UEBA ALGORITHMS

Samra

January 9, 2023

Contents

1 Matrix Profile **2**
- 1.1 Introduction . . . 2
- 1.2 Code . . . 2
- 1.3 Results . . . 5

2 ANN for User Classification **11**
- 2.1 Introduction . . . 11
- 2.2 Data Used . . . 11
- 2.3 Model . . . 12
- 2.4 Model . . . 13
- 2.5 Results . . . 13

1 Matrix Profile

1.1 Introduction

The Matrix Profile is a novel data structure with two primary components; a distance profile and profile index. The distance profile is a vector of minimum Z-Normalized Euclidean Distances. The profile index contains the index of its first nearest-neighbor.

The main goal of **Time series Analysis** is to find *motifs* and *anomalies.*

In this research, Stumpy based profiling is done to identify *anomalies.*

- **Stumpy** stump is Numba JIT-compiled version of the popular STOMP algorithm. It performs an ordered search for patterns and outliers within a specified time series.
- **Anomaly** A reference sub-sequence with a large matrix profile value (nearest neighbor is "faraway")
- **Motifs** a reference sub-sequence with a small matrix profile value.

1.2 Code

Functions to calculate and visualize matrix profiles of any given data for later use. **Imports**

```
import pandas as pd
import stumpy as sp
import matplotlib.pyplot as plt
import numpy as np
import os
from matplotlib.patches import Rectangle
```

Function to Calculate Profile

This function will take *1D np-array and a list of window size with labels* to compute matrix profile using *stumpy.stomp* function on each window and returns corresponding matrix profiles.

```
  def cal_profile(data,windows =windows):
''' data should be values of a column '''
    profiles = {}

    for label, window_size in windows:
        mp=sp.stump(data, window_size,ignore_trivial = True)
        key = '{} Profile'.format(label)
        profiles[key] = mp
    return profiles
```

Functions to Sort Index

```
def sort_index(profiles,windows =windows):
 ''' this function sorts matrix profile indices'''

    sorted_index=[]

    for label,value in windows:
        key='{} Profile'.format(label)
        sort=np.argsort(profiles[key][:,0])
        sorted_index.append(sort)

    return sorted_index
```

Functions to Find Anomalies

Find_motif_discord takes profile as input, using above defined sort_index function, sort indices, to get last 5 indices(farthest) as anomalies.

```
def find_motif_discord(profiles,windows =windows):
    sorted_index=sort_index(profiles,windows)
    motifs={}
    neighbors={}
    discords={}
    discord_distance={}
    i=0
    for label,window in windows:

        key='{} Profile'.format(label)

        motif=sorted_index[i][0]
        motifs[label]=motif
        neighbors[label]=profiles[key][motif,1]

        x=len(sorted_index[i])
        discord=sorted_index[i][x:x-6:-1]
        discord

        discords[label]=discord

        i+=1

    return motifs,neighbors, discords, discord_distance
```

Function to Visualize Profile and Anomalies **This function compute and plot the matrix profile, original data and anomalous points in both figures of each user for given data.**

```
def visualize(user_name,data,data_index,figname,
windows = windows,ylabel='Data'):
    profiles=cal_profile(data=data,windows=windows)
    motifs,neighbors,discords,distance=find_motif_discord(profiles)
    fig, axes = plt.subplots(len(windows)*2,1,figsize=(15,20))
    ax_idx=0
    axes[ax_idx].set_title(user_name)
    for label, window in windows:
        key = '{} Profile'.format(label)
        profile = profiles[key]
        axes[ax_idx].plot(data)
        axes[ax_idx].set_title(user_name+ ' '+label,fontsize=20)
        axes[ax_idx].set_xlim(0,len(data_index))

        ax_idx+=1
        #display discord
        axes[ax_idx].plot(profile[:,0])

        for v in discords[label]:
            x=(data[v:v+window])
            print(x)
            t=np.arange(v,v+window)
            axes[ax_idx -1].plot(t,x,color='red')
            axes[ax_idx].axvline(x=v,
            linestyle="dashed",
            color='red', marker='o')
            axes[ax_idx].set_title('Discords')
            axes[ax_idx].set_ylabel('Matrix Profile')
        ax_idx+=1
    plt.xlabel('Date')
    plt.tight_layout()
    plt.savefig(user_name+figname +'.jpg')
    return fig
```

