数据挖掘大作业二报告

一、 问题描述：

本次作业要求从作业一的两个数据集中任选一个进行分析，关联规则挖掘任务包含以下五个子任务：

- 对数据集进行处理，转换成适合关联规则挖掘的形式；
- 找出频繁项集；
- 导出关联规则，计算其支持度和置信度；
- 对规则进行评价，可使用 Lift 及其它指标，要求至少 2 种；
- 对挖掘结果进行可视化展示。

二、 数据说明：

本次作业我将选择作业一中的 Wine Reviews 数据集，详细信息如下：

◆ **数据集：Wine Reviews**

文件名	数据记录个数	属性条数
winemag-data_first150k.csv	150930	11
winemag-data-130k-v2.csv	129971	14

其中 **winemag-data_first150k.csv** 文件包含八个标称属性，分别是'country'，'description'，'designation'，'province'，'region_1'，'region_2'，'variety'，'winery'；**winemag-data-130k-v2.csv** 文件包含十一个标称属性，分别是'country', 'description', 'designation', 'province', 'region_1', 'region_2', 'taster_name'，'taster_twitter_handle'，'title'，'variety'，'winery'。

三、 数据分析过程：

3.1 将数据集转换成适合关联规则挖掘的形式

考虑到问题的复杂度和运算速度，且数据集中存在大量的数据缺失情况，因此在 process_data.py 文件中，我将数据集中的标称属性提取出来，并删除包含空值的数据，最后将数据存到./result/[数据集名]/[[文件名]/processd.csv”下。

```
for file_name in self.data_file_list:
    content = pd.read_csv(os.path.join(self.read_data, file_name))
    write_data_path = os.path.join(self.write_data, file_name.split('.')[0])
    makedir(write_data_path)
    print("-------------------------------------------------------------------------------------------------------------")
    print("Begin to process file: %s" % file_name)
    for title in content.columns.values:
        if content[title].dtypes == "int64" or content[title].dtypes == "float64":
            content = content.drop(columns = [title])
    content = content.dropna()
    with open(os.path.join(write_data_path, 'processd.csv'), 'w', encoding = 'utf-8') as fp:
        content.to_csv(fp)
```

3.2 找出频繁项集

从 processd.csv 文件读取出数据后，我们需要对多个属性进行关联规则挖掘，需要将来自于不同属性的值转化为可生成频繁项集的形式。首先我将每一个属性名和属性值的组合用一个二元组的形式表示，既（属性名，属性值），为一个单项。使用 python 中的 frozenset 类型表示项集。

在经过处理的数据集的基础上，采用 Apriori 算法构建频繁项集。在此任务中，频繁项集是指经常出现在一起的属性项的集合，而一个项集的支持度（support）定义为数据集中包含该项集的记录所占的比例。首先，规定最小支持度（min-support）为 0.25，最小置信度（min-confidence）为 0.5。

Apriori 算法首先会生成所有单个（属性名，属性值）的项集列表，然后扫描全部数据集来查看哪些项集满足最小支持度要求，其中不满足最小支持度的集合会被去掉，接着对剩下的集合进行组合（组合中，要求属性名相同的项仅有一个）以生成包含两个项的项集，接着重新扫描交易记录，去掉不满足最小支持度的项集，该过程重复进行直到所有项集都被滤掉。

在 Data 类的 association()函数中调用 apriori()函数获取频繁项集，apriori()函数代码如下：

- apriori 主函数：

```
def apriori(self, dataset):
    C1 = self.create_C1(dataset)
    dataset = [set(data) for data in dataset]
    L1, support_data = self.scan_D(dataset, C1)
    L = [L1]
    k = 2
    while len(L[k-2]) > 0:
        Ck = self.apriori_gen(L[k-2], k)
        Lk, support_k = self.scan_D(dataset, Ck)
        support_data.update(support_k)
        L.append(Lk)
        k += 1
    return L, support_data
```

- create_C1()函数用于生成初始的单个项项集集合：

```
def create_C1(self, dataset):
    # 扫描dataset，构建全部可能的单元素候选项集合(list)
    # 每个单元素候选项：（属性名，属性取值）
    C1 = []
    for data in dataset:
        for item in data:
            if [item] not in C1:
                C1.append([item])
    C1.sort()
    return [frozenset(item) for item in C1]
```

- scan_D()函数用于扫描项集集合，并过滤掉小于最小支持度的项集：

```
def scan_D(self, dataset, Ck):
    # 过滤函数
    # 根据待选项集Ck的情况，判断数据集D中Ck元素的出现频率
    # 过滤掉低于最小支持度的项集
    Ck_count = dict()
    for data in dataset:
        for cand in Ck:
            if cand.issubset(data):
                if cand not in Ck_count:
                    Ck_count[cand] = 1
                else:
                    Ck_count[cand] += 1

    num_items = float(len(dataset))
    return_list = []
    support_data = dict()
    # 过滤非频繁项集
    for key in Ck_count:
        support  = Ck_count[key] / num_items
        if support >= min_support:
            return_list.insert(0, key)
        support_data[key] = support
    return return_list, support_data
```

◆ apriori_gen()函数用于非重复地合并两个项集:

```
def apriori_gen(self, Lk, k):
    # 当待选项集不是单个元素时， 如k>=2的情况下，合并元素时容易出现重复
    # 因此针对包含k个元素的频繁项集，对比每个频繁项集第k-2位是否一致
    return_list = []
    len_Lk = len(Lk)

    for i in range(len_Lk):
        for j in range(i+1, len_Lk):
            # 第k-2个项相同时，将两个集合合并
            L1 = list(Lk[i])[:k-2]
            L2 = list(Lk[j])[:k-2]
            L1.sort()
            L2.sort()
            if L1 == L2:
                return_list.append(Lk[i] | Lk[j])
    return return_list
```

◆ 将频繁项集输出到结果文件

```
data_set = self.get_data_set(content)
# 获取频繁项集
freq_set, support_data = self.apriori(data_set)
support_data_out = sorted(support_data.items(), key= lambda d:d[1],reverse=True)
# 将频繁项集输出到结果文件
freq_set_file = open(os.path.join(write_data_path, 'freq_set.json'), 'w')
for (key, value) in support_data_out:
    result_dict = {'set':None, 'sup':None}
    set_result = list(key)
    sup_result = value
    result_dict['set'] = set_result
    result_dict['sup'] = sup_result
    json_str = json.dumps(result_dict, ensure_ascii=False)
    freq_set_file.write(json_str+'\n')
freq_set_file.close()
```

最后将数据存到./result/[数据集名]/[[文件名]/ freq_set.json 下。

3.3 导出关联规则并计算支持度、置信度、Lift 指标

基于使用 Apriori 算法产生的频繁项集，产生强关联规则的算法过程为：首先从一个频繁项集开始，创建一个规则列表，其中规则右部只包含一个元素，然后对这些规则计算是否满足最小置信度要求。接下来合并所有的剩余规则列表来创建一个新的规则列表，其中规则右部包含两个元素。最后，对于产生的每个规则，我们分别计算其支持度(support)、置信度(confidence)以及提升度(Lift)指标。计算公式如下：

◆ 支持度（support）

$$\mathrm{Sup(X)} = \frac{Sum(X)}{N}$$

◆ 置信度(confidence)

$$\mathrm{Conf(X \Rightarrow Y)} = \frac{Sup(X \cup Y)}{Sup(X)}$$

◆ 提升度(Lift)

用来判断规则 X⇒Y 中的 X 和 Y 是否独立，如果独立，那么这个规则是无效的。如果该值等于 1 ,说明两个条件没有任何关联。如果小于 1,说明 X 与 Y 是负相关的关系，意味着一个出现可能导致另外一个不出现。大于 1 才表示具有正相关的关系。一般在数据挖掘中当提升度大于 3 时,我们才承认挖掘出的关联规则是有价值的。

$$\mathrm{Lift(X \Rightarrow Y)} = \frac{Sup(X \cup Y)}{Sup(X) \times Sup(Y)} = \frac{\mathrm{Conf(X \cup Y)}}{Sup(Y)}$$

