



# 本科实验报告

课程名称： 数据挖掘

课程编号： 08060116

学生姓名： 邝庆璇

学号： 2016051598

学院： 信息科学与技术学院

系： 计算机系

专业： 计算机科学与技术

指导教师： 刘波

教师单位： 暨南大学计算机系

开课时间： 2018~2019 学年度 第 2 期

暨南大学教务处

2019 年 06 月 16 日

## 数据挖掘 课程实验项目目录

学生姓名：邝庆璇 学号：2016051598

序号	实验项目编号	实验项目名称	*实验项目 类型	成绩	指导教师
1	0806011601	关联分析	综合		刘波
2	0806011602	分类	综合		刘波
3	0806011603	聚类	综合		刘波

\*实验项目类型：演示性、验证性、验证性、设计性实验。

\*此表由学生按顺序填写。

# 暨南大学本科实验报告专用纸

课程名称 数据挖掘 成绩评定 \_\_\_\_\_  
实验名称 实验一 关联分析 指导教师 刘波  
试验编号 0806011601 类型 综合性  
学生姓名 邝庆璇 学号 2016051598  
学院 信息科学技术学院 系 计算机 专业 计算机科学与技术

## 1、实验目的：

- (1) 理解关联分析原理
- (2) 理解置信度、支持度等概念
- (3) 掌握 Apriori 算法

## 2、实验内容

(0) 对航空公司的乘客数据进行预处理（清洗、特征筛选），并得到 5 个关键的指标：客户关系时长 L、消费时间间隔 R、消费频率 F、飞行里程 M 和折扣系数的平均值 C，并进行归一化。

- (1) 用 Apriori 算法，对航空信息的这 5 个指标进行关联分析；
- (2) 输出并比较设定不同置信度下的实验结果。

## 3、实验原理

Apriori 算法的思路如书上 P94 页所言，即按照一定的置信度和支持度，每次迭代挑选 k 频繁项集。每次生成 k 频繁项集的方法：把只差 1 项的两个 k-1 频繁项集连接在一起，检查连接后的 k 项集的每个候选子集是不是都是频繁的；若是，则把该 k 项集归为 k 频繁项集。

### • 两条依据：

- 1) 若集合存在某一子集不是频繁项集，则该集合本身也不是频繁的。
- 2) 若集合本身是频繁的，则其任意非空子集都是频繁的。

### • 算法如下：

输入：数据集合 D，支持度阈值  $\alpha$

输出：最大的频繁 k 项集

- 1) 扫描整个数据集，得到所有出现过的数据，作为候选频繁 1 项集。k=1，

频繁 0 项集为空集。

## 2) 挖掘频繁 k 项集

a) 扫描数据计算候选频繁 k 项集的支持度

b) 去除候选频繁 k 项集中支持度低于阈值的数据集,得到频繁 k 项集。如果得到的频繁 k 项集为空, 则直接返回频繁 k-1 项集的集合作为算法结果, 算法结束。如果得到的频繁 k 项集只有一项, 则直接返回频繁 k 项集的集合作为算法结果, 算法结束。

c) 基于频繁 k 项集, 连接生成候选频繁 k+1 项集。

3) 令  $k=k+1$ , 转入步骤 2。

## 3、实验过程:

### 步骤 0. 预处理数据集

航空旅客的信息的原始数据集及其含义如下:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
1	MEMBER_ID	FFP_DATE	FIRST_FLIGENDER	FFP_TIER	WORK_CITY	WORK_PROV	WORK_COUN	AGE	LOAD_TIME	FLIGHT_CO	BP_SUM	EP_SUM	YR_SUM	YR_SUM_1	SUM_YR_2	SEG_KM	SUWEIGHTED	LAST_FLIG	AVG_FLIGH		
2	54993	2006/11/2	##### 男	6	.	北京	CN		31	2014/3/31	210	505308	0	74460	239560	234188	580717	558440.1	2014/3/31	26.25	
3	28065	2007/2/19	2007/8/3 男	6	.	北京	CN		42	2014/3/31	140	362480	0	41288	171483	167434	293678	367777.2	2014/3/25	17.5	
4	55106	2007/2/1	2007/8/30 男	6	.	北京	CN		40	2014/3/31	135	351159	0	39711	163618	164982	283712	355966.5	2014/3/21	16.875	
5	21189	2008/8/22	2008/8/23 男	5	Los Angeles	CA	US		64	2014/3/31	23	337314	0	34890	116350	125500	281336	306900.9	#####	2.875	
6	39546	2009/4/10	2009/4/15 男	6	贵阳	贵州	CN		48	2014/3/31	152	273844	0	42265	124560	130702	309928	300834.1	2014/3/27	19	
7	56972	2008/2/10	2009/9/29 男	6	广州	广东	CN		64	2014/3/31	92	313338	0	27323	112364	76946	294585	285067.7	2014/1/13	11.5	
8	44924	2006/3/22	2006/3/29 男	6	乌鲁木齐	新疆	CN		46	2014/3/31	101	248864	0	37689	120500	114469	287042	277095	2014/3/31	12.625	
9	22631	2010/4/9	2010/4/9 女	6	温州市	浙江	CN		50	2014/3/31	73	301864	0	39834	82440	114971	287230	276335.4	2014/3/29	9.125	
10	32197	2011/6/7	2011/7/1 男	5	DRANCY		FR		50	2014/3/31	56	262958	0	31700	72596	87401	321489	266346.6	2014/3/26	7	
11	31645	2010/7/5	2010/7/5 女	6	温州	浙江	CN		43	2014/3/31	64	204855	0	47052	85258	60267	375074	265556.2	2014/3/17	8	
12	58877	#####	##### 女	6	PARIS	PARIS	FR		34	2014/3/31	43	298321	0	39018	69056	91581	262013	259041.3	2014/3/10	5.375	
13	37994	#####	2004/12/2 男	6	北京	.	CN		47	2014/3/31	145	256093	0	44539	92975	126821	271438	258554.2	2014/3/26	18.125	
14	28012	#####	##### 男	5	SAN MARIN	CA	US		58	2014/3/31	29	210269	0	35539	44750	53977	321529	256942.5	2014/1/25	3.625	
15	54943	#####	##### 男	6	深圳	广东	CN		47	2014/3/31	118	241614	0	26426	105466	119832	179514	251029.1	2014/3/29	14.75	
16	57881	2010/2/1	2010/2/1 女	6	广州	广东	CN		45	2014/3/31	50	289917	0	38028	68941	79076	270067	248997.7	2014/3/30	6.25	
17	1254	2008/3/28	2008/4/5 男	4	BOWLAND H	CALIFORNIA	US		63	2014/3/31	22	286164	0	23338	69300	54764	234721	240843.6	2014/1/27	2.75	
18	8253	2010/7/15	2010/8/20 男	6	乌鲁木齐	新疆	CN		48	2014/3/31	101	219995	0	23381	93840	93114	172231	238802.6	2014/3/25	12.625	
19	58899	#####	2011/2/23 女	6	PARIS		FR		50	2014/3/31	40	249882	0	31823	66239	63260	284160	238081.8	2014/2/16	5	
20	26955	2006/4/6	2007/2/22 男	6	乌鲁木齐市	新疆	CN		54	2014/3/31	64	215013	0	22036	99735	93006	169358	237371.5	2014/3/30	8	
21	41616	2011/8/29	##### 男	6	东莞	广东	CN		41	2014/3/31	38	191038	0	24656	60930	52316	332896	235785.4	2014/3/8	4.75	
22	21501	2008/7/30	##### 男	6	.	北京	CN		49	2014/3/31	106	220641	0	30493	69566	122763	167113	228845.2	2014/3/28	13.25	
23	41281	2011/6/7	2011/6/9 男	6	VECHEL	NORD BRABAN			2014/3/31	23	255573	0	29947	46800	198224	214590	227815.4	2014/3/26	2.875		
24	47229	2005/4/10	2005/4/10 男	6	广州	广东	CN		69	2014/3/31	94	193169	0	35222	59169	74497	305250	226435.6	2014/3/9	11.75	
25	28474	2010/4/13	2010/4/13 男	6	CA	US			41	2014/3/31	20	256337	0	24423	64258	59600	222380	223470.7	2014/1/18	2.5	
26	58472	2010/2/14	2010/3/1 女	5		FR			48	2014/3/31	44	204801	0	30638	38510	75816	281837	221892.7	2014/3/15	5.5	
27	13942	#####	2010/11/1 男	6	PARIS	FRANCE	FR		39	2014/3/31	62	241719	0	32263	72806	83496	243674	220035.2	2014/3/15	7.75	
28	45075	2007/2/1	2007/3/23 男	6	湛江	广东	CN		46	2014/3/31	213	217809	0	20801	136769	96568	187917	215419.5	2014/3/		

(数据集, 由于列太多了, 不太完全)

	A	B	
16	第一年总票价	SUM_YR_1	
17	第二年总票价	SUM_YR_2	
18	观测窗口总飞行公里数	SEG_KM_SUM	
19	观测窗口总加权飞行公里数 (Σ舱位折扣×航段距离)	WEIGHTED_SEG_KM	
20	末次飞行日期	LAST_FLIGHT_DATE	最后一次飞行时间
21	观测窗口季度平均飞行次数	AVG_FLIGHT_COUNT	
22	观测窗口季度平均基本积分累积	AVG_BP_SUM	
23	观察窗口内第一次乘机时间至MAX (观察窗口始端, 入	BEGIN_TO_FIRST	
24	最后一次乘机时间至观察窗口末端时长	LAST_TO_END	
25	平均乘机时间间隔	AVG_INTERVAL	
26	观察窗口内最大乘机间隔	MAX_INTERVAL	
27	观测窗口中第1年其他积分 (合作伙伴、促销、外航	ADD_POINTS_SUM_YR_1	
28	观测窗口中第2年其他积分 (合作伙伴、促销、外航	ADD_POINTS_SUM_YR_2	
29	积分兑换次数	EXCHANGE_COUNT	
30	平均折扣率	avg_discount	
31	第1年乘机次数	P1Y_Flight_Count	
32	第2年乘机次数	L1Y_Flight_Count	
33	第1年里程积分	P1Y_BP_SUM	
34	第2年里程积分	L1Y_BP_SUM	
35	观测窗口总精英积分	EP_SUM	
36	观测窗口中其他积分 (合作伙伴、促销、外航转入等)	ADD_Point_Sum	
37	非乘机积分总和	Eli_Add_Point_Sum	
38	第2年非乘机积分总和	L1Y_Eli_Add_Points	
39	总累计积分	Points_Sum	
40	第2年观测窗口总累计积分	L1Y_Points_Sum	
41	第2年的乘机次数比率	Ration_L1Y_Flight_Count	
42	第1年的乘机次数比率	Ration_P1Y_Flight_Count	
43	第1年里程积分占最近两年积分比例	Ration_P1Y_BPS	
44	第2年里程积分占最近两年积分比例	Ration_L1Y_BPS	
45	非乘机的积分变动次数	Point_NotFlight	

(意义)

## 0.1 清洗掉空的和无意义的行

```
def clean_data(data):
    """
    去掉na及一些可能出错、没太多意义的数
    :param datafile: 原始数据文件路径
    :return: 清洗完的数据
    """
    # 去掉na
    data = data.dropna()
    # 只保留票价非零的, 或者平均折扣率与总飞行公里数同时为0的记录。
    index1 = data['SUM_YR_1'] != 0
    index2 = data['SUM_YR_2'] != 0
    index3 = (data['SEG_KM_SUM'] == 0) & (data['avg_discount'] == 0)
    data = data[index1 | index2 | index3]
    return data
```

## 0.2 选取其中 6 个比较有代表性的特征

```
def simplify_data(data):  
    """  
    选择比较有用的6个属性: FFP_DATE、LOAD_TIME、FLIGHT_COUNT、AVG_DISCOUNT、SEG_KM_SUM、LAST_TO_END  
    分别是: 第一次飞行日期, 观察窗口结束时间, 飞行次数, 平均折扣, 总飞行公里数, (到观察结束为止) 多少未飞行  
    :param datafile: input path  
    :param okfile: output filepath  
    """  
    valid_keys = ["FFP_DATE", "LOAD_TIME", "FLIGHT_COUNT", "avg_discount", "SEG_KM_SUM", "LAST_TO_END"]  
    data = data[valid_keys]  
    data.to_csv(okfile, encoding='utf-8')  
    return data
```

## 0.3 得到 5 个对客户价值的评价指标: 客户关系时长 L、消费时间间隔 R、消费频率 F、飞行里程 M 和折扣系数的平均值 C, 并归一化

```
def proc_indecator(data):  
    """  
    获取以下5个对于客户的评价指标, 并进行正则化  
    客户关系时长L、消费时间间隔R、消费频率F、飞行里程M和折扣系数的平均值C五个指  
    日期均为月份的差值  
    :param datafile: input path  
    :return: output  
    """  
    res_data = pd.DataFrame()  
    # L: 乘客入会日期与乘客里最早入会的日期的差值, 按月(30天)算算代表新/老程度  
    ffp_time = pd.to_datetime(data["FFP_DATE"], format="%Y/%m/%d")  
    load_time = pd.to_datetime(data["LOAD_TIME"], format="%Y/%m/%d")  
    res_data["L"] = (load_time - ffp_time).apply(lambda x: int(int(((str(x)).split()[0])) / 30))  
    # R: 客户最后一次坐飞机至今的月份数  
    res_data["R"] = data['LAST_TO_END']  
    # F: 超过你坐飞机的次数  
    res_data["F"] = data['FLIGHT_COUNT']  
    # M: 总里程数  
    res_data["M"] = data['SEG_KM_SUM']  
    # C: 平均折扣率  
    res_data["C"] = data['avg_discount']  
    # 标准化  
    res_data = (res_data - res_data.min()) / (res_data.max() - res_data.min())  
    return res_data  
###
```



