12 10:42:00                   0                             0
```

2.3 Splitting dataset

```
[14]: x = dataset[['sum_mag_of_acc', 'bmi', 'weight_kg', 'length_m']]
      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)
```

```
[14]: 696
```

2.4 Training dataset

```
[15]: rfc = RandomForestClassifier(n_estimators=7,random_state=0)
      rfc.fit(train_x, train_y)
```

```
[15]: RandomForestClassifier(n_estimators=7, random_state=0)
```

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

2.5 Result analysis

2.5.1 Validation

[17]:
```
accuracy_score(valid_y, prediction_y, normalize=True)
```

[17]:
```
0.7142857142857143
```

[18]:
```
print(classification_report(valid_y, prediction_y,␣
 ↪target_names=intensity_columns, zero_division=0))
```

```
                              precision    recall  f1-score   support

             intensity_light       0.72      0.78      0.75        60
          intensity_moderate       0.70      0.68      0.69        56
intensity_hard_and_very_hard       0.79      0.62      0.70        24

                   micro avg       0.72      0.71      0.72       140
                   macro avg       0.74      0.70      0.71       140
                weighted avg       0.73      0.71      0.72       140
                 samples avg       0.71      0.71      0.71       140
```

2.5.2 Test

[19]:
```
#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 and very hard'),␣
 ↪'intensity_hard_and_very_hard']    = 1

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

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

[20]:
```
#test_prediction_y = rfc.predict(x)
```

[21]:
```
#accuracy_score(y, test_prediction_y, normalize=True)
```

[22]:

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

2.6 Tune hyperparameteres

[23]:

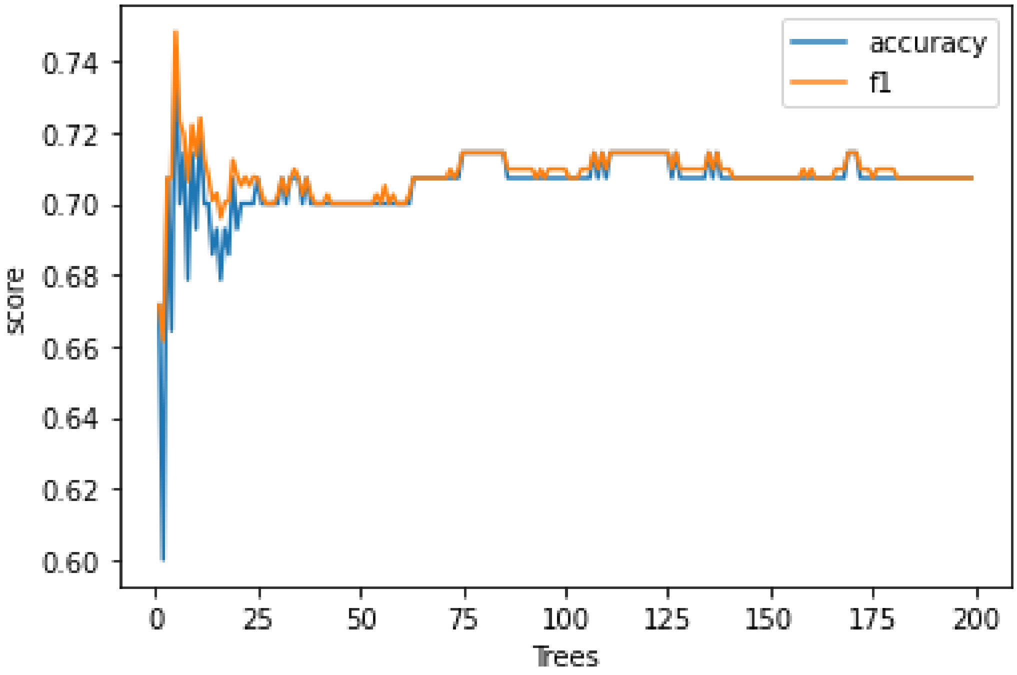
```
#### Quick analysis
accuracy_scores = []
f1_scores = []

n_estimator_numbers = range(1,200)

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' ))
```

[24]:

```
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.legend()
```

[24]:

```
<matplotlib.legend.Legend at 0x7fadf2512978>
```



[25]:
```
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:  5
f1:  5
```

[ ]:

[ ]: