SaniyCheck

October 6, 2020

[19]:
```
# import modules

import numpy as np
import datetime
import seaborn as sns
import datetime
import matplotlib.pyplot as plt
import pandas as pd

import gc

# Constants definition:
CS_Rejected = 1
CS_Entered  = 2
CS_Checked  = 4
CS_Valuated = 16

RS_Idle = 1
RS_NotSelected = 2
RS_SelectedForReview = 4
RS_Reviewed = 8
RS_Bypassed = 16

#RR:RejectionReason
RR_Accepted = 0
RR_RbyMO    = -1

#CIS:ClaimItemStatus
CIS_Accepted = 1
CIS_Rejected = 2
```

1 Introduction

Note: the data preparation step covers several tasks related to - Data concatenation - **Sanity check**: verify if all the data is coherent and exclude incoherencies; - Filling missing values in the dataset:

filling strategies specific to each column - Converting categorical data to numeric: date and text variables - Data normalization

2 Sanity check

The present file is dealing with the sanity check of the concatenated dataset. In this step, we define the excluding conditions, related to several or all of the data fields.

[2]:
```
# read the ftr file with all concatenated data
df_claim_si_concat = pd.read_feather('openIMIS csv/AllData.ftr')

# save the memory and shape dimensions:
memStats_claims = (df_claim_si_concat.memory_usage()/1024/1024).sum()
shape_claims = df_claim_si_concat.shape
```

[3]:
```
# Definition of excluding conditions and add another column with the associated␣
 ↪conditions

# The items that are submitted, but not yet in checked by the Rule Engile are␣
 ↪excluded
# Indeed, only items accepted by the Rule Engine can constitue an input for the␣
 ↪AI
exclusion_cnd1 = df_claim_si_concat['ClaimStatus']==CS_Entered

# Items rejected by the Rule Engine are excluded
exclusion_cnd2 = df_claim_si_concat['RejectionReason']>RR_Accepted

# Missing values in the ClaimAdminId, PolicyID, ProdID, VisiType fields
exclusion_cnd3 = (df_claim_si_concat['ClaimAdminId'].isnull())|\
(df_claim_si_concat['PolicyID'].isnull())|\
(df_claim_si_concat['ProdID'].isnull())|\
(df_claim_si_concat['VisitType'].isnull())

# Incoherence in the status fields are excluded
# - items with ClaimItemStatus == Rejected and RejectionReason == Accepted
# - items with ClaimStatus == Rejected and RejectionReason == Accepted
# - items with ClaimItemStatus == Accepted and RejectionReason == Rejected by␣
 ↪Medical Officer
exclusion_cnd4 = ((df_claim_si_concat['ClaimItemStatus']==CIS_Rejected)&\
                  (df_claim_si_concat['RejectionReason']==RR_Accepted))|\
((df_claim_si_concat['ClaimStatus']==CS_Rejected)&(df_claim_si_concat['RejectionReason']==RR_A
((df_claim_si_concat['ClaimItemStatus']==CIS_Accepted)&(df_claim_si_concat['RejectionReason']=

# Incoherence in the date related fields
# - items with DateFrom before 15-05-2016
# - items with DOB before the DateClaimed
# - items with DateClaimed before 15-05-2016
# - items with DateClaimed before DateFrom
```

```
exclusion_cnd5 = (df_claim_si_concat['DateFrom']<datetime.datetime(2016, 5,␣
 ↪15))|\
(df_claim_si_concat['DOB']>df_claim_si_concat['DateClaimed'])|\
(df_claim_si_concat['DateClaimed']<datetime.datetime(2016, 5, 15))|\
(df_claim_si_concat['DateClaimed']<df_claim_si_concat['DateFrom'])

# Incoherence between status and valuated price
exclusion_cnd6␣
 ↪=(df_claim_si_concat['RejectionReason']==RR_RbyMO)&(df_claim_si_concat['PriceValuated']>0)

# Check if ClaimAdminID has the same HFID as the ClaimHFID:
exclusion_cnd7 = (df_claim_si_concat['HFID']!=df_claim_si_concat['HFId'])
```

