Standard deviation feature rolling window

February 5, 2021

[1]:
```
#edit notebook width
from IPython.display import display, HTML

display(HTML(data="""
<style>
    div#notebook-container    { width: 95%; }
    div#menubar-container     { width: 65%; }
    div#maintoolbar-container { width: 99%; }
</style>
"""))
```

```
<IPython.core.display.HTML object>
```

[2]:
```
#navigeren naar de juiste directory
import os
os.chdir("/data/momo/tooling")
#controleren of er genavigeerd is naar de juiste directory
os.getcwd()
```

[2]:
```
'/data/momo/tooling'
```

[3]:
```
# Selecte a list from the CSV files
lijst = !ls /data/momo/Trainingset/week4
lijst2 = !ls /data/momo/week_data/00090042/max_time_gap_less_than_10s
```

[4]:
```
#importing required libraries
from _indexers import _generate_headers
from importing_csv import load_csv
from preprocessing import PEdata
from plotting import plot_features
import matplotlib.dates as mdates
import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as dt
import matplotlib.ticker as ticker
from tqdm import tqdm
```

```
#Importing the required libraries
from sklearn.metrics import accuracy_score, recall_score, precision_score
from mpl_toolkits.axes_grid1 import make_axes_locatable
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from _indexers import _generate_headers
from importing_csv import load_csv
from plotting import plot_features
from sklearn.svm import LinearSVC
from preprocessing import PEdata
import matplotlib.pyplot as plt
from tqdm import tqdm
import seaborn as sn
import pandas as pd
import numpy as np
import statistics
import pickle
import glob
```

[5]:
```
# Alle test files
file_name1 = "/data/momo/Trainingset/week4/00090051_20200831T2159_20200831T2329.
 ↪csv"
file_name2 = "/data/momo/Trainingset/week4/00090176_20200901T0458_20200901T0628.
 ↪csv"
file_name3 = "/data/momo/Trainingset/week4/00090174_20200829T0528_20200829T0658.
 ↪csv"
file_name4 = "/data/momo/Trainingset/week4/00090166_20200902T2242_20200903T0012.
 ↪csv"
file_name5 = "/data/momo/Trainingset/week4/00090172_20200902T0653_20200902T0823.
 ↪csv"
```

[6]:
```
# Read data function that reads and slices csv-files based on version variable.
# Version 1: Data sliced based on T(moment of leaving bed)
# version 2: Data sliced based on the bed status to add in-bed situations.
# version 3: Data will not be sliced


def lees_data(file_name, version):
    cont_df, _ , PE_df, _ =␣
 ↪load_csv(file_name,events=False,electric=True,reduce_rate=1,is_big=False)

    # Reshape 10Hz PE signal of 72 columbs to 120 Hz in 6 columbs
    PE_df = PEdata(PE_df)
```

```
    # Changing DataFrame index to timedata
    PE_df['datetime'] = pd.to_datetime(PE_df["time"].values, unit="ms",␣
↪utc=True)
    PE_df = PE_df.set_index('datetime')


    cont_df['datetime'] = pd.to_datetime(cont_df["time"].values, unit="ms",␣
↪utc=True)
    cont_df = cont_df.set_index('datetime')

    # plotting the bed_status, force distribution (FSR_values), vibrations␣
↪(PE_values) and the bedrest angle
    plot_list = {'bed_status': cont_df['bed_status'].astype(int),
                 'FSR_values': cont_df[_generate_headers('fsr')],
                 'PE_values': PE_df[["PE" + str(i) for i in range(1, 7)]],}


    #plot_pe de PE_values column selecteren
    plot_pe = pd.DataFrame(plot_list['PE_values'])


    #select the FSR_values column and name the result: "plot_fsr"
    plot_fsr = pd.DataFrame(plot_list['FSR_values'])

    #select the bed_status column and name the result: "plot_bed_status"
    plot_bed_status = pd.DataFrame(plot_list['bed_status'])


    if version == 1:
        #slice the column when the bed-status == 0 and name the result:␣
↪"value_0"
        value_0 = cont_df["bed_status"] == 0

        #select all columns of cont_df when its bed-status == 0 and name the␣
↪result:"Out_of_bed"
        Out_of_bed = cont_df[value_0]

        #slice the data 1 minute after the moment of leaving bed and name the␣
↪result:"end_time"
        end_time = Out_of_bed.index + pd.Timedelta(minutes=1)

        #slice the data 10 minutes before the moment of leaving bed and name␣
↪the result:"start_time"
        start_time = (end_time - pd.Timedelta(minutes=10))
```

```
    elif version == 2:
        #slice the column when the bed-status == 2 and name the result:␣
↪"value_2"
        value_2 = cont_df["bed_status"] == 2

        #select all columns of cont_df when its bed-status == 2 and name the␣
↪result:"In_bed"
        In_bed = cont_df[value_2]

        #slice the data 1 minute after the start of the csv_file and name the␣
↪result:"end_time"
        end_time = In_bed.index + pd.Timedelta(minutes=10)

        #slice the data from the start of the csv_file and name the result:
↪"start_time"
        start_time = In_bed.index




    #mask_fsr Een mask die slicet de  data op basis van de index Datetime met␣
↪gebruik van de start en end time variablen
    mask_fsr = (plot_fsr.index > start_time[0]) & (plot_fsr.index < end_time[0])

    #mask_pe Een mask die slicet de  data op basis van de index Datetime met␣
↪gebruik van de start en end time variablen
    mask_pe = (plot_pe.index > start_time[0]) & (plot_pe.index < end_time[0])

    #bed_status_df selecteer de bed_status datas op basis van .loc methode met␣
↪gebruik van mask_fsr
    bed_status_df = plot_bed_status.loc[mask_fsr]


    filter_fsr_df = plot_fsr.loc[mask_fsr]

    filter_pe_df = plot_pe.loc[mask_pe]

#     pe_columns = filter_pe_df.columns

#     filter_fsr_df[pe_columns] = filter_pe_df[pe_columns]

    filter_fsr_df['bed_status'] = bed_status_df['bed_status']
```

```
    filter_fsr_df['csv_file'] = pd.Series(dtype='str')

    filter_fsr_df['csv_file'] = file_name

    return filter_fsr_df
```

