behacom_ds_analysis

July 7, 2021

0.1 Adding necessary imports

"Plotly" library is used for visualization.

[1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
from datetime import datetime
import plotly
import plotly.express as px
import plotly.graph_objs as go
from sklearn.cluster import KMeans
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

0.2 Method for reading the dataset of i-th User

Read CSVs with memory optimization If the CSV files are readed without any optimization, columns are taken as float64, so the CSV occupies a lot of memory. In order to reduce memory, this procedure changes dtype parameter and optmizes it. For that purpose, a chunk is read and then types are optimized.

[2]:
```
columns = ['timestamp', 'keystroke_counter', 'current_app',
               'current_app_foreground_time', 'mouse_average_movement_duration',
               'changes_between_apps', 'click_speed_average_0',␣
 ↪'click_speed_average_1',
               'click_speed_average_2', 'click_speed_average_3']

def read_user_data(userno):
    filename=f"data/User{userno}/User{userno}_behacom.csv"
    print(f'[INFO] reading file <{filename}>...')

    data_user=pd.read_csv(filename,␣
 ↪encoding='latin-1',chunksize=50,usecols=columns)

    a = next(data_user)
    dtypes_col = a.dtypes.index
    dtypes_type = [i.name for i in a.dtypes.values]
```

```
    column_types = dict(zip(dtypes_col, dtypes_type))

    for k,v in column_types.items():
        if k == 'timestamp':
            column_types[k] = 'float64'
        elif ('average' in k):
            column_types[k] = 'float32'
        elif ('stddev' in k):
            column_types[k] = 'float32'
        elif v == 'float64':
            column_types[k] = 'float32'
        elif v == 'int64':
            if (k.startswith('press') or ('counter' in k) or ('usage' in k)):
                column_types[k] = 'int8'
            else:
                column_types[k] = 'int32'

    data_user=pd.read_csv(filename,␣
↪encoding='latin-1',dtype=column_types,usecols=columns)
    #print(data_user.shape)
    return data_user
```

0.3 Task 1: Read dataset

Reading all dataset iteratively and saving in a list: `df_users`, an indication of file reading is printed during the read process. Note that, the `timestamp` feature has been converted to `datetime` and stored in the dataframe.

[3]:
```
total_users = 12
df_users = []
basic_info = { 'name':[], 'length':[]}
for i in range(total_users):
    df = read_user_data(i)
    df_users.append(df)
    basic_info['name'].append(f'User {i}')
    basic_info['length'].append(df.shape[0])
    df_users[i]['date_time'] = df_users[i][:]['timestamp'].
↪astype('datetime64[ms]')
```

```
[INFO] reading file <data/User0/User0_behacom.csv>…
[INFO] reading file <data/User1/User1_behacom.csv>…
[INFO] reading file <data/User2/User2_behacom.csv>…
[INFO] reading file <data/User3/User3_behacom.csv>…
[INFO] reading file <data/User4/User4_behacom.csv>…
[INFO] reading file <data/User5/User5_behacom.csv>…
[INFO] reading file <data/User6/User6_behacom.csv>…
[INFO] reading file <data/User7/User7_behacom.csv>…
[INFO] reading file <data/User8/User8_behacom.csv>…
```

```
[INFO] reading file <data/User9/User9_behacom.csv>…
[INFO] reading file <data/User10/User10_behacom.csv>…
[INFO] reading file <data/User11/User11_behacom.csv>…
```

0.4 Task 2 & 3: Dataset overview

In the first figure we see the volume of data/input per user and observe that User 7 contains most and User 2 contains least amount of data.

[4]:
```
basic_info = pd.DataFrame(basic_info)
fig = px.bar(basic_info, x='name', y='length', color='name',␣
 ↪title='Distribution of data per user')
fig.show()
```

In the following figure we plot the keystroke trend per user over time. First the `date_time` feature is grouped with daily frequency then `keystroke_counter` is summed up. Finally the output is plotted in the figure. Note that, User x's *(x:0 to 11)* input start and end date are also informed in the figure's legend.