在 Data 类的 association()函数中调用 generate_rules ()函数来获取强关联规则列表、支持度、置信度和 Lift 指标等值，generate_rules()函数代码如下:

◆ generate_rules()函数，用于产生强关联规则:

```
def generate_rules(self, L, support_data):
    """
    产生强关联规则算法实现
    基于Apriori算法，首先从一个频繁项集开始，接着创建一个规则列表，
    其中规则右部只包含一个元素，然后对这些规则进行测试。
    接下来合并所有的剩余规则列表来创建一个新的规则列表，
    其中规则右部包含两个元素。这种方法称作分级法。
    :param L: 频繁项集
    :param support_data: 频繁项集对应的支持度
    :return: 强关联规则列表
    """
    big_rules_list = []
    for i in range(1, len(L)):
        for freq_set in L[i]:
            H1 = [frozenset([item]) for item in freq_set]
            # 只获取有两个或更多元素的集合
            if i > 1:
                self.rules_from_conseq(freq_set, H1, support_data, big_rules_list)
            else:
                self.cal_conf(freq_set, H1, support_data, big_rules_list)
    return big_rules_list
```

◆ rules_from_conseq()函数，用于递归地产生规则右部的结果项集：

```
def rules_from_conseq(self, freq_set, H, support_data, big_rules_list):
    # H->出现在规则右部的元素列表
    m = len(H[0])
    if len(freq_set) > (m+1):
        Hmp1 = self.apriori_gen(H, m+1)
        Hmp1 = self.cal_conf(freq_set, Hmp1, support_data, big_rules_list)
        if len(Hmp1) > 1:
            self.rules_from_conseq(freq_set, Hmp1, support_data, big_rules_list)
```

◆ cal_conf()函数，用于评价生成的规则，并计算支持度、置信度、lift 指标：

```
def cal_conf(self, freq_set, H, support_data, big_rules_list):
    # 评估生成的规则
    prunedH = []
    for conseq in H:
        sup = support_data[freq_set]
        conf = sup / support_data[freq_set - conseq]
        lift = conf / support_data[freq_set - conseq]
        if conf >= min_confidence:
            big_rules_list.append((freq_set-conseq, conseq, sup, conf, lift))
            prunedH.append(conseq)
    return prunedH
```