[7]:
```
#Read csv files
uit_bed1 = lees_data(file_name1,1)
uit_bed2 = lees_data(file_name2,1)
uit_bed3 = lees_data(file_name3,1)
uit_bed4 = lees_data(file_name4,1)
uit_bed5 = lees_data(file_name5,1)


in_bed1 = lees_data(file_name1,2)
in_bed2 = lees_data(file_name2,2)
in_bed3 = lees_data(file_name3,2)
in_bed4 = lees_data(file_name4,2)
in_bed5 = lees_data(file_name5,2)

uit_bed1.name = 'uit_bed1'
uit_bed2.name = 'uit_bed2'
uit_bed3.name = 'uit_bed3'
uit_bed4.name = 'uit_bed4'
uit_bed5.name = 'uit_bed5'
in_bed1.name = 'in_bed1'
in_bed2.name = 'in_bed2'
in_bed3.name = 'in_bed3'
in_bed4.name = 'in_bed4'
in_bed5.name = 'in_bed5'
```

[8]:
```
DF_time1 = uit_bed1[uit_bed1['bed_status']==0].index[0]
DF_time2 = uit_bed2[uit_bed2['bed_status']==0].index[0]
DF_time3 = uit_bed3[uit_bed3['bed_status']==0].index[0]
DF_time4 = uit_bed4[uit_bed4['bed_status']==0].index[0]
DF_time5 = uit_bed5[uit_bed5['bed_status']==0].index[0]
print(DF_time1)
print(DF_time2)
print(DF_time3)
print(DF_time4)
print(DF_time5)
```

```
2020-08-31 22:44:14.485000+00:00
2020-09-01 05:36:09.183000+00:00
2020-08-29 06:13:36.671000+00:00
2020-09-02 23:28:21.176000+00:00
2020-09-02 07:42:11.577000+00:00
```

[9]:
```
uit_bed1
```

[9]:

datetime	sp_resistive_0	sp_resistive_1	\
2020-08-31 22:35:14.568000+00:00	383	227	
2020-08-31 22:35:14.675000+00:00	383	227	
2020-08-31 22:35:14.766000+00:00	380	227	
2020-08-31 22:35:14.874000+00:00	383	227	
2020-08-31 22:35:14.963000+00:00	383	227	
…	…	…	
2020-08-31 22:45:14.020000+00:00	356	83	
2020-08-31 22:45:14.131000+00:00	357	83	
2020-08-31 22:45:14.218000+00:00	356	81	
2020-08-31 22:45:14.328000+00:00	356	82	
2020-08-31 22:45:14.415000+00:00	357	81	

datetime	sp_resistive_2	sp_resistive_3	\
2020-08-31 22:35:14.568000+00:00	457	571	
2020-08-31 22:35:14.675000+00:00	467	572	
2020-08-31 22:35:14.766000+00:00	462	568	
2020-08-31 22:35:14.874000+00:00	462	572	
2020-08-31 22:35:14.963000+00:00	465	571	
…	…	…	
2020-08-31 22:45:14.020000+00:00	217	267	
2020-08-31 22:45:14.131000+00:00	218	268	
2020-08-31 22:45:14.218000+00:00	211	268	
2020-08-31 22:45:14.328000+00:00	218	267	
2020-08-31 22:45:14.415000+00:00	217	267	

datetime	sp_resistive_4	sp_resistive_5	\
2020-08-31 22:35:14.568000+00:00	440	641	
2020-08-31 22:35:14.675000+00:00	442	640	
2020-08-31 22:35:14.766000+00:00	442	640	
2020-08-31 22:35:14.874000+00:00	437	640	
2020-08-31 22:35:14.963000+00:00	439	640	
…	…	…	
2020-08-31 22:45:14.020000+00:00	215	346	
2020-08-31 22:45:14.131000+00:00	216	346	
2020-08-31 22:45:14.218000+00:00	215	345	
2020-08-31 22:45:14.328000+00:00	214	345	
2020-08-31 22:45:14.415000+00:00	216	345	

