基于 K-mean++聚类和协同过滤算法的电视产品营销方案

摘 要

伴随着互联网技术的快速发展和应用拓展，“三网融合”为传统广播电视媒介带来了发展机遇，广播电视运营商可与众多的家庭用户实现信息的实时交互，使得全方位、个性化的产品营销和有偿服务成为现实。但是由于对用户在电视产品的选择方面的信息收集能力有限，因此需要对用户在电视观看方面进行处理，得到最终的模型。

本文工作主要是包括两个方面：

（1） 问题一，对数据表进行项目统计，构建用户_产品_偏好度矩阵 user_item_values。

统计项目包括：用户观看的节目中的类别分布
用户观看节目中相同导演出现的次数
用户观看节目中相同主演的次数
用户观看节目的年代分布（5 年为一个跨度）

偏好度（values）：项目出现一次，偏好度加 1。

基于内容的协同过滤算法计算推荐产品：

input(输入)：产品_用户_偏好度矩阵（user_item_values.T:即用户_产品_偏好度矩阵的转置矩阵）

output(输出):推荐产品列表

（2） 问题二，在用户基本信息表中，我们对套餐按照资费情况进行等级划分，对用户根据以往的消费记录进行相同等级划分，以便向用户进行平级消费套餐的推荐。

利用 k-means++算法对用户分类：

input:机顶盒设备号_频道_观看时长.csv；机顶盒设备号_点播金额 csv；
机顶盒设备号_分类.csv；机顶盒设备号_分类_时长.csv；

output:用户的标签

利用 k-means++算法对产品分类：

input:产品_类目.csv；产品_金额.csv；产品_集数.csv；
产品_出品年代.csv；产品_导演.csv；产品_演员.csv；

output:产品标签

基于用户的协同过滤算法计算推荐产品：

input:用户_产品_偏好度矩阵 user_item_values

output:相似用户的推荐产品

【关键字】收视偏好 产品推荐 用户分类 K-mean++聚类 协同过滤 营销推荐

目录

1 问题重述

1.1 问题重述

从用户观看的节目名称，年代，导演，主演，类别出发，对用户和影片进行多角度分类，并对用户与产品进行标签处理分类，对标签化的产品打包，最终给出适合用户的套餐推荐和节目推荐。

1.2 问题概述

根据题中所给的信息，题目共分为以下两个大问题：

（1）问题一：附件 1 中给出了用户观看记录信息的数据，分析用户收视偏好，给出附件 2 中产品的营销推荐方案。

①从节目的年代，类目，导演，演员四个特征对产品进行偏好分析。

②利用基于内容的协同过滤算法，根据用户的观看偏好，将用户喜好看的电视产品且用户还未观看过的电视产品推荐给用户；

③对用户的观看记录进行数据统计分析；

④从节目的导演，年代，演员，和类目四个角度对用户进行准确性的相似产品推荐。

（2）问题二：为扩大营销范围，利用附件 1,2,3 中的数据，对相似偏好的用户进行分类，对产品进行分类打包，并给出营销推荐方案。

①K-means++算法将用户与影片的类别的矩阵进行聚类打包，电视产品的分类打包则利用第一问的影评的类别进行分类打包；

②利用基于用户的推荐算法，对具有相似偏好的用户给出合适的电视产品的推荐方案；

③利用用户_金额.csv 文件构建用户_消费矩阵 user_cost.csv 文件，根据用户的消费记录，向用户推荐合适的电视产品套餐方案。

2 分析方法与过程

2.1 算法概述[1]