Automate matrix profiling on all users

```
s=os.listdir(os.getcwd()) #list all files in directory
t only csv files
iles=[]
file in files :
if file.endswith('csv'):
    csvfiles.append(file)

mpute and display matrix profiles of all users.
user_file in csvfiles:
user=pd.read_csv(user_file,index_col=0)
cols=user.columns
for i in range(len(user.columns)):
    profile=cal_profile(user[cols[i]].values)
    visualize(user_file,user[cols[i]].values,
    user.index,figname=cols[i])
```

1.3 Results

Profile for each feature (Negative, Positve and neutral for user GHH0288 is attached below. Compute and Analyze Matrix Profiles for few users Above defined functions are used here to compute matrix profiles of sentiments of emails for 5 users with different window sizes (1 is attached below). window sizes

```
windows=[('window=3', 3),('window=4', 4),('window=5', 5),]
```

Case 1: Profile of user GHH0288 for 6 months data for negative emails

```
user=pd.read_csv('GHH0288.csv',index_col=0)
user.index=pd.to_datetime(user.index)
user.index
user=user[(user.index.month<=6)& (user.index.year==2010)]

user.describe()
```

Visualize Data

```
plt.figure(figsize=(10,3))
plt.locator_params('both',len(user['Negative']))
user['Negative'].plot()
```

str_date	Negative	Neutral	Positive
2010-01-04	0.0	0.0	1.0
2010-01-05	0.0	0.0	1.0
2010-01-06	0.0	0.0	1.0
2010-01-07	0.0	0.0	1.0
2010-01-08	0.0	0.0	1.0
...	...	...	...
2010-06-24	0.0	0.0	1.0
2010-06-25	0.0	0.0	1.0
2010-06-28	1.0	0.0	0.0
2010-06-29	0.0	0.0	1.0
2010-06-30	1.0	0.0	0.0

Figure 1: 6 months Data of *GHH0288* for negative emails.

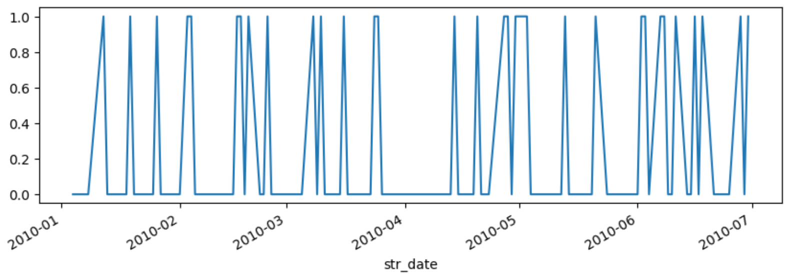

Figure 2: Visualization of data

Matrix Profile of GHH0288 for 6 months for negative emails

```
profile=cal_profile(user['Negative'].values)
x=visualize('GHH0288.csv',user['Negative'].values,
    data_index=user.index.date,figname='negative_profile')
```