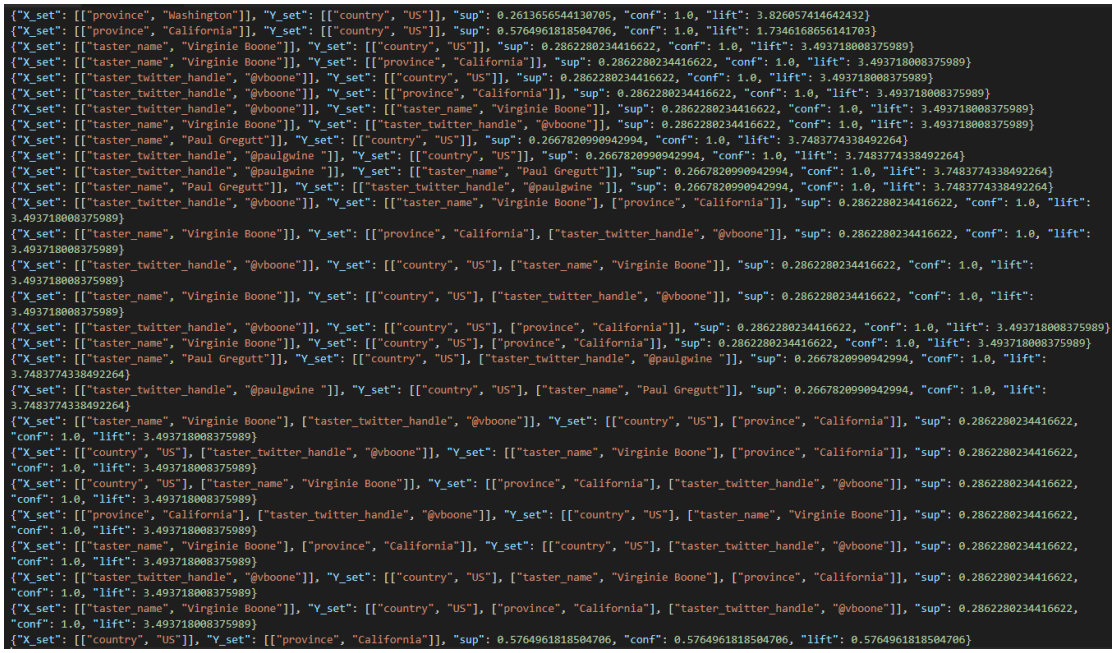
◆ 将关联规则输出到结果文件

```
# 获取强关联规则列表
big_rules_list = self.generate_rules(freq_set, support_data)
big_rules_list = sorted(big_rules_list, key= lambda x:x[3], reverse=True)
# 将关联规则输出到结果文件
rules_file = open(os.path.join(write_data_path, 'rules.json'), 'w')
for result in big_rules_list:
    result_dict = {'X_set':None, 'Y_set':None, 'sup':None, 'conf':None, 'lift':None}
    X_set, Y_set, sup, conf, lift = result
    result_dict['X_set'] = list(X_set)
    result_dict['Y_set'] = list(Y_set)
    result_dict['sup'] = sup
    result_dict['conf'] = conf
    result_dict['lift'] = lift
    json_str = json.dumps(result_dict, ensure_ascii=False)
    rules_file.write(json_str + '\n')
rules_file.close()
```