datetime	sp_resistive_6	sp_resistive_7	bed_status	\
2020-08-31 22:35:14.568000+00:00	687	398	2	

```
2020-08-31 22:35:14.675000+00:00              686              399            2
2020-08-31 22:35:14.766000+00:00              683              399            2
2020-08-31 22:35:14.874000+00:00              686              399            2
2020-08-31 22:35:14.963000+00:00              686              399            2
…                                             …                …              …
2020-08-31 22:45:14.020000+00:00              384              273            0
2020-08-31 22:45:14.131000+00:00              384              273            0
2020-08-31 22:45:14.218000+00:00              384              275            0
2020-08-31 22:45:14.328000+00:00              384              273            0
2020-08-31 22:45:14.415000+00:00              384              273            0

csv_file
datetime
2020-08-31 22:35:14.568000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:35:14.675000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:35:14.766000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:35:14.874000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:35:14.963000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
…
…
2020-08-31 22:45:14.020000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:45:14.131000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:45:14.218000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:45:14.328000+00:00
/data/momo/Trainingset/week4/00090051_20200831…
2020-08-31 22:45:14.415000+00:00
/data/momo/Trainingset/week4/00090051_20200831…

[5439 rows x 10 columns]
```

[10]:
```python
#Feature standard deviation rolling window
def Feature(data):


    if data.name in ["uit_bed1","uit_bed2","uit_bed3","uit_bed4","uit_bed5"]:
        DF_time = data[data['bed_status']==0].index[0]
    else:
        DF_time = 0
```

```
    # define the columns for the output dataframes
    lst_std_feature =␣
↪['std_feature_0','std_feature_1','std_feature_2','std_feature_3','std_feature_4','std_featu
    lst_peaks =␣
↪['Peak_fsr_0','Peak_fsr_1','Peak_fsr_2','Peak_fsr_3','Peak_fsr_4','Peak_fsr_5','Peak_fsr_6'
    lst_mean =␣
↪['Mean_fsr_0','Mean_fsr_1','Mean_fsr_2','Mean_fsr_3','Mean_fsr_4','Mean_fsr_5''Mean_fsr_6',
    lst_std =␣
↪['Std_fsr_0','Std_fsr_1','Std_fsr_2','Std_fsr_3','Std_fsr_4','Std_fsr_5','Std_fsr_6','Std_f
    lst_fsr = ['Fsr_0','Fsr_1','Fsr_2','Fsr_3','Fsr_4','Fsr_5','Fsr_6','Fsr_7']
    columns = lst_fsr +lst_std_feature + lst_mean + lst_std + lst_peaks

    # create the output dataframes
    DF_plot= pd.DataFrame(columns=columns,index=data.index)

    #add bestatus
    DF_plot['bed_status'] = data['bed_status']

    #Loop through the data
    for i in tqdm(range(len(data))):
        #copy the value of the FSR sensors and store them in DF_plot
        DF_plot['Fsr_0'][i] = data['sp_resistive_0'][i]
        DF_plot['Fsr_1'][i] = data['sp_resistive_1'][i]
        DF_plot['Fsr_2'][i] = data['sp_resistive_2'][i]
        DF_plot['Fsr_3'][i] = data['sp_resistive_3'][i]
        DF_plot['Fsr_4'][i] = data['sp_resistive_4'][i]
        DF_plot['Fsr_5'][i] = data['sp_resistive_5'][i]
        DF_plot['Fsr_6'][i] = data['sp_resistive_6'][i]
        DF_plot['Fsr_7'][i] = data['sp_resistive_7'][i]

        #using a rolling window of length 2 look back and calculate the mean␣
↪per fsr sensor
        DF_plot['Mean_fsr_0'] = data['sp_resistive_0'].rolling(3).mean()
        DF_plot['Mean_fsr_1'] = data['sp_resistive_1'].rolling(3).mean()
        DF_plot['Mean_fsr_2'] = data['sp_resistive_2'].rolling(3).mean()
        DF_plot['Mean_fsr_3'] = data['sp_resistive_3'].rolling(3).mean()
        DF_plot['Mean_fsr_4'] = data['sp_resistive_4'].rolling(3).mean()
        DF_plot['Mean_fsr_5'] = data['sp_resistive_5'].rolling(3).mean()
        DF_plot['Mean_fsr_6'] = data['sp_resistive_6'].rolling(3).mean()
        DF_plot['Mean_fsr_7'] = data['sp_resistive_7'].rolling(3).mean()

        #using a rolling window of length 2 look back and sum the fsr values␣
↪that are the standard*3 per fsr sensor
        DF_plot['Std_fsr_0'] = data['sp_resistive_0'].rolling(3).std()
        DF_plot['Std_fsr_1'] = data['sp_resistive_1'].rolling(3).std()
        DF_plot['Std_fsr_2'] = data['sp_resistive_2'].rolling(3).std()
```

