数据挖掘互评作业二: 频繁模式与关联规则挖掘

github 地址:
https://github.com/zx2308884687/DataMining/tree/HomeWork/%E4%BA%92%E8%AF%84%E4%BD%9C%E4%
(https://github.com/zx2308884687/DataMining/tree/HomeWork/%E4%BA%92%E8%AF%84%E4%BD%9C%E4%

In [40]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

1. 导入数据集并进行初步处理 使用的数据集为Wine Reviews

1.1 导入数据集

In [41]:

```
df1 = pd.read_csv('./Wine Reviews/winemag-data_first150k.csv')
df2 = pd.read_csv('./Wine Reviews/winemag-data-130k-v2.csv')
```

1.2 合并数据集

In [42]:

```
df1.drop(['country', 'description', 'designation'], axis=1, inplace=True)
df1.drop(df1.columns[0], axis=1, inplace=True)
df1.info()
df2.drop(['country', 'description', 'designation', 'taster_name', 'taster_twitter_handle', 'title'],
df2.drop(df2.columns[0], axis=1, inplace=True)
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150930 entries, 0 to 150929
Data columns (total 7 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   points    150930 non-null  int64  
 1   price     137235 non-null  float64
 2   province  150925 non-null  object 
 3   region_1  125870 non-null  object 
 4   region_2  60953 non-null   object 
 5   variety   150930 non-null  object 
 6   winery    150930 non-null  object 
dtypes: float64(1), int64(1), object(5)
memory usage: 8.1+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129971 entries, 0 to 129970
Data columns (total 7 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   points    129971 non-null  int64  
 1   price     120975 non-null  float64
 2   province  129908 non-null  object 
 3   region_1  108724 non-null  object 
 4   region_2  50511 non-null   object 
 5   variety   129970 non-null  object 
 6   winery    129971 non-null  object 
dtypes: float64(1), int64(1), object(5)
memory usage: 6.9+ MB
```

In [43]:

```
df = pd.concat([df1,df2],ignore_index=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 280901 entries, 0 to 280900
Data columns (total 7 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   points    280901 non-null  int64  
 1   price     258210 non-null  float64
 2   province  280833 non-null  object 
 3   region_1  234594 non-null  object 
 4   region_2  111464 non-null  object 
 5   variety   280900 non-null  object 
 6   winery    280901 non-null  object 
dtypes: float64(1), int64(1), object(5)
memory usage: 15.0+ MB
```

1.3 删除缺失值并重置索引

In [44]:

```
df = df.dropna(axis=0)
df.index = range(len(df))
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110996 entries, 0 to 110995
Data columns (total 7 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   points    110996 non-null  int64
 1   price     110996 non-null  float64
 2   province  110996 non-null  object
 3   region_1  110996 non-null  object
 4   region_2  110996 non-null  object
 5   variety   110996 non-null  object
 6   winery    110996 non-null  object
dtypes: float64(1), int64(1), object(5)
memory usage: 5.9+ MB
```

查看缺失值的数量，确保数据集中已经没有缺失值

In [45]:

```
df.isnull().sum()
```

Out[45]:

```
points      0
price       0
province    0
region_1    0
region_2    0
variety     0
winery      0
dtype: int64
```

2. 找出频繁模式

2.1 将数据集转换为指定格式

In [46]:

```
transactions = []
for i in range(1,df.iloc[:,0].size):  #行数
    line = []
    line.append("points" + '='+ str(df.loc[i,'points']))
    if(0<=df.loc[i,'price']<50):
        line.append("price" + '='+ 'price_0_50')
    elif(50<=df.loc[i,'price']<100):
        line.append("price" + '='+ 'price_50_100')
    elif(100<=df.loc[i,'price']<150):
        line.append("price" + '='+ 'price_100_150')
    else:
        line.append("price" + '='+ 'price_150')
    line.append("province" + '='+ str(df.loc[i,'province']))
    line.append("region_1" + '='+ str(df.loc[i,'region_1']))
    line.append("region_2" + '='+ str(df.loc[i,'region_2']))
    line.append("variety" + '='+ str(df.loc[i,'variety']))
    line.append("winery" + '='+ str(df.loc[i,'winery']))
    transactions.append(line)
```

2.2 输出频繁项集

In [59]:

```
# from efficient_apriori import apriori
# itemsets,rules = apriori(transactions, min_support=0.5, min_confidence=1)
# print(itemsets)
```