2.1.1 K-means++算法

对问题二中的用户与影片的类别的矩阵进行聚类，以此来对相似偏好的用户进行新型分类打包，这需要采用 K-means++算法。K-means++算法的具体原理如下：

第一步，选取度量样本（喜欢的导演，喜欢的演员，喜欢的年代，消费的金额，节目的类别）。

第二步，计算欧氏距离。

假设有两点，分别为 P 和 Q，对应坐标分别为：

$$\mathrm{P} = (x_1, x_2, \ldots, x_n) \in R^n$$

$$\mathrm{Q} = (y_1, y_2, \ldots, y_n) \in R^n$$

则，点 P 与点 Q 之间的欧氏距离为：

$$\mathrm{d(P,Q)} = \sqrt{\sum_{i=1}^{n} (x_i - y_{i)}{}^2}$$

第三步，算法与矩阵分解。

在 K-means++算法中，假设训练数据集 X 中有 m 个样本$\{X^{(1)}, X^{(2)}, \ldots, X^{(n)}\}$，其中，每个样本$X^{(i)}$为 n 维向量，此时样本可表示为一个 m×n 的矩阵：

$$X_{m\times n} = (X^{(1)}, X^{(2)}, \ldots, X^{(n)})^T = \begin{pmatrix} x_1{}^{(1)} & \cdots & x_n{}^{(1)} \\ \vdots & \ddots & \vdots \\ x_1{}^{(m)} & \cdots & x_n{}^{(m)} \end{pmatrix}_{m\times n}$$

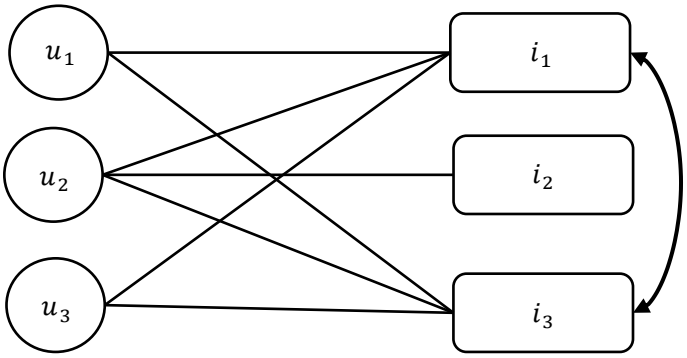
假设有 k 个类，分别为$\{C_1, \ldots, C_k\}$. 在 K-means++算法中，利用欧氏距离计算每个样本$X^{(i)}$与 k 个聚类中心之间的相似度，并将样本$X^{(i)}$划分到最相似的类别中，再利用划分到每个类别中的样本重新计算 k 个聚类中心。重复以上过程，直至质心不在改变为止。质心的求解方法如下所示:

$$C_k' = \frac{\sum_{X^{(i)}\in C_k} X^{(i)}}{\#(X^{(i)}\in C_k)},$$

其中$\sum_{X^{(i)}\in C_k} X^{(i)}$表示所有$C_k$类中的所有的样本的特征的向量的和，$\#(X^{(i)} \in C_k)$表示的是类别$C_k$中的样本的个数。

2.1.2 基于内容的协同过滤算法

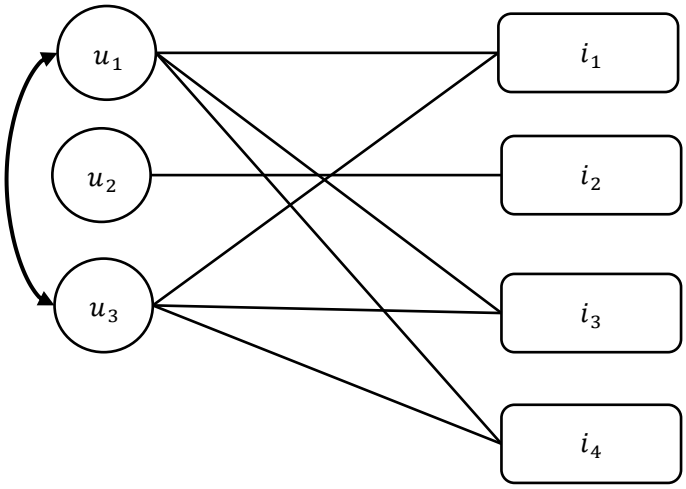
为了给出附件 2 中的产品营销方案，我们通过比较内容与内容之间的相似性，为用户推荐与其观看偏好相似的内容，所以这里采用基于内容的协同过滤算法。基于内容的协同过滤算法的原理如下所示：