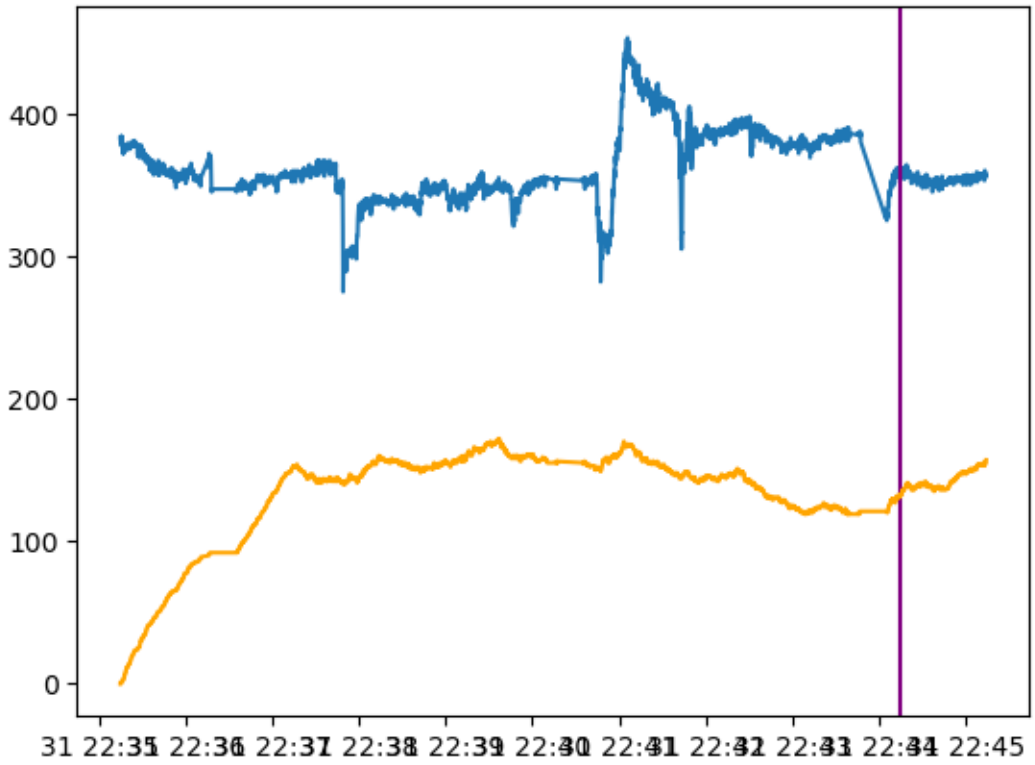
```
        DF_plot['Std_fsr_3'] = data['sp_resistive_3'].rolling(3).std()
        DF_plot['Std_fsr_4'] = data['sp_resistive_4'].rolling(3).std()
        DF_plot['Std_fsr_5'] = data['sp_resistive_5'].rolling(3).std()
        DF_plot['Std_fsr_6'] = data['sp_resistive_6'].rolling(3).std()
        DF_plot['Std_fsr_7'] = data['sp_resistive_7'].rolling(3).std()


        # store peaks per FSR sensor in the 'Peaks' column of DF_Plot
        if DF_plot['Fsr_0'][i] > (DF_plot['Mean_fsr_0'][i] +␣
↪DF_plot['Std_fsr_0'][i]):
            DF_plot['Peak_fsr_0'][i] = DF_plot['Fsr_0'][i]

        if DF_plot['Fsr_1'][i] > (DF_plot['Mean_fsr_1'][i] +␣
↪DF_plot['Std_fsr_1'][i]):
            DF_plot['Peak_fsr_1'][i] = DF_plot['Fsr_1'][i]

        if DF_plot['Fsr_2'][i] > (DF_plot['Mean_fsr_2'][i] +␣
↪DF_plot['Std_fsr_2'][i]):
            DF_plot['Peak_fsr_2'][i] = DF_plot['Fsr_2'][i]

        if DF_plot['Fsr_3'][i] > (DF_plot['Mean_fsr_3'][i] +␣
↪DF_plot['Std_fsr_3'][i]):
            DF_plot['Peak_fsr_3'][i] = DF_plot['Fsr_3'][i]

        if DF_plot['Fsr_4'][i] > (DF_plot['Mean_fsr_4'][i] +␣
↪DF_plot['Std_fsr_4'][i]):
            DF_plot['Peak_fsr_4'][i] = DF_plot['Fsr_4'][i]

        if DF_plot['Fsr_5'][i] > (DF_plot['Mean_fsr_5'][i] +␣
↪DF_plot['Std_fsr_5'][i]):
            DF_plot['Peak_fsr_5'][i] = DF_plot['Fsr_5'][i]

        if DF_plot['Fsr_6'][i] > (DF_plot['Mean_fsr_6'][i] +␣
↪DF_plot['Std_fsr_6'][i]):
            DF_plot['Peak_fsr_6'][i] = DF_plot['Fsr_6'][i]

        if DF_plot['Fsr_7'][i] > (DF_plot['Mean_fsr_7'][i] +␣
↪DF_plot['Std_fsr_7'][i]):
            DF_plot['Peak_fsr_7'][i] = DF_plot['Fsr_7'][i]

        #add one to the iterator
        i += 1

    #replace nan values wirth zeros
    #DF_plot = DF_plot.fillna(0)
```

```
    #Loop through the data
    for i in tqdm(range(len(data))):

        #sum the amount of peaks per minute
        DF_plot['std_feature_0'] =  DF_plot['Peak_fsr_0'].rolling(1000).count()
        DF_plot['std_feature_1'] =  DF_plot['Peak_fsr_1'].rolling(1000).count()
        DF_plot['std_feature_2'] =  DF_plot['Peak_fsr_2'].rolling(1000).count()
        DF_plot['std_feature_3'] =  DF_plot['Peak_fsr_3'].rolling(1000).count()
        DF_plot['std_feature_4'] =  DF_plot['Peak_fsr_4'].rolling(1000).count()
        DF_plot['std_feature_5'] =  DF_plot['Peak_fsr_5'].rolling(1000).count()
        DF_plot['std_feature_6'] =  DF_plot['Peak_fsr_6'].rolling(1000).count()
        DF_plot['std_feature_7'] =  DF_plot['Peak_fsr_7'].rolling(1000).count()

        #add one to the iterator
        i += 1

    #return the dataframe DF_plot
    return DF_plot,DF_time
```