步骤 0 处理完成后，得到数据集如下：

	A	B	C	D	E	F
1		L	R	F	M	C
2	1688	0.5294118	0.0013699	0.1848341	0.1244165	0.4596454
3	25202	0.872549	0.1082192	0.0379147	0.0174809	0.5079109
4	38513	0	0.5123288	0	0.0131507	0.3035106
5	15729	0.1568627	0.1410959	0.0853081	0.0355786	0.4056146
6	5006	0.0980392	0.1739726	0.0853081	0.0793867	0.4200809
7	38339	0.2352941	0.169863	0.0189573	0.0134367	0.2988393
8	35347	0.9803922	0.1136986	0.0331754	0.0166744	0.278577
9	3976	0.1960784	0.0109589	0.0616114	0.0477058	0.8786521
10	19774	0.1960784	0.0589041	0.0473934	0.0272198	0.4264575
11	24364	0.8039216	0.2479452	0.042654	0.0238994	0.3689629
12	21431	0.3431373	0.0465753	0.042654	0.0247213	0.4305027
13	34806	0.5196078	0.0506849	0.0379147	0.0115138	0.4545981
14	24285	0.4901961	0.0013699	0.0189573	0.0138796	0.6956105
15	45315	0.5392157	0.1849315	0.0047393	0.0075765	0.3578358
16	32479	0.2352941	0.3219178	0.0189573	0.0138262	0.425128
17	18093	0.2352941	0.1219178	0.07109	0.0406704	0.2881256
18	3158	0.8529412	0.0821918	0.1279621	0.0800294	0.5549654
19	31162	0.2647059	0.0219178	0.0236967	0.0125804	0.5143259
20	20654	0.2156863	0.0054795	0.0853081	0.0234118	0.4807051
21	28237	0.9117647	0.0438356	0.0379147	0.0215939	0.3233328
22	52705	0.1176471	0.7150685	0	0.003541	0.4226205
23	48586	0.0588235	0.7630137	0	0.0049126	0.4508574

（为了实验跟快速地进行，只随机地选择了原数据集 5 万多条数据中的部分保存。第一列是对应原数据集里面的编号。）

**步骤 1** 把上述的 LRFMC 的每一个指标，都给离散化。

1.1 通过聚类，给每个指标都划成 4 个区间，返回该区间的最小值以及样本个数。如下：

	1	2	3	4
L	0	0.2074964	0.4547082	0.7108682
Ln	23570	14505	11758	8185
R	0	0.1357649	0.3515713	0.6312748
Rn	28553	14970	8413	6082
F	0	0.0455043	0.1289501	0.2763089
Fn	38699	14089	4231	999
M	0	0.0282627	0.0726246	0.1581721
Mn	38865	13890	4453	810
C	0	0.3357799	0.4717739	0.6987478
Cn	12432	27207	16374	2005

LRFMC 每一行代表一个指标最小值，Xn 每行代表 X 指标的样品个数  
列代表每个指标划分成的 4 类。在 rules.csv 里。

代码如下：

```
def get_typeRules(data, keys, k):
    """把5列指标给离散化，即把每一列处于某个区间的数归为一类、另一区间又另一类
    :param k: 每一列都划分k个区间
    :return: 每个区间的最小值，以及该区间的样本个数
    """
    result = pd.DataFrame()
    for i in range(len(keys)):
        # 调用k-means算法，进行聚类离散化
        print(u'正在进行“%s”的聚类...' % keys[i])
        kmodel = KMeans(n_clusters = k)
        kmodel.fit(data[[keys[i]]].as_matrix()) # 训练模型

        r1 = pd.DataFrame(kmodel.cluster_centers_, columns=[keys[i]]) # 聚类中心
        r2 = pd.Series(kmodel.labels_).value_counts() # 分类统计
        #print(kmodel.cluster_centers_)
        #print(kmodel.labels_)
        r2 = pd.DataFrame(r2, columns=[keys[i]+'n']) # 转为DataFrame，记录各个类别的数目
        r = (pd.concat([r1, r2], axis=1)).sort_values(keys[i]) # 匹配聚类中心和类别数目
        r.index = [1, 2, 3, 4]

        r[keys[i]] = pd.rolling_mean(r[keys[i]], 2) # 用来计算相邻2列的均值，以此作为边界点。
        r[keys[i]][1] = 0.0 # 这两句代码将原来的聚类中心改为边界点。
        result = result.append(r.T)
    print(result.head())
    return result
```



## 1.2 实现指标离散化为标号，如 L1, L2, L3, L4, M1, M2,....

```
1 L3,R0,F3,M3,C3
2 L2,R0,F1,M3,C3
3 L2,R0,F3,M3,C2
4 L2,R0,F3,M3,C2
5 L1,R0,F3,M3,C2
6 L1,R0,F3,M3,C1
7 L1,R0,F2,M3,C2
8 L3,R0,F3,M3,C2
9 L3,R0,F1,M3,C2
10 L3,R0,F3,M3,C3
11 L1,R0,F2,M3,C2
12 L2,R0,F1,M3,C2
13 L1,R0,F3,M3,C3
14 L3,R0,F3,M3,C3
15 L0,R0,F2,M3,C1
16 L2,R0,F3,M3,C3
17 L3,R0,F3,M3,C1
18 L1,R0,F3,M3,C2
19 L3,R0,F3,M3,C3
20 L3,R0,F3,M3,C3
21 L3,R0,F3,M3,C3
```