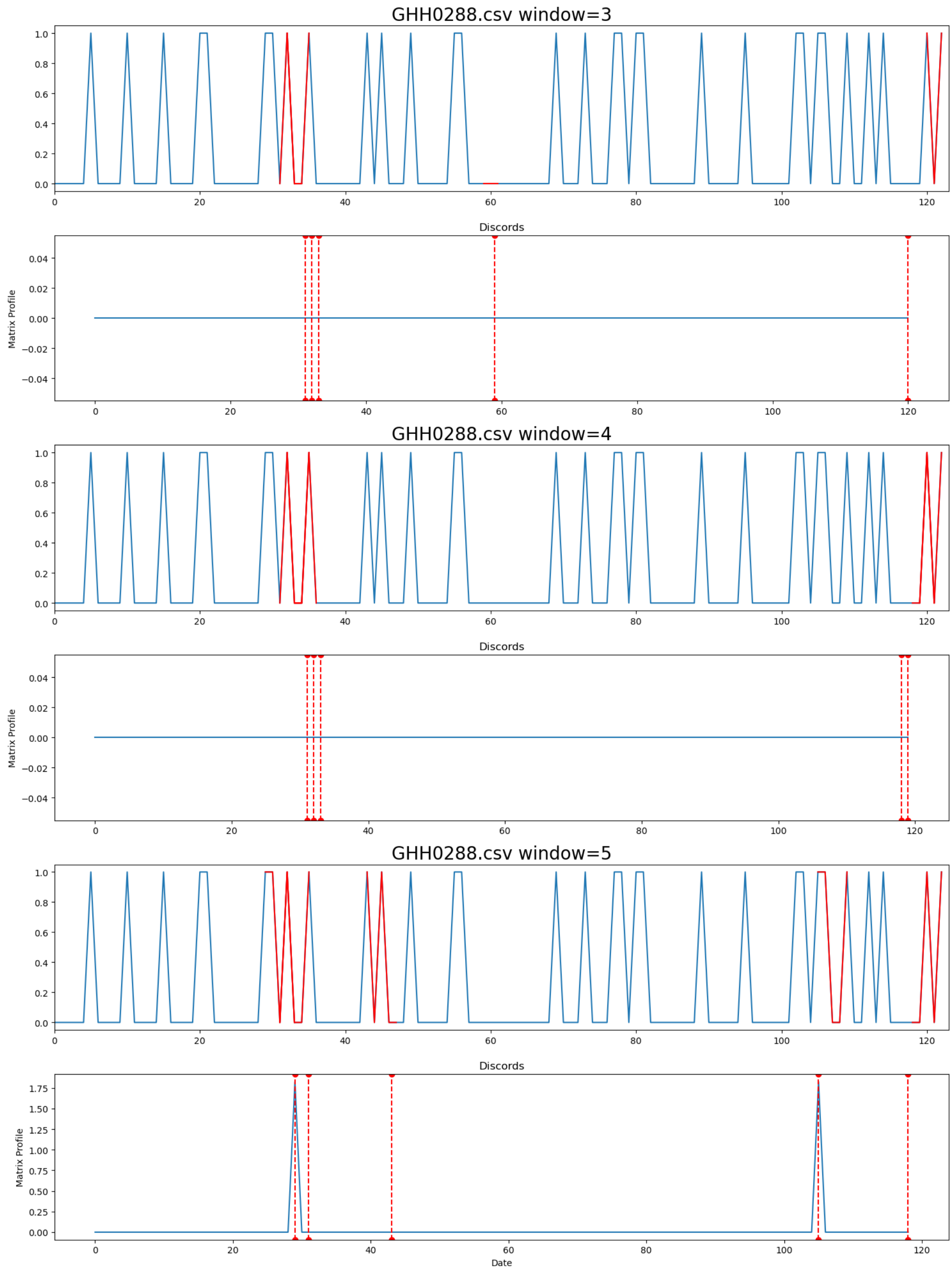

Figure 3: 6 months Matrix profile *GHH0288*.