[11]:
```
DF_plot1, DF_time = Feature(uit_bed1)
plt.plot(DF_plot1.index, DF_plot1['Fsr_0'])
plt.axvline(x=DF_time, color='purple', label="first bedstatus 0")
plt.plot(DF_plot1.index, DF_plot1['std_feature_0'], color = 'orange')
```

```
100%|      | 5439/5439 [01:53<00:00, 47.83it/s]
100%|      | 5439/5439 [01:50<00:00, 49.07it/s]
```

[11]:
```
[<matplotlib.lines.Line2D at 0x7fb0a0ab7400>]
```



[12]:
```
# create variables that can be changed for testing
Training_composition = [50,50] #percentage of the in-bed / out of bed
test_size = 0.3 #size of the validation dataset
```

[13]:
```
#Split the data into a training and test set by percentage
train, valid = train_test_split(DF_plot1, test_size=0.3)
```

[14]:
```
ModelColumns =␣
↪['std_feature_0','std_feature_1','std_feature_2','std_feature_3','std_feature_4','std_featu
```

[15]:
```
#Copy the features from the train set and store them in train_X
train_X = train[ModelColumns]


#Copy the features from the train set and store them in train_y
train_y = train['bed_status']

#Copy the features from the test set and store them in test_X
valid_X = valid[ModelColumns]

#Copy the features from the train set and store them in test_y
valid_y = valid['bed_status']
```

[16]:
```
train
```

[16]:

datetime	Fsr_0	Fsr_1	Fsr_2	Fsr_3	Fsr_4	Fsr_5	Fsr_6	\
2020-08-31 22:36:37.172000+00:00	349	230	471	583	456	658	705	
2020-08-31 22:37:51.015000+00:00	297	180	438	565	445	709	735	
2020-08-31 22:37:23.143000+00:00	358	238	479	590	458	678	721	
2020-08-31 22:39:24.980000+00:00	357	244	479	597	440	655	686	
2020-08-31 22:35:25.052000+00:00	378	224	465	575	434	649	678	
…	…	…	…	…	…	…	…	
2020-08-31 22:37:05.695000+00:00	357	240	475	588	454	669	713	
2020-08-31 22:39:22.084000+00:00	351	236	466	587	439	653	692	
2020-08-31 22:42:41.553000+00:00	380	120	235	288	225	344	379	
2020-08-31 22:43:21.641000+00:00	383	118	229	289	222	345	371	
2020-08-31 22:38:51.655000+00:00	348	228	462	573	425	630	661	

datetime	Fsr_7	std_feature_0	std_feature_1	…	\
				…	
2020-08-31 22:36:37.172000+00:00	422	95.0	90.0	…	
2020-08-31 22:37:51.015000+00:00	449	141.0	136.0	…	
2020-08-31 22:37:23.143000+00:00	434	146.0	137.0	…	
2020-08-31 22:39:24.980000+00:00	408	164.0	188.0	…	
2020-08-31 22:35:25.052000+00:00	402	24.0	11.0	…	
…	…	…	…	…	
2020-08-31 22:37:05.695000+00:00	437	142.0	126.0	…	
2020-08-31 22:39:22.084000+00:00	417	165.0	190.0	…	
2020-08-31 22:42:41.553000+00:00	267	131.0	143.0	…	
2020-08-31 22:43:21.641000+00:00	279	126.0	116.0	…	
2020-08-31 22:38:51.655000+00:00	403	152.0	175.0	…	

datetime	Peak_fsr_1	Peak_fsr_2	Peak_fsr_3	\
2020-08-31 22:36:37.172000+00:00	NaN	NaN	583	
2020-08-31 22:37:51.015000+00:00	NaN	NaN	NaN	
2020-08-31 22:37:23.143000+00:00	NaN	NaN	NaN	
2020-08-31 22:39:24.980000+00:00	NaN	NaN	NaN	
2020-08-31 22:35:25.052000+00:00	NaN	NaN	575	
…	…	…	…	
2020-08-31 22:37:05.695000+00:00	NaN	NaN	588	
2020-08-31 22:39:22.084000+00:00	236	NaN	587	
2020-08-31 22:42:41.553000+00:00	NaN	NaN	NaN	
2020-08-31 22:43:21.641000+00:00	NaN	NaN	NaN	
2020-08-31 22:38:51.655000+00:00	228	NaN	NaN	