(dis.txt 里)

代码如下：

```
def get_everyKind(data, tIndex, keys=list("LRFMC")):
    """
    给每个LRFMC的每个指标都离散化为XN，X为LRFMC，N为1234区间号
    :param data: 原始LRFMC一共5列数据
    :param tIndex: LRFMC五行、各自程度为1,2,3,4的区间
    :return: 每行类似于L1,R2,F2,M0,C1...这样，L1的1表示L指标它根据区间分为1
    """
    # 转换规则为字典
    rule_dict = {}
    for index, row in tIndex.iterrows():
        rulelist = list(row)
        rule_dict[rulelist[0]] = rulelist[1:].copy()
        rule_dict[rulelist[0]].append(1.01) # 最大
    # 分
    res_list = []
    for index, row in data.iterrows():
        tmp_lsit = []
        for key in keys:
            n = len(rule_dict[key]) # 含1
            for i in range(n - 1):
                if row[key] > rule_dict[key][i] and row[key] <= rule_dict[key][i + 1]:
                    tmp_lsit.append(key + str(i))
                    break
            if len(tmp_lsit) == len(keys):
                res_list.append(",".join(tmp_lsit))
    return res_list
```

### 1.3 把数据转换为类似于 one-hot 矩阵

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1		C0	C1	C2	C3	F0	F1	F2	F3	L0	L1	L2	L3	M1	M2	M3	R0
2	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1
3	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	1
4	2	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	1
5	3	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	1
6	4	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1
7	5	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	1
8	6	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	1
9	7	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	1
10	8	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1
11	9	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	1
12	10	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1
13	11	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	1
14	12	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1
15	13	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	1
16	14	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	1
17	15	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	1
18	16	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	1
19	17	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1
20	18	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1
21	19	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1
22	20	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	1
23	21	0	0	0	1	0	0	1	0	0	0	0	1	0	0		

(association\_matrix.csv 文件里，每一行代表每一行数据)

代码如下：

```
def get_buyOrNotMatrix(data):  
    '''一类(如A1,B2,C1...)为一列，每行顾客是否属于此类为0/1'''  
    print(u'\n转换原始数据至0-1矩阵...')  
    ct = lambda x : pd.Series(1, index = x[pd.notnull(x)])  
    b = list(map(ct, data.as_matrix()))  
    data = pd.DataFrame(b).fillna(0)  
    return data
```

## 步骤2：进行关联分析

最低支持度为 0.06，分别尝试最小置信度为 0.60, 0.65, 0.70, 0.75, 0.80, 0.85 下，寻找频繁项集。

代码如下：

```
60 # 连接函数，用于实现L_{k-1}到C_k的连接
61 def connect_string(x, ms):
62     x = list(map(lambda i: sorted(i.split(ms)), x))
63     l = len(x[0])
64     r = []
65     for i in range(len(x)):
66         for j in range(i, len(x)):
67             if x[i][:l-1] == x[j][:l-1] and x[i][l-1] != x[j][l-1]:
68                 r.append(x[i][:l-1] + sorted([x[j][l-1], x[i][l-1]]))
69     return r
70
71 # 寻找关联规则的函数
72 def get_rules(d, support, confidence, ms='--') :
73     result = pd.DataFrame(index=['support', 'confidence']) # 定义输出结果
74     support_series = 1.0 * d.sum() / len(d) # 支持度序列
75     column = list(support_series[support_series > support].index) # 1-频繁的下标
76     k = 0
77
78     while len(column) > 1:
79         k += 1
80         print('\n正在进行第%d次搜索...' % k)
81         column = connect_string(column, ms)
82         print('数目: %d...' % len(column))
83         sf = lambda i: d[i].prod(axis=1, numeric_only=True) # 新一批支持度的计算函数
84
85         d_2 = pd.DataFrame(list(map(sf, column)), index=[ms.join(i) for i in column]).T
86         support_series_2 = 1.0 * d_2[[ms.join(i) for i in column]].sum() / len(d) # 计算连接后的支持度
87         column = list(support_series_2[support_series_2 > support].index) # 新一轮支持度筛选
88         support_series = support_series.append(support_series_2)
89         column_2 = [] # 备选的关联
90
91         for i in column:
92             i_list = i.split(ms)
93             for j in range(len(i_list)):
94                 column_2.append(i_list[:j] + i_list[j+1:] + i_list[j:j+1])
95
96         cofidence_series = pd.Series(index=[ms.join(i) for i in column_2]) # 对于每个序列的置信度
97         for i in column_2: # 计算置信度
98             cofidence_series[ms.join(i)] = support_series[ms.join(sorted(i))] / support_series[ms.join(i[:len(i)-1])]
99
100         for i in cofidence_series[cofidence_series > confidence].index: # 置信度筛选
101             result[i] = 0.0
102             result[i]['confidence'] = cofidence_series[i]
103             result[i]['support'] = support_series[ms.join(sorted(i.split(ms)))]
104
105     result = (result.T.sort_values(['confidence', 'support'], ascending=False))
106     return result
```