Case2: Matrix Profile of *GHH0288* for complete data

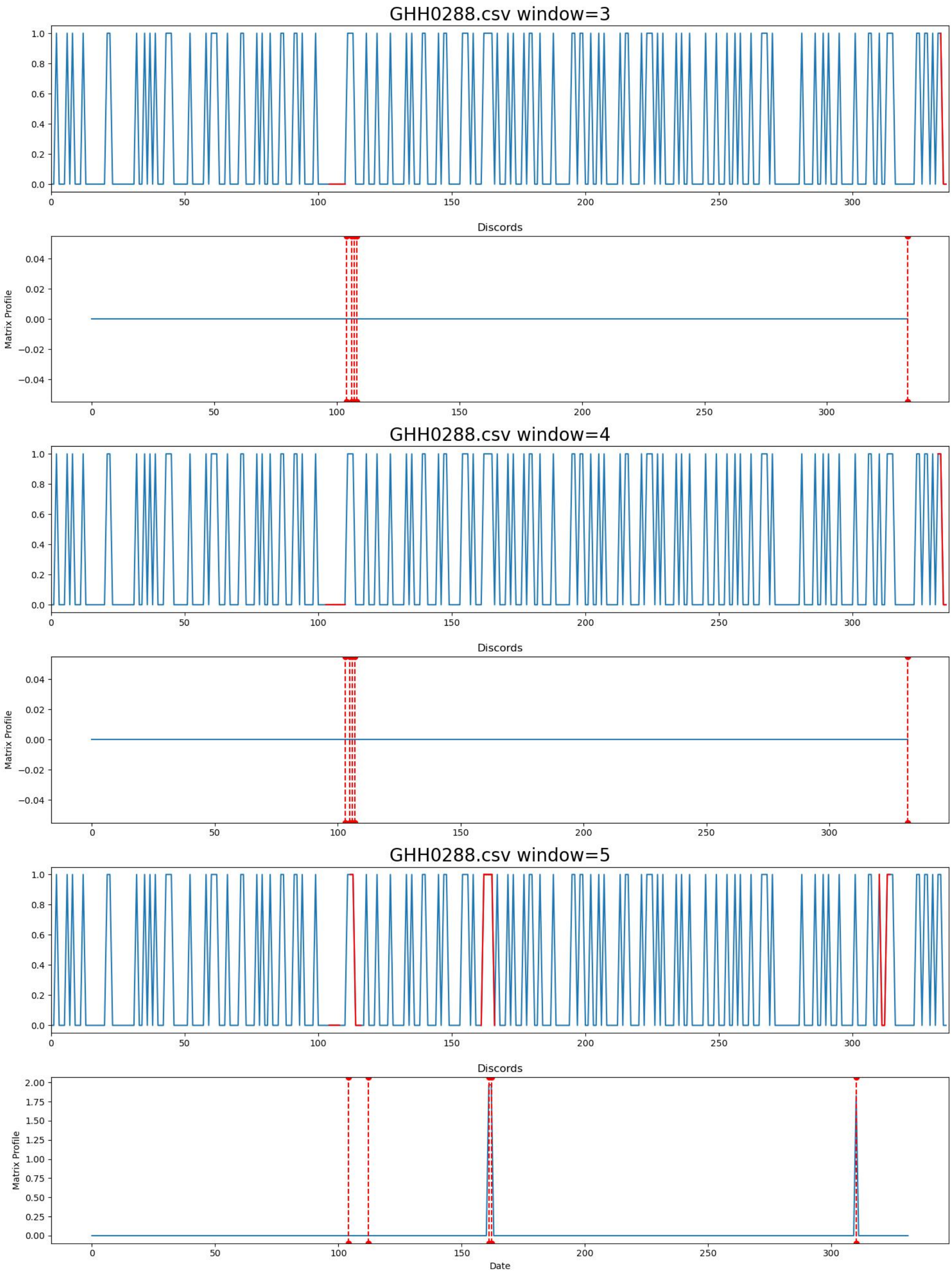

Figure 4: Negative profile of user *GHH0288*.