datetime	Peak_fsr_4	Peak_fsr_5	Peak_fsr_6	\
2020-08-31 22:36:37.172000+00:00	NaN	NaN	705	

```
2020-08-31 22:37:51.015000+00:00         NaN         NaN         NaN
2020-08-31 22:37:23.143000+00:00         NaN         678         NaN
2020-08-31 22:39:24.980000+00:00         NaN         NaN         NaN
2020-08-31 22:35:25.052000+00:00         NaN         NaN         NaN
…                                        …           …           …
2020-08-31 22:37:05.695000+00:00         454         NaN         NaN
2020-08-31 22:39:22.084000+00:00         NaN         653         NaN
2020-08-31 22:42:41.553000+00:00         NaN         NaN         NaN
2020-08-31 22:43:21.641000+00:00         NaN         NaN         NaN
2020-08-31 22:38:51.655000+00:00         NaN         NaN         NaN

                                    Peak_fsr_7  bed_status  Mean_fsr_5  \
datetime
2020-08-31 22:36:37.172000+00:00         NaN           2  658.000000
2020-08-31 22:37:51.015000+00:00         NaN           2  711.666667
2020-08-31 22:37:23.143000+00:00         NaN           2  676.333333
2020-08-31 22:39:24.980000+00:00         NaN           2  655.000000
2020-08-31 22:35:25.052000+00:00         NaN           2  646.000000
…                                        …           …          …
2020-08-31 22:37:05.695000+00:00         NaN           2  668.333333
2020-08-31 22:39:22.084000+00:00         NaN           2  650.666667
2020-08-31 22:42:41.553000+00:00         NaN           2  344.666667
2020-08-31 22:43:21.641000+00:00         279           2  345.333333
2020-08-31 22:38:51.655000+00:00         403           2  631.000000

                                    Mean_fsr_6
datetime
2020-08-31 22:36:37.172000+00:00  704.333333
2020-08-31 22:37:51.015000+00:00  734.666667
2020-08-31 22:37:23.143000+00:00  720.333333
2020-08-31 22:39:24.980000+00:00  686.333333
2020-08-31 22:35:25.052000+00:00  679.000000
…                                      …
2020-08-31 22:37:05.695000+00:00  713.000000
2020-08-31 22:39:22.084000+00:00  692.000000
2020-08-31 22:42:41.553000+00:00  380.000000
2020-08-31 22:43:21.641000+00:00  374.333333
2020-08-31 22:38:51.655000+00:00  663.666667

[3807 rows x 42 columns]
```