## 5、实验结果：

频繁项集结果保存在以下 6 个文件夹中：

association_result_0.60.txt	2019/6/16 11:07	TXT 文件	21 KB
association_result_0.65.txt	2019/6/16 11:07	TXT 文件	19 KB
association_result_0.70.txt	2019/6/16 11:07	TXT 文件	18 KB
association_result_0.75.txt	2019/6/16 11:07	TXT 文件	17 KB
association_result_0.80.txt	2019/6/16 11:07	TXT 文件	17 KB
association_result_0.85.txt	2019/6/16 11:07	TXT 文件	16 KB

以置信度最小为 0.85 的为列：

结果为：

	support	confidence
C1--F3--M3--R0	0.116178	1.000000
F3--L3--M3--R0	0.106406	1.000000
C1--F3--L3--R0	0.074376	1.000000
C1--F3--R0	0.180782	0.997006
F3--L3--R0	0.159066	0.996599
F3--M3--R0	0.235613	0.995413
C1--F3--M2--R0	0.064604	0.991667
C2--F3--L3--R0	0.064061	0.991597
F3--R0	0.383822	0.988811
F3--L2--R0	0.086862	0.987654
C2--F3--M3--R0	0.086862	0.987654
C1--L3--M3--R0	0.074919	0.985714

## 6、实验总结：

1. Apriori 算法理解起来似乎不算难，而且也有书上的算法来参考但实现起来也有种种问题要克服。个人代码能力需要提供。

2. 数据预处理和特征选择是一本数据挖掘相关的书提供的思路，我发现其实在真正关联分析前的这些步骤，也是很重要的，并且对于我这种对 python 里面 pandas, numpy 这些库不算熟的人来说，也要耗一点精力和时间。

3. 收获：对关联规则算法更加理解加深了，并且对 python 里上述 pandas, numpy, sklearn 这些数据挖掘处理的常用库，更加熟练了。后面两个实验做起来比这个顺利了一点。

4. 一个注意的点：用 pandas 读 csv 和存 csv 时，一定要用 `.iloc[:, 1:]`，把第 0 列 index 去掉！不然连索引 index 也会被当成数据，然后实验结果会异常！以后我写代码时也应注意这一点。

# 暨南大学本科实验报告专用纸

课程名称 数据挖掘 成绩评定   
实验名称 实验二 分类 指导教师 刘波  
试验编号 0806011602 类型 综合性  
学生姓名 邝庆璇 学号 2016051598  
学院 信息科学技术学院 系 计算机 专业 计算机科学与技术

## 1、实验目的：

- (1) 理解数据挖掘分类算法的原理
- (2) 掌握并熟悉常用分类算法

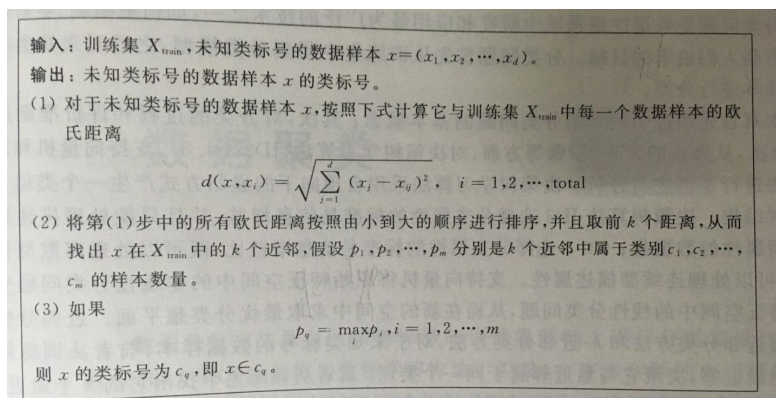
## 2、实验内容

利用 K 近邻算法，根据实验一中预处理得到的 LRMC，把乘客分为“常客”和“并非常客”两类。

## 3、实验原理

如书上 P128 所言，基本步骤为：

- 1) 计算测试数据与各个训练数据之间的距离；
- 2) 按照距离的递增关系进行排序；
- 3) 选取距离最小的 K 个点；
- 4) 确定前 K 个点所在类别的出现频率；
- 5) 返回前 K 个点中出现频率最高的类别作为测试数据的预测分类。



(课本 P129)