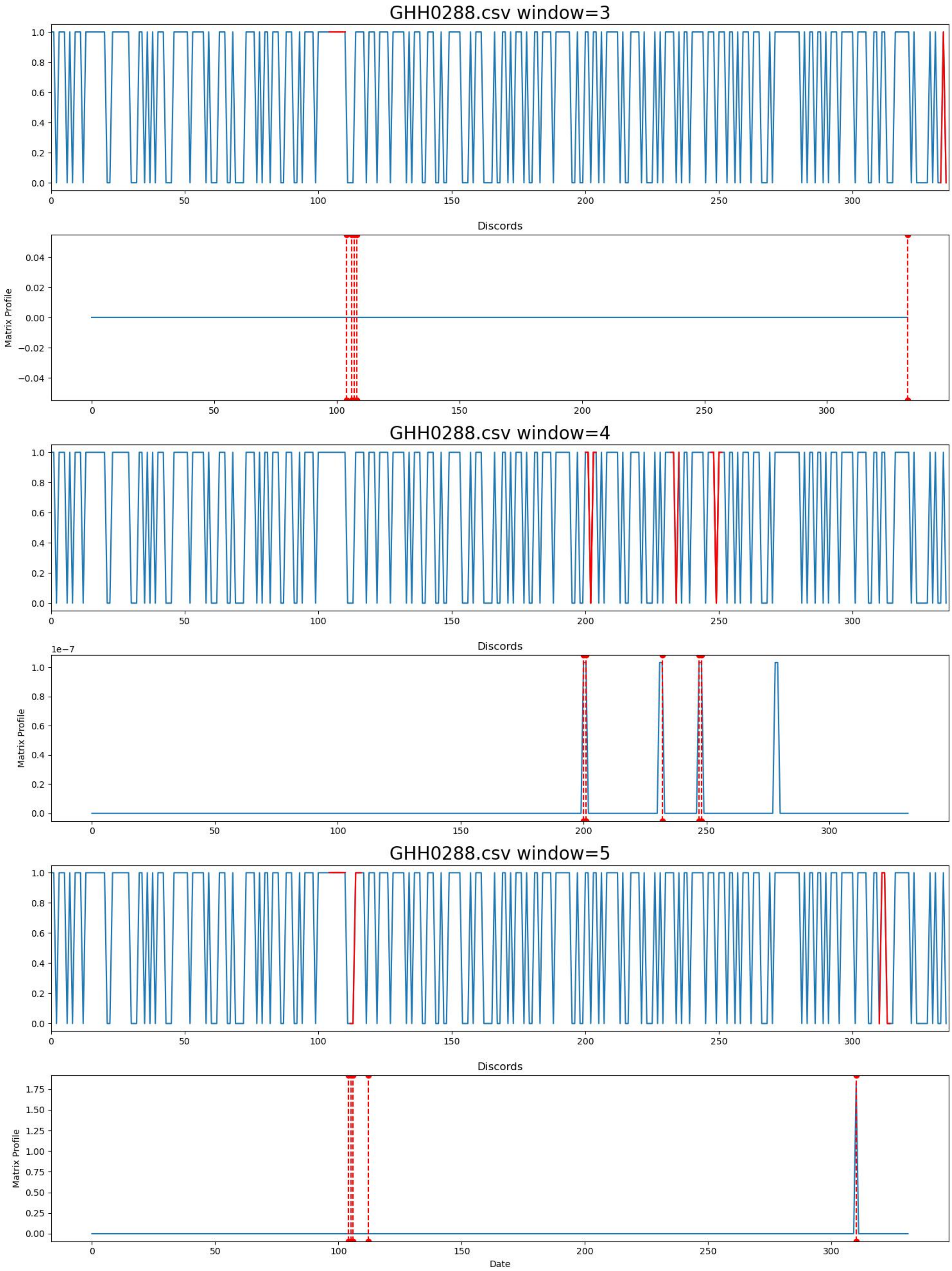

Figure 5: Positive profile of user *GHH0288*.