[5]:
```
fig = go.Figure()
for i in range(total_users):
    df_grp_keystroke = df_users[i].groupby(pd.
 ↪Grouper(key='date_time',freq='D')).agg({"keystroke_counter": "sum"}).
 ↪reset_index().sort_values(by='date_time')
    startDate = df_grp_keystroke['date_time'].dt.date.min()
    endDate = df_grp_keystroke['date_time'].dt.date.max()
    fig.add_trace(go.Scatter(x=df_grp_keystroke['date_time'],␣
 ↪y=df_grp_keystroke['keystroke_counter'], name=f'User {i}, start:␣
 ↪{startDate}, end: {endDate}'))

fig.update_layout(
    title='Keystroke trend per user over time'
)
fig.show()
```

Here the `date_time` feature is grouped again with daily frequency but now the size of the daily volume of input is taken into account.

[6]:
```
fig = go.Figure()
for i in range(total_users):
    df_grp_input_trend = df_users[i].groupby(pd.
 ↪Grouper(key='date_time',freq='D')).size().to_frame(name='counts').
 ↪reset_index().sort_values(by='date_time')
    fig.add_trace(go.Scatter(x=df_grp_input_trend['date_time'],␣
 ↪y=df_grp_input_trend['counts'], name=f'User {i}'))

fig.update_layout(
    title='Daily input distribution per user'
```

```
)
fig.show()
```

0.5 Task 4 & 5

Considering the inputs of User 0 for this task. Note that the hour of the day feature has been added in the column `hour_of_day` from `date_time` feature.

User activity: to define user activity we consider the sum of following features:- * `keystroke_counter` *(total number of keystrokes generated by the user during the time window)* * `mouse_average_movement_duration` *(average duration of the mouse movements in milliseconds)* * `click_speed_average_N` *(set of features represents the average time elapsed to complete a click, N represents each one of the mouse buttons, 0 is left button click, 1 is right button click, 2 is left button double click and 3 is middle button click.).* * `changes_between_apps` *(number of changes between different foreground applications during the time window)*

[7]:
```
df_user0 = df_users[0].sort_values(by='date_time')
df_user0['hour_of_day'] = df_user0['date_time'].dt.hour
df_user0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6059 entries, 0 to 6058
Data columns (total 12 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   timestamp                        6059 non-null   float64
 1   keystroke_counter                6059 non-null   int8
 2   click_speed_average_0            6059 non-null   float32
 3   click_speed_average_1            6059 non-null   float32
 4   click_speed_average_2            6059 non-null   float32
 5   click_speed_average_3            6059 non-null   float32
 6   mouse_average_movement_duration  6059 non-null   float32
 7   current_app                      6059 non-null   object
 8   changes_between_apps             6059 non-null   int32
 9   current_app_foreground_time      6059 non-null   float32
 10  date_time                        6059 non-null   datetime64[ns]
 11  hour_of_day                      6059 non-null   int64
dtypes: datetime64[ns](1), float32(6), float64(1), int32(1), int64(1), int8(1),
object(1)
memory usage: 408.3+ KB
```

0.5.1 Activity of user

We would like to learn how active an user over the course of the day. Therefore we calculate `user_activity` from `hour_of_day`.

[8]:
```
```

```
df_grp_user0 = df_user0.groupby(pd.Grouper(key='hour_of_day')).
 ↪agg(total_keystroke_counter=('keystroke_counter', 'sum'),␣
 ↪total_mouse_average_movement_duration=('mouse_average_movement_duration',␣
 ↪'sum'), total_changes_between_apps=('changes_between_apps', 'sum'),␣
 ↪total_click_speed_average_0=('click_speed_average_0', 'sum'),␣
 ↪total_click_speed_average_1=('click_speed_average_1', 'sum'),␣
 ↪total_click_speed_average_2=('click_speed_average_2', 'sum'),␣
 ↪total_click_speed_average_3=('click_speed_average_3', 'sum')).reset_index()
df_grp_user0['user_activity'] = df_grp_user0['total_keystroke_counter'] +␣
 ↪df_grp_user0['total_mouse_average_movement_duration'] +␣
 ↪df_grp_user0['total_changes_between_apps'] +␣
 ↪df_grp_user0['total_click_speed_average_0'] +␣
 ↪df_grp_user0['total_click_speed_average_1'] +␣
 ↪df_grp_user0['total_click_speed_average_2'] +␣
 ↪df_grp_user0['total_click_speed_average_3']
```