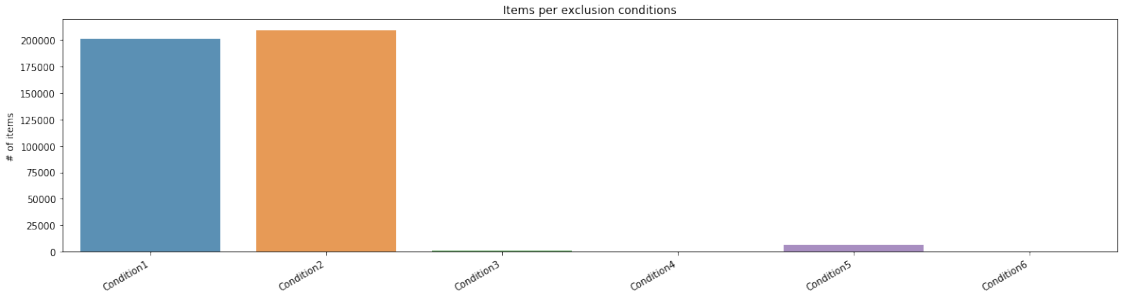
[4]:
```
# Associate these conditions to items in the dataset:

# create a list of our conditions
conditions = [
    (exclusion_cnd1),
    (exclusion_cnd2),
    (exclusion_cnd3),
    (exclusion_cnd4),
    (exclusion_cnd5),
    (exclusion_cnd6),
    (exclusion_cnd7),
    ~(exclusion_cnd1&exclusion_cnd2&exclusion_cnd3&\
      exclusion_cnd4&exclusion_cnd5&exclusion_cnd6&exclusion_cnd7)
    ]

# create a list of the values we want to assign for each condition
values = ['Condition1', 'Condition2','Condition3','Condition4','Condition5',\
          'Condition6','Condition7','Clean data']

# create a new column and use np.select to assign values to it using our lists␣
 ↪as arguments
df_claim_si_concat['SanityCheck']=np.select(conditions, values)

# if necessary, save the curent data:
#df_claim_si_concat.to_pickle('openIMIS csv/AllData_ExclusionConds.pkl')
#df_claim_si_concat.to_csv('openIMIS csv/AllData_ExclusionConds.csv')
```

[5]:
```
df_claim_si_concat['SanityCheck'].value_counts()
```

[5]:
```
Clean data    28553825
Condition2      209210
Condition1      201402
Condition5        6199
Condition3        1451
Condition4         145
```

```
Condition6          33
Name: SanityCheck, dtype: int64
```

[6]:
```
# Visualisation: illustrate the number of items per excluding condition

figs = plt.figure(figsize=(20,5))

df_selection = df_claim_si_concat.loc[df_claim_si_concat['SanityCheck']!='Clean␣
 ↪data']
items_count  = df_selection['SanityCheck'].value_counts().sort_index()
sns.barplot(items_count.index, items_count.values, alpha=0.8)
plt.title("Items per exclusion conditions")
plt.xticks(range(6),['Condition1',␣
 ↪'Condition2','Condition3','Condition4','Condition5','Condition6'])
plt.ylabel("# of items")
plt.gcf().autofmt_xdate()
```



[7]:
```
# Save the clean dataset to another dataframe
df_cleandata = df_claim_si_concat.loc[df_claim_si_concat['SanityCheck']=='Clean␣
 ↪data'].copy()

memStats_cleandata = (df_cleandata.memory_usage()/1024/1024).sum()
shape_cleandata = df_cleandata.shape

# if necessary save the cleaned dataset to a pkl or csv file
#df_cleandata.to_pickle('openIMIS csv/AllData_CleanData.pkl')
#df_cleandata.to_csv('openIMIS csv/AllData_CleanData.csv')

# if neccesary, show information about the columns of the cleaned dataset:
# columns = list(df_cleandata)[:]
# print(f'''The cleandata set is thus composed of {df_cleandata.shape[0]} rows␣
 ↪and {df_cleandata.shape[1]} colums:''')
# for i in columns:
#     print(f''' - the {i} is {df_cleandata.dtypes[i]} and has {df_cleandata[i].
 ↪notnull().sum()} non null values, \
```

```
#     {df_cleandata[i].isnull().sum()} null values, {df_cleandata[i].
 ↪value_counts().count()} distinct values;''')
```

