In [1]:

**from** **datetime** **import** timedelta

**import** **time**

**import** **numpy** **as** **np**

**from** **functools** **import** partial

**from** **sklearn.model\_selection** **import** train\_test\_split

**import** **itertools**

**import** **pickle**

**import** **pandas** **as** **pd**

**from** **functools** **import** partial

**import** **itertools**

**from** **multiprocessing** **import** Pool

In [2]:

**def** getCol2Count(df,col):

**return** dict(df[col].value\_counts())

**def** \_\_removeChatteringAlarmsHelper(alarms,chattering\_timedelta\_threshold, chattering\_count\_threshold):

*"""Find the chatterings in an alarms list from the same source.*

*"""*

alarms\_without\_chattering = []

alarms = [alarm **for** alarm **in** sorted(alarms, key=**lambda** arg: arg["StartTime"], reverse=**False**)]

i = 0

j = 0

**while** i < (len(alarms)):

alarms\_without\_chattering.append(alarms[i])

prev\_start = alarms[i]["StartTime"]

prev\_end = alarms[i]["EndTime"]

count\_alarms = 0

j = i + 1

**while** j < len(alarms):

next\_start = alarms[j]["StartTime"]

next\_end = alarms[j]["EndTime"]

*# this assert is very important: the prev alarm has to turn off before the start of*

*# the next one*

**assert**(prev\_start <= next\_start)

**assert**(prev\_end <= next\_start)

**assert**(prev\_end <= next\_end)

delta = timedelta.total\_seconds(next\_start - prev\_start)

**assert** (delta >= 0)

**if** delta > chattering\_timedelta\_threshold:

**break**

count\_alarms += 1

j += 1

**if** count\_alarms >= chattering\_count\_threshold:

i = j

**else**:

i += 1

**return** alarms\_without\_chattering

**def** tempFun(df,chat\_delta,chat\_count,sname):

alarms\_without\_chatterings = []

df\_source = df.loc[df['SourceName'].isin([sname])]

**for** condition **in** df\_source["Condition"].unique():

df\_condition = df\_source.loc[df\_source['Condition'].isin([condition])]

alarms = \_\_removeChatteringAlarmsHelper(df\_condition.to\_dict(orient="records"),chattering\_timedelta\_threshold=chat\_delta,chattering\_count\_threshold=chat\_count)

alarms\_without\_chatterings = alarms\_without\_chatterings + alarms

**return** alarms\_without\_chatterings

**def** removeChatteringAlarms(df,chattering\_timedelta\_threshold=**None**,chat\_count=**None**):

*# for sname in df["SourceName"].unique():*

alarms\_without\_chatterings = []

sources=[sname **for** sname **in** df["SourceName"].unique()]

myFun = partial(tempFun,df,chattering\_timedelta\_threshold,chat\_count)

**with** Pool(3) **as** p:

alarms\_without\_chatterings = p.map(myFun, sources)

alarms\_without\_chatterings = list(itertools.chain.from\_iterable(alarms\_without\_chatterings))

**return** pd.DataFrame(alarms\_without\_chatterings)

**class** **AlarmsProcessing**:

**def** \_\_init\_\_(self,config) -> **None**:

self.config = config

self.alias2name = **None**

self.name2alias = **None**

self.df = pd.read\_csv(self.config['file-path'], low\_memory=**False**, usecols=self.config['usecols'],parse\_dates=self.config['date-cols'])

**if** self.config['alias']:

self.convertSourceNamesToAlias()

**for** m **in** self.df["Year-Month"].unique():

month\_df = self.df[self.df["Year-Month"].isin([m])].sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**)

days = sorted(list(month\_df['Day'].unique()))

print(f"[**{**m**}**],Days:**{**days**}**")

**def** convertSourceNamesToAlias(self):

alias2name = {f"A**{**k**}**": v **for** k, v **in** enumerate(self.df["SourceName"].unique())}

name2alias = {v: k **for** k, v **in** alias2name.items()}

self.df["SourceName"] = self.df["SourceName"].apply(**lambda** sname: name2alias[sname])

self.alias2name = alias2name

self.name2alias = name2alias

*# return name2alias, alias2name*

**def** removeChatteringAlarms(self,df,chattering\_timedelta\_threshold=60,chattering\_count\_threshold=2):

**return** removeChatteringAlarms(df=df,chattering\_timedelta\_threshold=chattering\_timedelta\_threshold,chat\_count=chattering\_count\_threshold)

**def** removeMomentaryAlarms(self,df,monmentarly\_filter=**None**):

df = df[(df["TimeDelta"] > monmentarly\_filter)].reset\_index(drop=**True**)

**return** df

**def** removeStalingAlarms(self,df,staling\_filter=**None**):

df = df[(df["TimeDelta"] < staling\_filter)].reset\_index(drop=**True**)

**return** df

**def** removeConditionsAlarms(self,df,conditions\_filter):

df = df[~df["Condition"].isin(conditions\_filter)].reset\_index(drop=**True**)

**return** df

**def** removeSources(self,df, sources\_filter):

df = df[(~df["SourceName"].isin(sources\_filter))].reset\_index(drop=**True**)

**return** df

**def** removeSourcesBasedOnMinCount(self,df,min\_alarms\_per\_source\_filter):

source2count = dict(df["SourceName"].value\_counts())

select\_sources = [k **for** k, v **in** source2count.items() **if** v >= min\_alarms\_per\_source\_filter]

df = df[df["SourceName"].isin(select\_sources)]

**return** df

**def** getDFWithCommonSourcesInAllMonths(self,df):

each\_month\_source\_names = [[sname **for** sname **in** df[df["Year-Month"]==month]["SourceName"].unique()] **for** month **in** df["Year-Month"].unique()]

common\_sourcenames\_in\_all\_months = set.intersection(\*[set(l) **for** l **in** each\_month\_source\_names])

df = df[df["SourceName"].isin(common\_sourcenames\_in\_all\_months)]

**return** df

**def** updatSourceNamewithCondition(self,df):

**def** \_concatenateSourceNameAndCondition(sname, condition):

**return** sname+"-"+condition

df["SourceName"] = df[["SourceName", "Condition"]].apply(

**lambda** arg: \_concatenateSourceNameAndCondition(\*arg), axis=1)

**return** df

**def** getCondition2Sources(self,df,condition):

**return** df[df["Condition"]==condition]["SourceName"].unique()

In [3]:

**class** **PrepareDataset**:

**def** \_\_init\_\_(self,config) -> **None**:

self.config = config

**def** getSeqsFromAlarmsDF(self,df,seq\_duration\_gap,filter\_short\_seq):

print(f">> Duration to next seq: **{**seq\_duration\_gap**}**, ignore seq len: **{**filter\_short\_seq**}**")

list\_of\_sequences = []

alarms= df.sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**).to\_dict(orient="records")

alarms = [alarm **for** alarm **in** sorted(alarms, key=**lambda** arg: arg["StartTime"], reverse=**False**)] *# sorting*

*# print('check',len(alarms))*

**assert** alarms[0]['StartTime'] < alarms[-1]['StartTime']

i =0

j= 0

max\_seq\_len = -1

**while** i <len(alarms):

prev\_start = alarms[i]["StartTime"]

seq = []

seq.append(alarms[i])

j = i+1

**while** j < len(alarms):

next\_start = alarms[j]["StartTime"]

delta = timedelta.total\_seconds(next\_start - prev\_start)

*# print(delta)*

**assert** delta >= 0

**if** delta >= seq\_duration\_gap:

**break**

seq.append(alarms[j])

j += 1

i = j

**if** len(seq) > max\_seq\_len:

max\_seq\_len = len(seq)

**if** len(seq)>=filter\_short\_seq:

seq = [alarm **for** alarm **in** sorted(seq, key=**lambda** arg: arg["StartTime"], reverse=**False**)]

seq = [alarm["SourceName"] **for** alarm **in** seq]

list\_of\_sequences.append(seq)

**return** list\_of\_sequences, max\_seq\_len

**def** splitDFtoTrainValidDfsPerMonthByRows(self,df,p=0.2):

tarain\_dfs = []

valid\_dfs = []

**for** m **in** df["Year-Month"].unique():

month\_df = df[(df["Year-Month"].isin([m]))].sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**)

train\_df = month\_df[0:int(len(month\_df)\*(1-p))]

train\_df = train\_df.reset\_index(drop=**True**)

valid\_df = month\_df[int(len(month\_df)\*(1-p)):]

valid\_df = valid\_df.reset\_index(drop=**True**)

**assert** len(train\_df)+len(valid\_df) == len(month\_df)

tarain\_dfs.append(train\_df)

valid\_dfs.append(valid\_df)

t\_df = pd.concat(tarain\_dfs, ignore\_index=**True**).sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**)

v\_df = pd.concat(valid\_dfs, ignore\_index=**True**).sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**)

**assert** len(t\_df)+len(v\_df) == len(df)

**return** t\_df, v\_df

**def** splitDFtoTrainValidDfsPerMonthByDays(self,df,p=0.2):

tarain\_dfs = []

valid\_dfs = []

**for** m **in** df["Year-Month"].unique():

month\_df = df[(df["Year-Month"].isin([m]))].sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**)

days = sorted(list(month\_df['Day'].unique()))

*# if len(days)<5:*

*# print(f"Skipping : [{m}],Days:{days}")*

*# continue*

print(f"[**{**m**}**]: Days: **{**days**}**")

train\_days = days[0:len(days)-int(len(days)\*p)]

valid\_days = days[len(days)-int(len(days)\*p):len(days)]

*# print(f"[{m}]Train Days: {train\_days}, val days = {valid\_days} ")*

train\_df = month\_df[month\_df['Day'].isin(train\_days)].reset\_index(drop=**True**)

valid\_df = month\_df[month\_df['Day'].isin(valid\_days)].reset\_index(drop=**True**)

**assert** len(train\_df)+len(valid\_df) == len(month\_df)

tarain\_dfs.append(train\_df)

valid\_dfs.append(valid\_df)

t\_df = pd.concat(tarain\_dfs, ignore\_index=**True**).sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**)

v\_df = pd.concat(valid\_dfs, ignore\_index=**True**).sort\_values(by='StartTime', ascending=**True**).reset\_index(drop=**True**)

*# assert len(t\_df)+len(v\_df) == len(df)*

**return** t\_df, v\_df

**def** writeSequeces2TokenFile(self,file\_path,li\_of\_seqs):

**with** open(file\_path,"w") **as** f:

**for** seq **in** li\_of\_seqs:

f.write(f"**{**' '.join(seq)**}\n**")

print("Done")

In [4]:

config = {

'file-path': "../.data/plant\_old\_0/final/all-months-alarms-2.csv",

'usecols':["SourceName", "Condition", "StartTime","EndTime","TimeDelta","Year-Month",'Day'],

'date-cols': ["StartTime", "EndTime"],

'alias': **False**

}

alarm = AlarmsProcessing(config=config)

start = time.time()

print("Total Time to load the data ", time.time()-start)

alarm.df

In [5]:

con2count=getCol2Count(df=alarm.df,col='Condition')

con2count

*#Important Trip, HHH, HTRP, LTRP, LLL, -> most imp*

In [6]:

*# remove communication alarms*

df = alarm.removeConditionsAlarms(df=alarm.df,conditions\_filter=["IOP", "IOP-",'VEL-','VEL+'])

con2count=getCol2Count(df=df,col='Condition')

con2count

In [7]:

df = alarm.removeChatteringAlarms(df=df,chattering\_timedelta\_threshold=60,chattering\_count\_threshold=3)

con2count=getCol2Count(df=df,col='Condition')

con2count

In [8]:

*# df3 = alarm.removeConditionsAlarms(df=alarm.df,conditions\_filter=["IOP", "IOP-",'VEL+','VEL-'])*

*# con2count=getCol2Count(df=df3,col='Condition')*

*# print(con2count)*

*# df4 = removeChatteringAlarms(df=df3,chattering\_timedelta\_threshold=60, chat\_count =2)*

*# con2count=getCol2Count(df=df4,col='Condition')*

*# con2count*

In [9]:

*# total = 0*

*# for cond,count in getCol2Count(df=df,col='Condition').items():*

*# l = df[df['Condition']==cond]["SourceName"].unique()*

*# total += len(l)*

*# print(f">>{cond},{count},{len(l)}")*

*# print(len(getCol2Count(df,col='SourceName')),total)*

In [10]:

df2 = alarm.removeSourcesBasedOnMinCount(df,min\_alarms\_per\_source\_filter=70)

total = 0

**for** cond,count **in** getCol2Count(df=df2,col='Condition').items():

l = df2[df2['Condition']==cond]["SourceName"].unique()

total += len(l)

print(f">>**{**cond**}**,**{**count**}**,**{**len(l)**}**")

print(len(getCol2Count(df2,col='SourceName')),total)

In [11]:

*# temp\_sources = []*

*# for s in df["SourceName"].unique():*

*# # print(len(df2[df2['SourceName']==s]['Condition'].unique()))*

*# if len(df[df['SourceName']==s]['Condition'].unique())==1:*

*# temp\_sources.append(s)*

*# print(temp\_sources)*

In [12]:

FINAL\_DF = df2

alarm\_dataset = PrepareDataset({})

train\_df, valid\_df = alarm\_dataset.splitDFtoTrainValidDfsPerMonthByDays(FINAL\_DF,p=0.3)

valid\_df, test\_df = alarm\_dataset.splitDFtoTrainValidDfsPerMonthByDays(valid\_df,p=0.2)

seqs\_train, \_ = alarm\_dataset.getSeqsFromAlarmsDF(train\_df,seq\_duration\_gap=60\*15,filter\_short\_seq=0)

seqs\_valid, \_ = alarm\_dataset.getSeqsFromAlarmsDF(valid\_df,seq\_duration\_gap=60\*15,filter\_short\_seq=0)

seqs\_test, \_ = alarm\_dataset.getSeqsFromAlarmsDF(test\_df,seq\_duration\_gap=60\*15,filter\_short\_seq=0)

alarm\_dataset.writeSequeces2TokenFile(file\_path="../.data/train.tokens",li\_of\_seqs=seqs\_train)

alarm\_dataset.writeSequeces2TokenFile(file\_path="../.data/val.tokens",li\_of\_seqs=seqs\_valid)

alarm\_dataset.writeSequeces2TokenFile(file\_path="../.data/test.tokens",li\_of\_seqs=seqs\_test)

print("DataSet is complete")

In [13]:

*# duration\_from\_1\_seq\_to\_next = 60\*15 # duration in seconds*

*# filter\_short\_seq = 3 # remove the sequence whose size is less than 4*

*# li\_of\_seqs,max\_seq\_len = getSequenceOfWholeData(df\_rnn,duration\_from\_1\_seq\_to\_next,filter\_short\_seq)*

*# print(len(li\_of\_seqs))*

*# print(li\_of\_seqs[:2])*

In [14]:

*# seq\_len\_2\_count ={}*

*# for seq in li\_of\_seqs:*

*# # seq = removeSameAlarms(seq)*

*# l = len(seq)*

*# seq\_len\_2\_count[l] = 1+seq\_len\_2\_count.get(l,0)*

*# seq\_len\_2\_count = {k:v for k,v in sorted(seq\_len\_2\_count.items(), key=lambda t: t[1] )}*

*# seq\_len\_2\_count*

In [15]:

*# def removeSameAlarms(seq):*

*# new\_seq = []*

*# new\_seq.append(seq[0])*

*# for a in seq:*

*# if a == new\_seq[-1]:*

*# continue*

*# new\_seq.append(a)*

*# return new\_seq*

In [16]:

*# sum(seq\_len\_2\_count.keys())/len(seq\_len\_2\_count.keys())*

*# l = 5\*['5']*