tabular_data_06

September 24, 2024

1 call utility functions to get the analysis file

[1]:
```
import pandas as pd
import numpy as np
#
import matplotlib.pyplot as plt
import seaborn as sns
import os

import dhs_util
from dhs_util import *

os.chdir('/Users/yingli/Development/TopicsInDataScience/')
df = pd.read_csv('dhs_service_records_synthesized_final.csv')

df = dhs_preprocessing(df)
df, service_map = add_service_label(df)
df = add_age_bin(df)

recipient = get_recipient_attribute(df)
```

1.1 preparing data for association rule mining

- prepare the transaction, i.e., for each recipient, make a list that contains all the services the recipient used

[2]:
```
serv_list = []
for groups in df.groupby('id').groups.values():
    serv_list.append(df.loc[groups]['serv'].tolist())
```

[3]:
```
for i in range(10):
    print(serv_list[i])
```

```
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12']
['S12', 'S12', 'S12', 'S12']
```

```
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S09', 'S09', 'S09', 'S09', 'S09', 'S09', 'S11', 'S11', 'S11', 'S11', 'S11',
'S11', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12', 'S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12']
['S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11',
'S11', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12', 'S12']
```

[4]:
```
# another way to do the same
serv_list_n = []
for groups in df.groupby('id').groups.values():
    serv_list_n.append(list(df.loc[groups]['serv'].to_numpy()))

# can check equality
serv_list == serv_list_n
```

[4]:
```
True
```

[5]:
```
for i in range(10):
    print(serv_list[i])
```

```
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12']
['S12', 'S12', 'S12', 'S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S09', 'S09', 'S09', 'S09', 'S09', 'S09', 'S11', 'S11', 'S11', 'S11', 'S11',
'S11', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12', 'S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12']
['S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12']
['S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11', 'S11',
'S11', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12', 'S12',
'S12', 'S12']
```

1.1.1 use mlxtend package

- package has full documentation

- license: permissive BSD, allows for usage, even commercially usable

1.1.2 in-class work

- read through doc to install
- read through examples

[6]:
```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.preprocessing import *
from mlxtend.frequent_patterns import association_rules
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import fpmax
from mlxtend.frequent_patterns import hmine
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[6], line 1
----> 1 from mlxtend.preprocessing import TransactionEncoder
      2 from mlxtend.preprocessing import *
      3 from mlxtend.frequent_patterns import association_rules

ModuleNotFoundError: No module named 'mlxtend'
```

[ ]:
```
# re-do the prep of list of services again
serv_list = []
for groups in df.groupby('id').groups.values():
    serv_list.append(df.loc[groups]['serv'].tolist())

# following the tutorial example
def oneHotCoding(serv_list):
    te = TransactionEncoder()
    te_ary = te.fit(serv_list).transform(serv_list)
    te_df = pd.DataFrame(te_ary, columns=te.columns_)
    return te_df

serv_oneHot = oneHotCoding(serv_list)
```

[ ]:
```
serv_oneHot
```

- using groupby([“id”,“serv”]) and then oneHotCoding seem to take very long time
- recall we have done something like this earlier, when we transformed df into a matrix for computing correlations
- that code is cleaned up and put in our dhs_util module as a function “get_id_service_matrix”

[ ]:
```
df_id_serv = get_id_service_matrix(df) # this gives number of times the service␣
↪is used
df_id_serv.iloc[:,1:23] = df_id_serv.iloc[:,1:23] > 0  # this converts value␣
↪into True or False
```

- this was much faster thant the list operation and mlxtend oneHotCoding

[ ]:
```
df_id_serv
```

- one difference is that this dataframe has the "id" as column, not index
- we could turn it into an index or just use the other columns

[ ]:
```
apriori(df_id_serv.iloc[:,1:23], use_colnames=True, min_support=0.01)\
    .sort_values(by="support", ascending=False)
```

- compare

[ ]:
```
apriori(serv_oneHot, min_support=0.01, use_colnames=True)\
    .sort_values(by="support", ascending=False)
```

[ ]:
```
min_freq = 1000 # if we want to set threshold by frequency of the itemsets
min_support = min_freq/serv_oneHot.shape[0]
min_confidence = 0.6
min_rule_support = 0.2
min_lift = 0.15
```

[ ]:
```
apriori(serv_oneHot, min_support=min_support,use_colnames=True)\
    .sort_values(by="support", ascending=False)
```

[ ]:
```
freq_itemset_apriori =␣
↪apriori(serv_oneHot,min_support=min_support,use_colnames=True)
freq_itemset_apriori.describe()
```

[ ]:
```
freq_itemset_fpgrowth =␣
↪fpgrowth(serv_oneHot,min_support=min_support,use_colnames=True)
freq_itemset_fpgrowth.describe()
```

[ ]:
```
freq_itemset_fpmax =␣
↪fpmax(serv_oneHot,min_support=min_support,use_colnames=True)
freq_itemset_fpmax.describe()
```

[ ]:
```
# compute and print the association rules
def serv_rules(freq_itemsets,metrics,threshold):
    asso_rules = association_rules(freq_itemsets, metric=metrics,␣
↪min_threshold=threshold)
    return asso_rules.sort_values(by='lift', ascending=False)[['antecedents',␣
↪'consequents', 'support', 'confidence', 'lift']]
```

```
rule_apriori = serv_rules(freq_itemset_apriori,"confidence",0.60)
rule_fpgrowth = serv_rules(freq_itemset_fpgrowth,"confidence",0.60)
```

[ ]:
```
rule_fpgrowth
```

[ ]:
```
hmine(serv_oneHot,min_support=0.0001,use_colnames=True)
```

[ ]:
```
fpmax(serv_oneHot,min_support=0.0001,use_colnames=True)
```

[ ]:
```
freq_itemset_fpgrowth = fpmax(serv_oneHot,min_support=0.0001,use_colnames=True)
asso_rules = association_rules(freq_itemset_fpgrowth, metric="support",␣
 ↪min_threshold=0.0003,support_only=True)
asso_rules.sort_values(by='lift', ascending=False)[['antecedents',␣
 ↪'consequents', 'support', 'confidence', 'lift']]
```

[ ]:
```
def predict(antecedent, rules, consequents_only = False):
    # get the rules for this antecedent
    preds = rules[rules['antecedents'] == antecedent]
    if consequents_only:
        # a way to convert a frozen set with one element to string
        preds = preds['consequents'].apply(iter).apply(next)
    return preds
```

[ ]:
```
rule_fpmax = association_rules(freq_itemset_fpmax, metric="confidence",␣
 ↪min_threshold=0.001, support_only=True)
```

[ ]:
```
predict({"S06"}, rule_fpmax, consequents_only=False)
```

[ ]:
```
predict({"S09"}, rule_fpmax, consequents_only=False)
```

[ ]:
```
predict({"S09"}, rule_fpgrowth)
```

[ ]:
```
serv_list =  ['S'+str(i).zfill(2) for i in range(1,23)]
for i in serv_list:
    print(i)
    if (len(predict({i},rule_fpgrowth))>0):
        print(i), print(predict({i},rule_fpgrowth))
```

[ ]:
```
rule_fpgrowth.info()
```

[ ]:
```
rule_fpgrowth.sort_values("support",ascending=False)
```

[ ]:
```
df[(df.serv=="S09")].merge(df[(df.serv =="S12")], on = "id").id.nunique()/df.id.
 ↪nunique()
```

[ ]:
```
predict({"S11", "S18"}, rule_fpgrowth)
```

[ ]:
```

```