[8]:
```
# delete the full dataset, as it is no longer necessary, il allows to free␣
 ↪memory
del [[df_claim_si_concat, df_selection]]
df_claim_si_concat=pd.DataFrame()
df_selection = pd.DataFrame()
gc.collect()
```

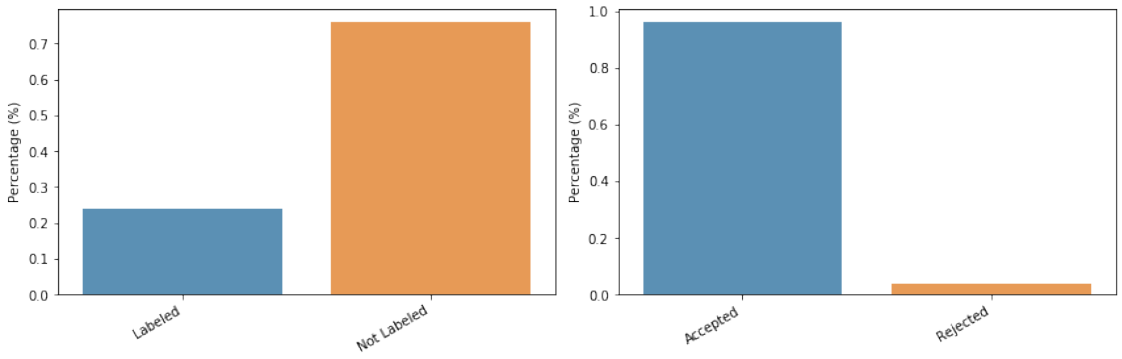
[8]: 27

[9]:
```
# Verify the labeled dataset
label = df_cleandata['ReviewStatus']==RS_Reviewed

conditions = [
    (label),
    ~(label)
    ]

# create a list of the values we want to assign for each condition
values = ['Labeled', 'Not Labeled']

# create a new column
df_cleandata['Label']=np.select(conditions, values)

# saved the labeled data in a new dataframe
df_cleandata_labeled = df_cleandata.loc[df_cleandata['Label']=='Labeled'].copy()

memStats_labeleddata = (df_cleandata_labeled.memory_usage()/1024/1024).sum()
shape_labeleddata = df_cleandata_labeled.shape
```

[10]:
```
# create a list of accepted/rejected
accepted_by_MO = df_cleandata_labeled['RejectionReason']==RR_Accepted

conditions = [
    (accepted_by_MO),
    ~(accepted_by_MO)
    ]

# create a list of the values we want to assign for each condition
values = ['Accepted', 'Rejected']


# create a new column related to 'Class' Accepted/Rejected:
df_cleandata_labeled['Class']=np.select(conditions, values)
```

[11]:
```
# Save data to pickle
df_cleandata_labeled.to_pickle('openIMIS csv/AllData_LabeledData.pkl')
```

[13]:
```
# Visualisation: illustrate the label/not labeled and Accepted/Rejected item␣
 ↪counts
figs = plt.figure(figsize=(12,4))
ax1 = figs.add_subplot(121)
items_count  = df_cleandata['Label'].value_counts(normalize=True).sort_index()
sns.barplot(items_count.index, items_count.values, alpha=0.8)
plt.xticks(range(2),['Labeled', 'Not Labeled'])
plt.ylabel("Percentage (%)")
plt.gcf().autofmt_xdate()


ax2 = figs.add_subplot(122)

items_count  = df_cleandata_labeled['Class'].value_counts(normalize=True).
 ↪sort_index()
sns.barplot(items_count.index, items_count.values, alpha=0.8)
plt.xticks(range(2),['Accepted', 'Rejected'])
plt.ylabel("Percentage (%)")
plt.tight_layout()
plt.gcf().autofmt_xdate()
```