In [48]:

```
def createC1(dataSet):  # 产生单个item的集合
    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])

    C1.sort()
    return map(frozenset, C1)  # 给C1.list每个元素执行函数


def scanD(D, ck, minSupport):  # dataset,a list of candidate set,最小支持率 支持度计数

    ssCnt = {}
    # temp_D = list(D)
    numItem = float(len(D))
    temp_ck = list(ck)
    for tid in D:
        for can in temp_ck:
            if can.issubset(tid):
                if can not in ssCnt:
                    ssCnt[can] = 1
                else:
                    ssCnt[can] += 1

    retList = []
    supportData = {}
    for key in ssCnt:
        if numItem == 0:
            continue
        support = ssCnt[key] / numItem
        if support >= minSupport:
            retList.insert(0, key)
            supportData[key] = support
    return retList, supportData  # 返回频繁k项集，相应支持度


def aprioriGen(Lk, k):  # create ck(k项集)
    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i + 1, lenLk):
            L1 = list(Lk[i])[:k - 2]
            L2 = list(Lk[j])[:k - 2]
            L1.sort()
            L2.sort()  # 排序
            if L1 == L2:  # 比较i,j前k-1个项若相同，和合并它俩
                retList.append(Lk[i] | Lk[j])  # 加入新的k项集 | stanf for union
    return retList # ck


def apriori(dataSet, minSupport=0.5):
    C1 = createC1(dataSet) # c1 = return map
    # D = map(set, dataSet) # D = map
    D = dataSet
    L1, supportData = scanD(D, C1, minSupport)  # 利用k项集生成频繁k项集（即满足最小支持率的k项集）
    itemsets = [L1]  # itemsets保存所有频繁项集

    k = 2
```

```
    while (len(itemsets[k - 2]) > 0):  # 直到频繁k-1项集为空
        Ck = aprioriGen(itemsets[k - 2], k)  # 利用频繁k-1项集 生成k项集
        Lk, supK = scanD(D, Ck, minSupport)
        supportData.update(supK)  # 保存新的频繁项集与其支持度
        itemsets.append(Lk)  # 保存频繁k项集
        k += 1
    return itemsets, supportData  # 返回所有频繁项集，与其相应的支持率
```

In [49]:

```
itemsets, supdata = apriori(transactions)
print(itemsets)
print(supdata)
```

```
[[frozenset({'price=price_0_50'}), frozenset({'province=California'})], [frozenset
({'price=price_0_50', 'province=California'})], []]
{frozenset({'province=California'}): 0.7036352988873372, frozenset({'price=price_0_5
0'}): 0.8116671922158656, frozenset({'price=price_0_50', 'province=California'}): 0.
5541871255461958}
```

3. 输出关联规则

In [54]:

```
def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    prunedH = []
    lift = []
    file = open("generate_rules.txt","a",encoding = "utf-8")
    for conseq in H:  # 后件中的每个元素
        conf = supportData[freqSet] / supportData[freqSet - conseq]
        if conf >= minConf:
            file.write(str(freqSet - conseq)+"-->"+str(conseq)+" support:"+str(supportData[freqSet]
            brl.append((freqSet - conseq, conseq, supportData[freqSet], conf))  # 添加入规则集中
            prunedH.append(conseq)  # 添加入被修剪过的H中
    file.close()
    return prunedH


def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    m = len(H[0])  # H是一系列后件长度相同的规则，所以取H0的长度即可
    if (len(freqSet) > m + 1):
        Hmp1 = aprioriGen(H, m + 1)
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        if (len(Hmp1) > 1):
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)


def generateRules(L, supportData, minConf=0.7):
    bigRuleList = []  # 存储规则
    for i in range(1, len(L)):
        for freqSet in L[i]:
            H1 = [frozenset([item]) for item in freqSet]
            if (i > 1):
                rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList
```

In [55]:

```
rules = generateRules(itemsets, supdata, minConf=0.5)
print(rules)
```

```
[(frozenset({'province=California'}), frozenset({'price=price_0_50'}), 0.55418712554
61958, 0.7876056338028169), (frozenset({'price=price_0_50'}), frozenset({'province=C
alifornia'}), 0.5541871255461958, 0.68277630395933)]
```

4. 使用lift进行评估

In [56]:

```
def lift_eval(rules, suppData): # lift evaluation
    # lift(A, B) = P(A交B) / (P(A) * P(B)) = P(A) * P(B | A) / (P(A) * P(B)) = P(B | A) / P(B) = con
    lift = []
    for rule in rules:
        freqSet_conseq = rule[0]
        conseq = rule[1]
        lift_val = float(rule[3]) / float(suppData[rule[1]])
        lift.append([freqSet_conseq, conseq, lift_val])
    return lift
```

In [57]:

```
lifts = lift_eval(rules, supdata)
print(lifts)
```

```
[[frozenset({'province=California'}), frozenset({'price=price_0_50'}), 0.97035538870
63487], [frozenset({'price=price_0_50'}), frozenset({'province=California'}), 0.9703
553887063487]]
```

5. 使用卡方进行评估

In [ ]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif
x, y = transactions.data, transactions.target
# result=mutual_info_classif(x,y,random_state=666)
#mutual_info_classif是有一定的随机性的
result=mutual_info_classif(x,y)
#返回每个特征与标签的互信息估计量
result
#筛选出来互 信息量估计量 最大的前2个特征
x_new = SelectKBest(mutual_info_classif, k=2).fit_transform(x, y)
print(x_new)
```

6.结果分析

经过预处理后，保留的葡萄酒数据均为us的数据。

从频繁项集和关联规则的结果可以看出，几乎所有的葡萄酒价格都在0到50之间，并且大多数的葡萄酒都是来自California省。

7.可视化 可视化部分包含在代码中