[17]:
```
train.loc[train['bed_status'] == 2, 'bed_status'] = 1
```

[18]:
```
train
```

[18]:
```
                                    Fsr_0 Fsr_1 Fsr_2 Fsr_3 Fsr_4 Fsr_5 Fsr_6  \
datetime
```

```
2020-08-31 22:36:37.172000+00:00   349   230   471   583   456   658   705
2020-08-31 22:37:51.015000+00:00   297   180   438   565   445   709   735
2020-08-31 22:37:23.143000+00:00   358   238   479   590   458   678   721
2020-08-31 22:39:24.980000+00:00   357   244   479   597   440   655   686
2020-08-31 22:35:25.052000+00:00   378   224   465   575   434   649   678
…                                  …     …     …     …     …     …     …
2020-08-31 22:37:05.695000+00:00   357   240   475   588   454   669   713
2020-08-31 22:39:22.084000+00:00   351   236   466   587   439   653   692
2020-08-31 22:42:41.553000+00:00   380   120   235   288   225   344   379
2020-08-31 22:43:21.641000+00:00   383   118   229   289   222   345   371
2020-08-31 22:38:51.655000+00:00   348   228   462   573   425   630   661

                                   Fsr_7  std_feature_0  std_feature_1  …  \
datetime                                                                 …
2020-08-31 22:36:37.172000+00:00   422           95.0           90.0  …
2020-08-31 22:37:51.015000+00:00   449          141.0          136.0  …
2020-08-31 22:37:23.143000+00:00   434          146.0          137.0  …
2020-08-31 22:39:24.980000+00:00   408          164.0          188.0  …
2020-08-31 22:35:25.052000+00:00   402           24.0           11.0  …
…                                  …              …              …  …
2020-08-31 22:37:05.695000+00:00   437          142.0          126.0  …
2020-08-31 22:39:22.084000+00:00   417          165.0          190.0  …
2020-08-31 22:42:41.553000+00:00   267          131.0          143.0  …
2020-08-31 22:43:21.641000+00:00   279          126.0          116.0  …
2020-08-31 22:38:51.655000+00:00   403          152.0          175.0  …

                                   Peak_fsr_1  Peak_fsr_2  Peak_fsr_3  \
datetime
2020-08-31 22:36:37.172000+00:00         NaN         NaN         583
2020-08-31 22:37:51.015000+00:00         NaN         NaN         NaN
2020-08-31 22:37:23.143000+00:00         NaN         NaN         NaN
2020-08-31 22:39:24.980000+00:00         NaN         NaN         NaN
2020-08-31 22:35:25.052000+00:00         NaN         NaN         575
…                                          …           …           …
2020-08-31 22:37:05.695000+00:00         NaN         NaN         588
2020-08-31 22:39:22.084000+00:00         236         NaN         587
2020-08-31 22:42:41.553000+00:00         NaN         NaN         NaN
2020-08-31 22:43:21.641000+00:00         NaN         NaN         NaN
2020-08-31 22:38:51.655000+00:00         228         NaN         NaN

                                   Peak_fsr_4  Peak_fsr_5  Peak_fsr_6  \
datetime
2020-08-31 22:36:37.172000+00:00         NaN         NaN         705
2020-08-31 22:37:51.015000+00:00         NaN         NaN         NaN
2020-08-31 22:37:23.143000+00:00         NaN         678         NaN
2020-08-31 22:39:24.980000+00:00         NaN         NaN         NaN
2020-08-31 22:35:25.052000+00:00         NaN         NaN         NaN
```

```
…                                          …          …          …
2020-08-31 22:37:05.695000+00:00         454        NaN        NaN
2020-08-31 22:39:22.084000+00:00         NaN        653        NaN
2020-08-31 22:42:41.553000+00:00         NaN        NaN        NaN
2020-08-31 22:43:21.641000+00:00         NaN        NaN        NaN
2020-08-31 22:38:51.655000+00:00         NaN        NaN        NaN

                                   Peak_fsr_7  bed_status  Mean_fsr_5  \
datetime
2020-08-31 22:36:37.172000+00:00         NaN           1  658.000000
2020-08-31 22:37:51.015000+00:00         NaN           1  711.666667
2020-08-31 22:37:23.143000+00:00         NaN           1  676.333333
2020-08-31 22:39:24.980000+00:00         NaN           1  655.000000
2020-08-31 22:35:25.052000+00:00         NaN           1  646.000000
…                                        …           …          …
2020-08-31 22:37:05.695000+00:00         NaN           1  668.333333
2020-08-31 22:39:22.084000+00:00         NaN           1  650.666667
2020-08-31 22:42:41.553000+00:00         NaN           1  344.666667
2020-08-31 22:43:21.641000+00:00         279           1  345.333333
2020-08-31 22:38:51.655000+00:00         403           1  631.000000

                                   Mean_fsr_6
datetime
2020-08-31 22:36:37.172000+00:00  704.333333
2020-08-31 22:37:51.015000+00:00  734.666667
2020-08-31 22:37:23.143000+00:00  720.333333
2020-08-31 22:39:24.980000+00:00  686.333333
2020-08-31 22:35:25.052000+00:00  679.000000
…                                        …
2020-08-31 22:37:05.695000+00:00  713.000000
2020-08-31 22:39:22.084000+00:00  692.000000
2020-08-31 22:42:41.553000+00:00  380.000000
2020-08-31 22:43:21.641000+00:00  374.333333
2020-08-31 22:38:51.655000+00:00  663.666667

[3807 rows x 42 columns]
```

[19]:
```
#scaler = StandardScaler()
#train_X = scaler.fit_transform(train_X)
#valid_X = scaler.fit_transform(valid_X)
```

[20]:
```
#Convert the training sets to a numpy array and convert all values to int's
train_X = np.array(train_X).astype('int')
train_y = np.array(train_y).astype('int')

#Convert the test sets to a numpy array and convert all values to int's
valid_X = np.array(valid_X).astype('int')
```

```
valid_y = np.array(valid_y).astype('int')
```

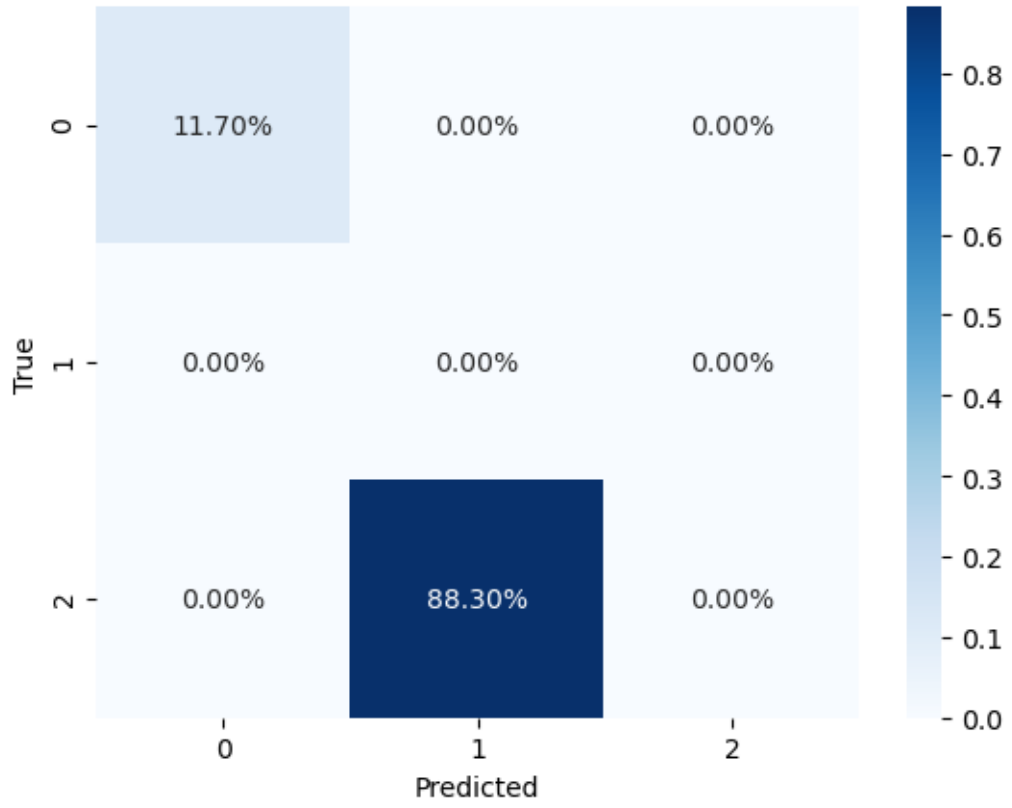
[21]:
```
#Linear Support vector classifier model

#Initialize and fit the model on training data
model = LinearSVC(dual=False)
model.fit(train_X, train_y)
```