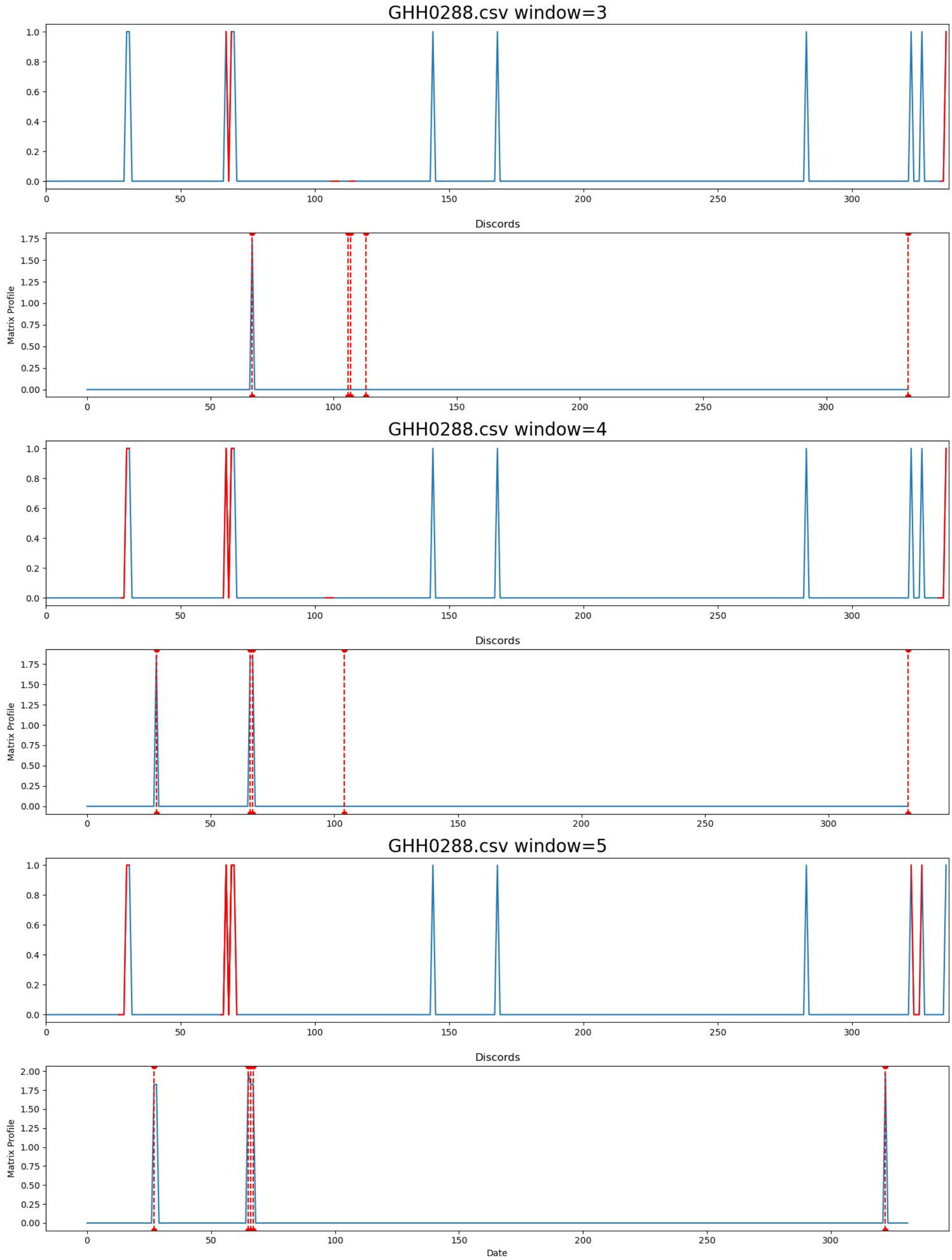

Figure 6: Neutral profile of user *GHH0288*.

where the red lines shows the anomalies.

2 ANN for User Classification

2.1 Introduction

To predict the user from it's behavior, Vanilla artificial Neural Networks was applied. Data used is given in section 2.2.

2.2 Data Used

All user files were merged after adding a new column *"user_name"* and shuffled fig 7. User names were encoded using *One Hot encoding* fig 8.

```
path=r'C:\Users\TLS\Desktop\FYDP_updated\user_files'
all_users=glob.glob(os.path.join(path + "\*.csv"),recursive = True)
all_users
data=pd.DataFrame()
for user in all_users:
    name=os.path.basename(user).strip('.csv')
    aux=pd.read_csv(user)
    aux['user_name']=name
    data=pd.concat([aux,data],axis=0)
data=data.sample(frac=1)
data.to_csv('concatenated_users.csv')
data
```

	str_date	Negative	Neutral	Positive	Logoff	Logon	Connect	Disconnect	user_name
100	3/14/2011	6	0	4	1	2	0	0	IBM0671
42	2/1/2011	3	0	5	1	1	0	0	GCR0554
201	6/3/2010	7	0	8	1	1	0	0	IDO0176
185	5/13/2010	1	1	7	1	1	0	0	DLS0383
283	10/14/2010	1	0	0	1	1	0	0	FWM0707
277	9/24/2010	1	0	2	1	2	1	1	SAB0781
259	8/25/2010	8	0	6	1	2	2	2	RUM0880
137	7/20/2010	3	0	8	2	3	2	2	AKR0057
15	1/11/2010	0	0	3	1	1	0	0	IIL0513
83	3/2/2010	4	0	5	1	1	0	0	DZC0228
301	10/25/2010	7	1	3	1	1	0	0	GRC0145
287	10/15/2010	1	0	0	1	1	0	0	KQC0708
340	9/22/2010	0	0	0	1	1	0	0	CJM0138
290	10/11/2010	9	1	5	1	1	0	0	LPC0492
151	4/21/2011	0	0	1	1	1	0	0	MSG0839
120	3/28/2011	4	1	10	1	2	1	1	BBG0325
161	4/20/2011	1	0	0	1	2	1	1	AHD0848

Figure 7: Merged User data.