[9]:
```
df_grp_user0.head()
```

[9]:
```
   hour_of_day  total_keystroke_counter  \
0           13                   2634.0
1           14                   1669.0
2           15                   3375.0
3           16                   4743.0
4           18                   8835.0

   total_mouse_average_movement_duration  total_changes_between_apps  \
0                           47771.558594                          62
1                           12679.520508                          78
2                           27691.400391                         149
3                           40354.621094                         163
4                           96267.023438                          59

   total_click_speed_average_0  total_click_speed_average_1  \
0                 5.251769e+11                  2850.580078
1                 1.180759e+12                   296.500000
2                 5.995796e+11                  1925.420044
3                 3.970916e+12                  1522.010010
4                 1.161061e+12                  2672.500000

   total_click_speed_average_2  total_click_speed_average_3  user_activity
0                  7001.160156                          0.0   5.251770e+11
1                  2304.449951                          0.0   1.180760e+12
2                  7971.709961                          0.0   5.995796e+11
3                  8019.540039                          0.0   3.970916e+12
4                 11562.750000                          0.0   1.161061e+12
```

0.5.2 Activeness of User 0 over the day

In this figure, the activeness of User 0 over the day has been plot.

From this histogram user's sleep activity can be distinguished.

We observe that there is no or rare activity from 00:00 untill 08:00, therefore it might be User 0's sleep time.

Also the user is active after waking up highly active during midday, before dayend and at night until midnight.

[10]:
```
fig = px.histogram(df_grp_user0, x="hour_of_day", y='user_activity', nbins=12,␣
 ↪histnorm='probability', title='Activeness of User 0 over hour of day')
fig.show()
```

[11]:
```
fig = px.scatter(df_grp_user0, x="hour_of_day", y='user_activity')
fig.show()
```

0.5.3 Classification of activeness of User 0

Now we are interested in dividing these activeness into states like: fully-active (having a considerable number of interactions via mouse, keyboard), middle, and passive (very few interactions). Because our data has no level, K-means Clustering approach is used to cluster the data.

Step 1: Prepare the dataset

[12]:
```
df_xy = pd.DataFrame(df_grp_user0, columns=['hour_of_day', 'user_activity'])
df_xy.columns = ['x', 'y']
df_xy['state'] = '' # adding column for future level
df_xy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23 entries, 0 to 22
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   x       23 non-null     int64
 1   y       23 non-null     float64
 2   state   23 non-null     object
dtypes: float64(1), int64(1), object(1)
memory usage: 680.0+ bytes
```

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

[13]:

	x	y	state
0	13	5.251770e+11	
1	14	1.180760e+12	
2	15	5.995796e+11	
3	16	3.970916e+12	

```
4  18  1.161061e+12
```

Step 2: Data pre-process/scaling

[14]:
```
# Pre-processing data with MinMaxScalar,
df_xy[['y']] = StandardScaler().fit_transform(df_xy[['y']])
# df_xy[['y']] = MinMaxScaler().fit_transform(df_xy[['y']])
# df_xy['y'] = np.log10(df_xy['y'])
# df_xy['y'].replace([np.inf, -np.inf], np.nan, inplace=True)
# df_xy['y'].fillna(0, inplace=True)
df_xy.head()
```

[14]:
```
    x         y state
0  13 -0.542091
1  14 -0.100884
2  15 -0.492018
3  16  1.776889
4  18 -0.114142
```