[14]:
```
df_cleandata_notlabeled = df_cleandata.loc[df_cleandata['Label']=='Not␣
 ↪Labeled'].copy()

memStats_notlabeled = (df_cleandata_notlabeled.memory_usage()/1024/1024).sum()
shape_notlabeled = df_cleandata_notlabeled.shape

# df_cleandata_notlabeled.to_pickle('openIMIS csv/AllData_NotLabeledData.pkl')
```

[15]:
```
del [[df_cleandata,df_cleandata_notlabeled]]
df_cleandata=pd.DataFrame()
df_cleandata_notlabeled=pd.DataFrame()
gc.collect()
```

[15]: 3190

3 Field selection

As several columns were necessary only for visualisation reasons or for checking the status of a claim, these columns will be droped and we will continue working with only the selected columns Reasons: - the following columns have an UUID correspondence, and hence these IDs will be droped 'ClaimID', 'ItemID', 'InsureeID', 'HFID', 'ClaimAdminId', 'FamilyID', 'LocationId','HFLocationId' (ClaimUUID', 'ItemUUID', 'InsureeUUID', 'HFUUID', 'ClaimAdminUUID', 'FamilyUUID','LocationUUID','HFLocationUUID') - some IDs are allready checked and no longer necessary 'PolicyID','ProdID', 'FamHeadInsuree','HFId','InsureeHFID','ClaimItemID', - names, necessary only for visualisation, not usefull for AI model 'ItemName','HFLocationName','ICDName','InsureeLocationName', - fields necessary only for checking the valuation of an item: 'ValidityFromReview', 'AuditUserIDReview' - communication fields: 'Explanation','Explanation', 'Justification','ClaimExplanation', 'Adjustment', 'ClaimCode',

[16]:
```
colno = df_cleandata_labeled.shape[1]
droped_cols =␣
 ↪['ClaimItemID','ClaimID','ItemID','InsureeID','HFID','ClaimAdminId','FamilyID',\
'LocationId','HFLocationId',\
'PolicyID','ProdID', 'FamHeadInsuree','HFId','InsureeHFID',\
'ItemName','HFLocationName','ICDName','InsureeLocationName',\
'ValidityFromReview', 'AuditUserIDReview']
df_cleandata_labeled.drop(droped_cols, axis=1, inplace=True)

new_colno = df_cleandata_labeled.shape[1]

df_cleandata_labeled.to_pickle('openIMIS csv/AllData_LabeledData_Selection.pkl')
```

[18]:
```
# Summary:
nb_total = shape_cleandata[0]
nb_labeled = df_cleandata_labeled.shape[0]
nb_accepted = accepted_by_MO.sum()
nb_rejected = nb_labeled - nb_accepted
print(f'''From {nb_total} records in the clean dataset:
- {nb_labeled} are labeled (reviewed by a MO), representing␣
 ↪{round(nb_labeled*100/nb_total,2)} % of all the data
- {nb_total - nb_labeled} are not labeled (accepted without manual reviewing),␣
 ↪representing {round((nb_total - nb_labeled)*100/nb_total,2)} % of all the␣
 ↪data
With respect to the labeled data, we can count:
- {round(nb_accepted*100/nb_labeled,2)} % of labeled records were accepted␣
 ↪({nb_accepted})
- {round(nb_rejected*100/nb_labeled,2)} % of labeled records were rejected␣
 ↪({nb_rejected}).

Droped {colno-new_colno} columns (of {colno}, as no longer necessary). New␣
 ↪dimensions of labeled dataset: \
{nb_labeled,new_colno}
```

```
''')
```

```
From 28553825 records in the clean dataset:
- 6851424 are labeled (reviewed by a MO), representing 23.99 % of all the data
- 21702401 are not labeled (accepted without manual reviewing), representing
76.01 % of all the data
With respect to the labeled data, we can count:
- 96.22 % of labeled records were accepted (6592349)
- 3.78 % of labeled records were rejected (259075).

Droped 20 columns (of 85, as no longer necessary). New dimensions of labeled
dataset: (6851424, 65)
```