最后将数据存到./result/[数据集名]/[[文件名]/ rules .json 下。

3.4 对规则进行可视化及分析

以 winemag-data-130k-v2.csv 文件为例，其关联结果位于"./result/wine-reviews/winemag-data_first150k/rules .json"下，文件每行描述了一个关联规则，且所有关联规则按照置信度由大到小排列，可视化结果如下：



```
{"X_set": [["province", "Washington"]], "Y_set": [["country", "US"]], "sup": 0.2613656544130705, "conf": 1.0, "lift": 3.826057414642432}
{"X_set": [["province", "California"]], "Y_set": [["country", "US"]], "sup": 0.5764961818504706, "conf": 1.0, "lift": 1.7346168656141703}
{"X_set": [["taster_name", "Virginie Boone"]], "Y_set": [["country", "US"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_name", "Virginie Boone"]], "Y_set": [["province", "California"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_twitter_handle", "@vboone"]], "Y_set": [["country", "US"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_twitter_handle", "@vboone"]], "Y_set": [["province", "California"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_twitter_handle", "@vboone"]], "Y_set": [["taster_name", "Virginie Boone"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_name", "Virginie Boone"]], "Y_set": [["taster_twitter_handle", "@vboone"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_name", "Paul Gregutt"]], "Y_set": [["country", "US"]], "sup": 0.2667820990942994, "conf": 1.0, "lift": 3.7483774338492264}
{"X_set": [["taster_twitter_handle", "@paulgwine "]], "Y_set": [["country", "US"]], "sup": 0.2667820990942994, "conf": 1.0, "lift": 3.7483774338492264}
{"X_set": [["taster_twitter_handle", "@paulgwine "]], "Y_set": [["taster_name", "Paul Gregutt"]], "sup": 0.2667820990942994, "conf": 1.0, "lift": 3.7483774338492264}
{"X_set": [["taster_name", "Paul Gregutt"]], "Y_set": [["taster_twitter_handle", "@paulgwine "]], "sup": 0.2667820990942994, "conf": 1.0, "lift": 3.7483774338492264}
{"X_set": [["taster_twitter_handle", "@vboone"]], "Y_set": [["taster_name", "Virginie Boone"], ["province", "California"]], "sup": 0.2862280234416622, "conf": 1.0, "lift":
3.493718008375989}
{"X_set": [["taster_name", "Virginie Boone"]], "Y_set": [["province", "California"], ["taster_twitter_handle", "@vboone"]], "sup": 0.2862280234416622, "conf": 1.0, "lift":
3.493718008375989}
{"X_set": [["taster_twitter_handle", "@vboone"]], "Y_set": [["country", "US"], ["taster_name", "Virginie Boone"]], "sup": 0.2862280234416622, "conf": 1.0, "lift":
3.493718008375989}
{"X_set": [["taster_name", "Virginie Boone"]], "Y_set": [["country", "US"], ["taster_twitter_handle", "@vboone"]], "sup": 0.2862280234416622, "conf": 1.0, "lift":
3.493718008375989}
{"X_set": [["taster_twitter_handle", "@vboone"]], "Y_set": [["country", "US"], ["province", "California"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_name", "Virginie Boone"]], "Y_set": [["country", "US"], ["province", "California"]], "sup": 0.2862280234416622, "conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_name", "Paul Gregutt"]], "Y_set": [["country", "US"], ["taster_twitter_handle", "@paulgwine "]], "sup": 0.2667820990942994, "conf": 1.0, "lift":
3.7483774338492264}
{"X_set": [["taster_twitter_handle", "@paulgwine "]], "Y_set": [["country", "US"], ["taster_name", "Paul Gregutt"]], "sup": 0.2667820990942994, "conf": 1.0, "lift":
3.7483774338492264}
{"X_set": [["taster_name", "Virginie Boone"], ["taster_twitter_handle", "@vboone"]], "Y_set": [["country", "US"], ["province", "California"]], "sup": 0.2862280234416622,
"conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["country", "US"], ["taster_twitter_handle", "@vboone"]], "Y_set": [["taster_name", "Virginie Boone"], ["province", "California"]], "sup": 0.2862280234416622,
"conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["country", "US"], ["taster_name", "Virginie Boone"]], "Y_set": [["province", "California"], ["taster_twitter_handle", "@vboone"]], "sup": 0.2862280234416622,
"conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["province", "California"], ["taster_twitter_handle", "@vboone"]], "Y_set": [["country", "US"], ["taster_name", "Virginie Boone"]], "sup": 0.2862280234416622,
"conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_name", "Virginie Boone"], ["province", "California"]], "Y_set": [["country", "US"], ["taster_twitter_handle", "@vboone"]], "sup": 0.2862280234416622,
"conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_twitter_handle", "@vboone"]], "Y_set": [["country", "US"], ["taster_name", "Virginie Boone"], ["province", "California"]], "sup": 0.2862280234416622,
"conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["taster_name", "Virginie Boone"]], "Y_set": [["country", "US"], ["province", "California"], ["taster_twitter_handle", "@vboone"]], "sup": 0.2862280234416622,
"conf": 1.0, "lift": 3.493718008375989}
{"X_set": [["country", "US"]], "Y_set": [["province", "California"]], "sup": 0.5764961818504706, "conf": 0.5764961818504706, "lift": 0.5764961818504706}
```


本次实验通过 Apriori 算法按照 support, confidence, LIFT 等指标计算出了频繁项集和他们之间的关系，对最终挖掘得到的关联规则进行分析我们可以得知，”province”属性和”country”属性关联度极高，且“province –> country”规则的置信度极高，与事实相符；还有“taster_twitter_handle”属性和“taster_name“也有较大的关联度，及品酒者与其拥有的 twitter 账号名对应。

此外，我还挖掘出酒的产地和品酒人之间的一些有趣的关系，品酒者“Virginie Boone“和”Paul Gregutt“格外喜欢在”US“国家的”California“州产的酒。