Step 3: Process the data in K-Means method

[15]:
```
kmeans = KMeans(n_clusters=3).fit(pd.DataFrame(df_xy, columns = ['x', 'y']))
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[ 3.14285714 -0.83495151]
 [19.5         0.57314107]
 [11.5         0.15744151]]
```

Step 4: Prepare the categories based on the centroids

[16]:
```
# save categories in list based on the centroids most/least values
states=['', '', '']
cen = centroids[np.ix_([0,1,2],[1])]
states[np.argmax(cen)] = 'fully-active'
states[np.argmin(cen)] = 'passive'
states[[i for i in range(len(states)) if states[i] == '' ][0]] = 'middle'
```

Step 5: Cluster the dataset using centroids and show output

[17]:
```
for idx, row in df_xy.iterrows():
    diff_y = [row.y-cen[1] for cen in centroids]
    diff_y = np.abs(diff_y)
    idxmin = np.argmin(diff_y)

    df_row = row.to_frame().transpose()
    df_xy.at[idx,'state'] = states[idxmin]
```

```
fig = px.scatter(df_xy, x='x', y='y', color='state', title='Classification of␣
↪activeness of User 0')
fig.show()
```

So we have clustered the data into three categories as plotted in above figure. High activity, middle activity and bare activity are shown in green, red and blue colors respectively.

As the levels are now know, we can use regression to predict future activeness of the user.

0.6 Task 6: Probability of switching among states

Calculating the probability of switching between states of User 0. We will use the outcome and levels in previous section for the calculation.

There are 24 states of User 0 after groupped by `hour_of_day`. * There are in total 6 `fully-active` states. * There is one case that after being `fully-active` the user is in a `middle` activity state. * Among the rest 5 states, there are 3 cases where after `fully-active` state, the user is in `passive` state. * Therefore after a `fully-active` state, there is $3 \div 6 = 0.5$ probability to move into `passive` state and $1 \div 6 = 0.17$ probability to move into `middle` activity state.

Following table shows the calculation of switching probability from the states in the first column to the states in first row.

›	fully-active	middle	passive
fully-active	$2 \div 6 = 0.33$	$1 \div 6 = 0.17$	$3 \div 6 = 0.50$
middle	$1 \div 5 = 0.20$	$2 \div 5 = 0.40$	$2 \div 5 = 0.40$
passive	$3 \div 12 = 0.25$	$2 \div 12 = 0.17$	$7 \div 12 = 0.58$

0.7 Task 7: Insight about user's behaviors

0.7.1 Most active/used app

One interesting fact would be to learn the most used application by an user. Here we group the dataset by `current_app` then sum `current_app_foreground_time`, from these the app with maximum foregound time is stored for each user. Finally the info is plotted into the first figure. From the second figure we learn the most used app of all time.

[18]:
```
columns = ['user', 'user_no', 'current_app', 'current_app_foreground_time']
data = []
for i in range(total_users):
    df_current_app = df_users[i].groupby(pd.Grouper(key='current_app')).
↪agg({"current_app_foreground_time": "sum"}).reset_index()
    row_current_app = df_current_app.
↪loc[df_current_app['current_app_foreground_time'].idxmax()]
    data.append([f'User {i}', i, row_current_app.current_app, row_current_app.
↪current_app_foreground_time])

# most used app by user
df_current_app = pd.DataFrame(data=data, columns=columns)
```

```
# sort by user_no
# df_current_app.sort_values(by='user_no', inplace=True)
df_current_app['current_app'] = df_current_app['current_app'].str.replace('.
↪exe', '', regex=False)
fig = px.bar(df_current_app, x='user', y='current_app_foreground_time',␣
↪color='current_app', title='Most active/used app per User')
fig.update_layout(xaxis={'categoryorder':'category ascending'})
fig.show()
# most used app count
df_current_app = df_current_app.groupby(pd.Grouper(key='current_app')).size().
↪to_frame(name='counts').reset_index()
fig = px.bar(df_current_app, x='current_app', y='counts', color='current_app',␣
↪title='Most active/used app')
fig.show()
```

0.7.2 Activity of all users per date

From the following figure we can observe activity of all users per date.