#### 4、实验过程：

(1) 把实验一里面预处理好的、含有每个顾客 LFMRC 这 5 个指标的数据 (air\_LRFMC.csv)，按照 F (Frequency, 顾客搭乘频率) 是否大于 0.06，把顾客分为“常客”和“并非常客”两类，并据此把 F 指标分别离散化为 1 和 0。

```
def getLabels(data, freq = 0.06):  
    """  
    以F=freq为界限，分两类，列为label并去除F这一列  
    :param freq: 阈值  
    :return: 加了label之后  
    """  
    data['F'] = data['F'].apply(lambda x: 1 if x >= freq else 0)  
    x, y = data[list("LFMRC")], data['F']  
    return (x, y)
```

其中，x 是包含了 LRFMC 这 4 个指标的 4 列，y 是 F 指标的 0-1 值（即上述是否为“常客”的两列）

(2) 把数据随机划分为训练集和测试集：

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=211)
```

(3) 尝试取不同的 K 值，用 K 近邻法，对测试集进行分类

```
best_k = 2  
best_score = np.inf  
best_pred = pd.Series()  
for k in range(2, 9):  
    y_predict = getKnnClassify(x_test, x_train, y_train, k, y_test)  
    # 计算平方差损失，并更新最佳  
    curr_score = mean_squared_error(y_test, y_predict)  
    print("k = %d, MSE损失%f" % (k, curr_score))  
    if (curr_score < best_score):  
        best_score, best_k, best_pred = curr_score, k, y_predict
```

getKnnClassify 函数见下页

```

def getDistance(xlist1:list, xlist2:list):
    '''获取xlist1和xlist2欧氏距离'''
    return np.sqrt(np.sum(np.power([xlist1[i] - xlist2[i] for i in range(len(xlist1))], 2)))

def getKnnClassify(x_test:pd.DataFrame, x_train:pd.DataFrame, y_train:pd.Series, k=4, y_test=None):
    '''
    用K临近分类，根据LRMC这4个指标，把数据分为F指标下的两类
    :param x: LRMC这4列
    :return: y_predict即每一行的分类结果
    '''
    # y_predict = pd.DataFrame(index=x_test.index)
    y_predict = []
    j = -1
    for i_test, row_test in x_test.iterrows():
        lrmc_list = list(row_test) # 一行的lrmc指标
        d_list = [] # 欧式距离
        ytrain_list = list(y_train)
        i = 0
        for i_train, row_train in x_train.iterrows():
            curr_d = getDistance(lrmc_list, list(row_train))
            d_list.append((y_train.iloc[i], curr_d)) # d_list: (class, distance)
            i += 1
        # 选择K近邻
        d_list = sorted(d_list, key=lambda x: x[1])[:k]
        # 获取概率
        count_dict = dict().fromkeys(set(y_train), 0)
        for tup in d_list:
            count_dict[tup[0]] += 1
        # 添加概率最大者
        y_predict.append(max(count_dict.items(), key=lambda x: x[1])[0])
        j += 1
    return pd.Series(y_predict, x_test.index) # 一定要series!

```

## 5、实验结果

不同 K 值下的损失:


```


k = 2, MSE损失0.130000
k = 3, MSE损失0.115000
k = 4, MSE损失0.130000
k = 5, MSE损失0.120000
k = 6, MSE损失0.125000
k = 7, MSE损失0.115000
k = 8, MSE损失0.125000
best k = 3, 损失0.115000

```

可见最佳的 K 是 3，即挑选临近的 3 个训练集的点。

预测结果保存在下面 3 个文件夹，knn\_both 第一列测试集预测结果，第二列测试集真实；knn\_pred 前几列分别为除了 F 外的 4 个指标，最后一列是测试集预测结果

 knn\_both\_3.csv

 knn\_pred\_3.csv

## 6、实验总结

有了上个实验的积累，分类的实现就好做多了，而且 KNN 比 apriori 算法还简单一点。虽然实现上面仍然踩了一些坑，毕竟之前的确很久没用 pandas 这些了，但后来问题也解决了。

# 暨南大学本科实验报告专用纸

课程名称 数据挖掘 成绩评定                       
实验名称 实验三 聚类 指导教师 刘波  
试验编号 0806011603 类型 综合性  
学生姓名 邝庆璇 学号 2016051598  
学院 信息科学技术学院 系 计算机 专业 计算机科学与技术

## 1、实验目的：

- (1) 理解聚类算法原理
- (2) 掌握并熟悉聚类算法的使用

## 2、实验内容

利用 K 近邻算法，根据实验一中预处理得到的 LRMC 这 5 个指标，把乘客进行聚类。

## 3、实验原理

原理如书上所言。

- 1.创建 k 个点作为起始质心，可以随机选择(位于数据边界内);
- 2.while 任意一个点的簇分配结果发生改变：
  - 3.For 数据集中每一个点：
    - ①对每个质心计算质心与数据点之间的距离
    - ②将数据点分配到距其最近的簇
- 4.对每一个簇，计算簇中所有点的均值并将均值作为质心；回到 2；

## 4、实验步骤

尝试 K 从 2 到 6，对 KRMFC 这 5 个指标进行聚类

核心代码如下：