Data Preproessing

```
hot=ce.OneHotEncoder() # encoding
hot.fit(data['user_name'])
e_hot.transform(data['user_name'])

ded=pd.concat([data,y],axis=1) #combine dataset
ded.to_csv('encoded_users.csv')
```

	str_date	Negative	Neutral	Positive	Logoff	Logon	Connect	Disconnect	user_name	IBM0671	...	EAH0466	EGD0132	AHG0634	LHB0606	JRH0455
100	3/14/2011	6	0	4	1	2	0	0	IBM0671	1	...	0	0	0	0	0
42	2/1/2011	3	0	5	1	1	0	0	GCR0554	0	...	0	0	0	0	0
201	6/3/2010	7	0	8	1	1	0	0	IDO0176	0	...	0	0	0	0	0
185	5/13/2010	1	1	7	1	1	0	0	DLS0383	0	...	0	0	0	0	0
283	10/14/2010	1	0	0	1	1	0	0	FWM0707	0	...	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
80	2/28/2011	5	0	6	1	2	1	1	RVF0201	0	...	0	0	0	0	0
250	8/13/2010	6	1	8	1	2	3	3	MTB0620	0	...	0	0	0	0	0
295	10/18/2010	7	0	2	1	1	0	0	KSH0569	0	...	0	0	0	0	0
97	3/11/2011	4	0	6	1	2	8	8	CTA0020	0	...	0	0	0	0	0
248	8/11/2010	2	0	7	1	1	0	0	AMH0794	0	...	0	0	0	0	0

Figure 8: Encoded User data

2.3 Model

User 'IBM0671'

```
encoded=pd.read_csv('encoded_users.csv',nrows=5000,index_col=0)

# encoded=encoded.drop('user_name',axis=1)
encoded

x=encoded.iloc[:,1:7]
x
y=encoded.iloc[:,9:]
y

x_train,x_test,y_train,y_test=
    train_test_split(x, y['IBM0671'], test_size=0.3, random_state=42)
```

x_train

	Negative	Neutral	Positive	Logoff	Logon	Connect
17	2	2	11	1	2	0
302	1	0	8	2	2	0
125	5	0	10	1	1	0
206	4	1	4	1	1	0
320	0	0	1	1	1	0
...	...	...	...	...	...	...
277	2	0	7	1	2	0
192	8	1	7	1	2	0
216	1	0	0	1	1	9
98	8	1	5	1	1	0
81	4	0	6	1	1	0

3500 rows × 6 columns

Figure 9: Train data for *'IBM0671'*.

2.4 Model

```
equential()
zer = HeNormal()
d(Dense(6,  kernel_initializer=initializer ,activation = 'relu',
            input_shape=(6,))) #input layer

d(Dense(12,kernel_initializer=initializer ,
on='relu')) #hidden layer
d(Dense(1,activation='sigmoid')) #output layer
```

```
l.compile(optimizer = 'adam',
 = 'binary_crossentropy',
ics =[tf.keras.metrics.Recall(),
eras.metrics.Precision(),
uracy'])

l.fit(x_train,␣y_train,␣batch_size=32,␣epochs␣=␣100)
```

2.5 Results

Matrix Profile of GHH0288 for 6 months

```
,recall ,precision , accuracy = model.evaluate(x_test , y_test)
print('␣Categorical␣Accuracy:␣%.2f' % (accuracy*100))
print('␣recall:␣%.2f' % (recall*100))
print('␣Precision:␣%.2f' % (precision*100))

t([[6.0,0.0,4.0,1.0,2.0,0.0]])
```

Training Accuracy	99.86
Testing Accuracy	99.87
Precision	0.00
Recall	0.00

Table 1: Results of ANN for *'IBM0671'*.

True positive rate is 0, due to which Precision and recall is 0. A reason for that is data is highly imbalanced, and due to limitation of computation resources, model is trained and tested only on 5000 samples, which may contain a few Positive samples.