Notice that the users are barely active/inactive on weekends and Spanish public holidays. In other words, less users are activue on weekends/holidays.

For example there is no activity on 06.12.2019 celebrated as Constitution Day and 25.12.2019 celebrated as Christmas Day etc.

[19]:
```
list_df = []
for i in range(total_users):
    df_user_activity = df_users[i].groupby(pd.
↪Grouper(key='date_time',freq='D')).
↪agg(total_keystroke_counter=('keystroke_counter', 'sum'),␣
↪total_mouse_average_movement_duration=('mouse_average_movement_duration',␣
↪'sum'), total_changes_between_apps=('changes_between_apps', 'sum'),␣
↪total_click_speed_average_0=('click_speed_average_0', 'sum'),␣
↪total_click_speed_average_1=('click_speed_average_1', 'sum'),␣
↪total_click_speed_average_2=('click_speed_average_2', 'sum'),␣
↪total_click_speed_average_3=('click_speed_average_3', 'sum')).reset_index()
    df_user_activity['user_activity'] =␣
↪df_user_activity['total_keystroke_counter'] +␣
↪df_user_activity['total_mouse_average_movement_duration'] +␣
↪df_user_activity['total_changes_between_apps'] +␣
↪df_user_activity['total_click_speed_average_0'] +␣
↪df_grp_user0['total_click_speed_average_1'] +␣
↪df_user_activity['total_click_speed_average_2'] +␣
↪df_user_activity['total_click_speed_average_3']
    list_df.append(df_user_activity)
# most activity of all users per day
df_from_list = pd.concat(list_df)
```

```
df_from_list = df_from_list.groupby(pd.Grouper(key='date_time',freq='D')).
↪agg(total_user_activity=('user_activity', 'sum')).reset_index()

fig = go.Figure()
fig.add_trace(go.Bar(x=df_from_list['date_time'],␣
↪y=df_from_list['total_user_activity']))
fig.update_layout(title='Activity of all users per day')
fig.show()
```

[20]:
```
list_df = []
for i in range(total_users):
    df_user_activity = pd.DataFrame(df_users[i], columns=['date_time',␣
↪'keystroke_counter', 'mouse_average_movement_duration',␣
↪'changes_between_apps', 'click_speed_average_0', 'click_speed_average_1',␣
↪'click_speed_average_2', 'click_speed_average_3'])
    df_user_activity['day_name'] = df_user_activity['date_time'].dt.day_name()
    df_user_activity['name'] = f'User {i}'
    df_user_activity['user_activity'] = df_user_activity['keystroke_counter'] +␣
↪df_user_activity['mouse_average_movement_duration'] +␣
↪df_user_activity['changes_between_apps'] +␣
↪df_user_activity['click_speed_average_0'] +␣
↪df_user_activity['click_speed_average_1'] +␣
↪df_user_activity['click_speed_average_2'] +␣
↪df_user_activity['click_speed_average_3']
    list_df.append(df_user_activity)
# most activity of all users per day of week
df_from_list = pd.concat(list_df)
fig = px.bar(df_from_list, x='day_name', y='user_activity', color='name',␣
↪title='Activity of all users per day of week')
fig.show()
```

Another interesting fact is that, a typo in dataset describing paper was found regarding a feature: `click_speed_aveage_N` on page 6 in the first row of table 4, it should be `click_speed_average_N`.

0.8 Remarks

- The aggregated dataset is huge to be saved in memory. Hence to utilize limited memory only selected columns are loaded which have been used later in the calculation.
- If we want to use prediction for leveling future user activity from the learning of task 5:
- We need to apply standardization in the very beginning before defining user activity.
- The study in task 5 was performed for User 0. To get more general outcome, all user's activity must be considered.

[ ]:
```

```