```
def getDistance(xlist1:list, xlist2:list):
    '''获取xlist1和xlist2欧氏距离'''
    return np.sqrt(np.sum(np.power([xlist1[i] - xlist2[i] for i in range(len(xlist1))], 2)))

def getRandCenter(data:pd.DataFrame, k:int):
    '''随机生成k个簇的中心'''
    feature_count = data.shape[1] # 列数
    centers = np.zeros((k, feature_count))
    for i in range(feature_count):
        # 选取第i列的最大最小，作为范围
        data_min, data_max = np.min(data[:,i]), np.max(data[:, i])
        # 生成随机
        centers[:, i] = data_min + np.random.rand(k) * (data_max - data_min)
    return centers

def clusterByKMeans(data, k=3) :
    '''kmeans聚类'''
    m = data.shape[0]
    res_cluster = np.mat(np.zeros((m, 2))) # 用于存放该样本属于哪类及质心距离
    # res_cluster第一列存放该数据所属的中心点，第二列是该数据到中心点的距离
    centers = getRandCenter(data, k)
    hasChanges = True # 聚类是否已经收敛，若收敛则结束

    while hasChanges:
        hasChanges = False
        for i in range(m): # 把每一个数据点划分到离它最近的中心点
            min_dis = np.inf # min_distance
            min_index = -1
            for j in range(k):
                curr_min = getDistance(centers[j, :], data[i, :])
                if curr_min < min_dis: # 更新最小距离
                    min_dis, min_index = curr_min, j
            if res_cluster[i, 0] != min_index:
                hasChanges = True # 若变了，则要继续迭代
            res_cluster[i, :] = min_index, min_dis ** 2 # 第i个数据点的分配情况存入字典
            # print(centers) # 每次迭代的中心

        for cent in range(k) : # 重新计算中心点
            ptsInClust = data[np.nonzero(res_cluster[:, 0].A == cent)[0]]
            centers[cent, :] = np.mean(ptsInClust, axis=0) # 算出这些数据的中心点

    return centers, res_cluster
```

利用轮廓系数来对聚类效果打分，选取最佳效果的 K

```
best_k = 2
best_score = -1
best_pred = pd.Series() # pred即cluster分配
for k in range(2, 7):
    centers, res_cluster = clusterByKMeans(mData)
    cluster_list = [int(res_cluster[k,0]) for k in range(len(res_cluster))] # 转为pd.Series
    y_pred = pd.Series(cluster_list, index=data.index)
    # 按照轮廓系数打分，并更新最佳的k
    curr_score = silhouette_score(data, y_pred)
    print("k = %d, 轮廓系数%f" % (k, curr_score))
    if curr_score > best_score:
        best_score, best_k, best_pred = curr_score, k, y_pred

# 保存最佳的
print("最佳k = %d, 轮廓系数%f" % (best_k, best_score))
pd.concat([data, best_pred], axis=1).to_csv(resp + str(best_k) + ".csv")
```



## 5、实验结果

k = 2, 轮廓系数打分0.579303

k = 3, 轮廓系数打分0.579303

k = 4, 轮廓系数打分0.579303

k = 5, 轮廓系数打分0.579303

k = 6, 轮廓系数打分0.579303

最佳k = 2, 轮廓系数0.579303

得分比较接近，可能是这 5 个特征的数据并不合适聚类，也许需要进行降维或者其他处理。结果保存在 kmeans\_result\_3.csv 这个文件中，截图如下：

	L	R	F	M	C	
3779	0.0980392	0.4808219	0	0.0067201	0.3218393	2
3248	0.0980392	0.3068493	0.0047393	0.004549	0.490619	2
2430	0.1960784	0.330137	0.0236967	0.026746	0.422046	1
3635	0.6568627	0.190411	0.0189573	0.0192901	0.1744335	0
2474	0.0882353	0.0136986	0.0900474	0.0192195	0.5183617	0
2509	0.2352941	0.3191781	0.0094787	0.0113966	0.4039418	0
2086	0.0588235	0.739726	0	0.0043078	0.5417833	2
3352	0.0490196	0.3589041	0.042654	0.0283657	0.443475	1
2488	0.5588235	0.060274	0.0379147	0.0226433	0.4884015	0
2172	0.1372549	0.1041096	0.0047393	0.009017	0.1031171	2

（最后一列为类别）

## 6、实验总结

KMeans 聚类的思想不难，但这次想试试不用 pandas 的 DataFrame，改用了 numpy 的 np.matrix，有不少不熟悉的地方，但也学到了不少。