[21]:
```
LinearSVC(dual=False)
```

[22]:
```
#predict the bedstatus on the validation set valid_X
pred_y = model.predict(valid_X)
```

[23]:
```
#Get the number of true negatives,false negatives, true positives and false␣
 ↪negatives for the validation set
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(valid_y, pred_y)
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]

import seaborn as sns
ax = sns.heatmap(CM/np.sum(CM), annot=True,
            fmt='.2%', cmap='Blues')
ax.set(xlabel="Predicted", ylabel = "True")
```

[23]:
```
[Text(50.722222222222214, 0.5, 'True'),
 Text(0.5, 23.52222222222222, 'Predicted')]
```



[24]:
```
#Score the model based on the validation data and print the outcome
print("Model scores based on validation data:")
print("")
print("accuracy score: " + str(accuracy_score(valid_y, pred_y)))
print("recall score: " + str(recall_score(valid_y, pred_y)))
print("precision score: " + str(precision_score(valid_y, pred_y)))
print("")
print("")
print("")
```

```
Model scores based on validation data:

accuracy score: 0.1170343137254902
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-84f19cef08cf> in <module>
      3 print("")
      4 print("accuracy score: " + str(accuracy_score(valid_y, pred_y)))
----> 5 print("recall score: " + str(recall_score(valid_y, pred_y)))
      6 print("precision score: " + str(precision_score(valid_y, pred_y)))
```

```
      7 print("")

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/sklearn/utils/validation.py␣
 ↪in inner_f(*args, **kwargs)
     70                             FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/sklearn/metrics/
 ↪_classification.py in recall_score(y_true, y_pred, labels, pos_label, average␣
 ↪sample_weight, zero_division)
   1739                                                 warn_for=('recall',),
   1740                                                ␣
 ↪sample_weight=sample_weight,
-> 1741                                                ␣
 ↪zero_division=zero_division)
   1742     return r
   1743

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/sklearn/utils/validation.py␣
 ↪in inner_f(*args, **kwargs)
     70                             FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/sklearn/metrics/
 ↪_classification.py in precision_recall_fscore_support(y_true, y_pred, beta,␣
 ↪labels, pos_label, average, warn_for, sample_weight, zero_division)
   1432         raise ValueError("beta should be >=0 in the F-beta score")
   1433     labels = _check_set_wise_labels(y_true, y_pred, average, labels,
-> 1434                                     pos_label)
   1435
   1436     # Calculate tp_sum, pred_sum, true_sum ###

/opt/jupyterhub/anaconda/lib/python3.6/site-packages/sklearn/metrics/
 ↪_classification.py in _check_set_wise_labels(y_true, y_pred, average, labels,␣
 ↪pos_label)
   1263             raise ValueError("Target is %s but average='binary'. Please "
   1264                              "choose another average setting, one of %r "
-> 1265                              % (y_type, average_options))
   1266     elif pos_label not in (None, 1):
   1267         warnings.warn("Note that pos_label (set to %r) is ignored when
```

```
ValueError: Target is multiclass but average='binary'. Please choose another␣
↪average setting, one of [None, 'micro', 'macro', 'weighted'].
```

[ ]:
```
#Plot 1: show the composition of the Dataset used to train the model

#calculate the composition of the training set
train_0 = sum(train['Bed status'] == 0)
train_1 = sum(train['Bed status'] == 1)
train_total = len(train)
Percentage_train_0 = train_0/train_total * 100
Percentage_train_1 = train_1/train_total * 100

# compositions plot training data
All_percentage_train = [Percentage_train_0,Percentage_train_1]
bars_train = ('bedstatus  0', ' bedstatus  2')
y_pos = np.arange(len(bars_train))

#Plot the training data with custom title
plt.bar(y_pos, All_percentage_train, color=['#DC267F', '#FE6100',],ec='black')
plt.title("Composition training dataset bedstatus 0 and 2 ")

# use the xticks function to add custom labels
plt.xticks(y_pos)
plt.xlabel(bars_train)
plt.ylabel("percentage")
plt.ylim(0, 100)
```