假设用户分别为u_1、u_2、u_3，其中用户u_1互动的商品有i_1和i_3，用户u_2互动的商品有i_1，i_2和i_3，用户u_3互动的商品有i_1。通过计算，商品i_1和i_3,较为相似，对于用户u_3来说，用户u_1互动过的商品i_3是用户u_3未互动过的，因此会为用户u_3推荐商品i_3。

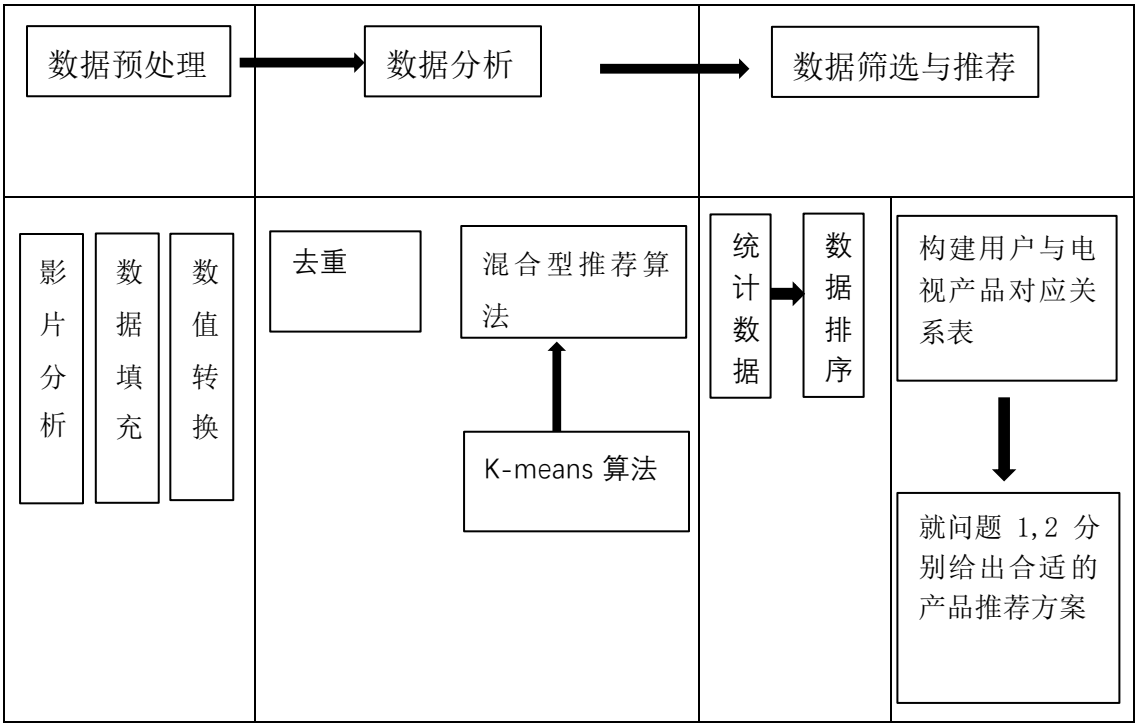
2.1.3 基于用户的协同过滤算法

为了给出附件 3 中的产品营销方案，利用基于内容的协同过滤算法，原理如下所示：



假设用户分别为u_1、u_2、u_3，其中用户u_1互动的商品有i_1和i_3，用户u_2互动的商品有i_2，用户u_3互动的商品有i_1，i_3和i_4,。通过计算，用户u_1和u_3,较为相似，对于用户u_1来说，用户u_3互动过的商品i_4是用户u_1未互动过的，因此会为用户u_1推荐商品i_4。

2.2 总体流程图



本用例主要包括如下步骤：

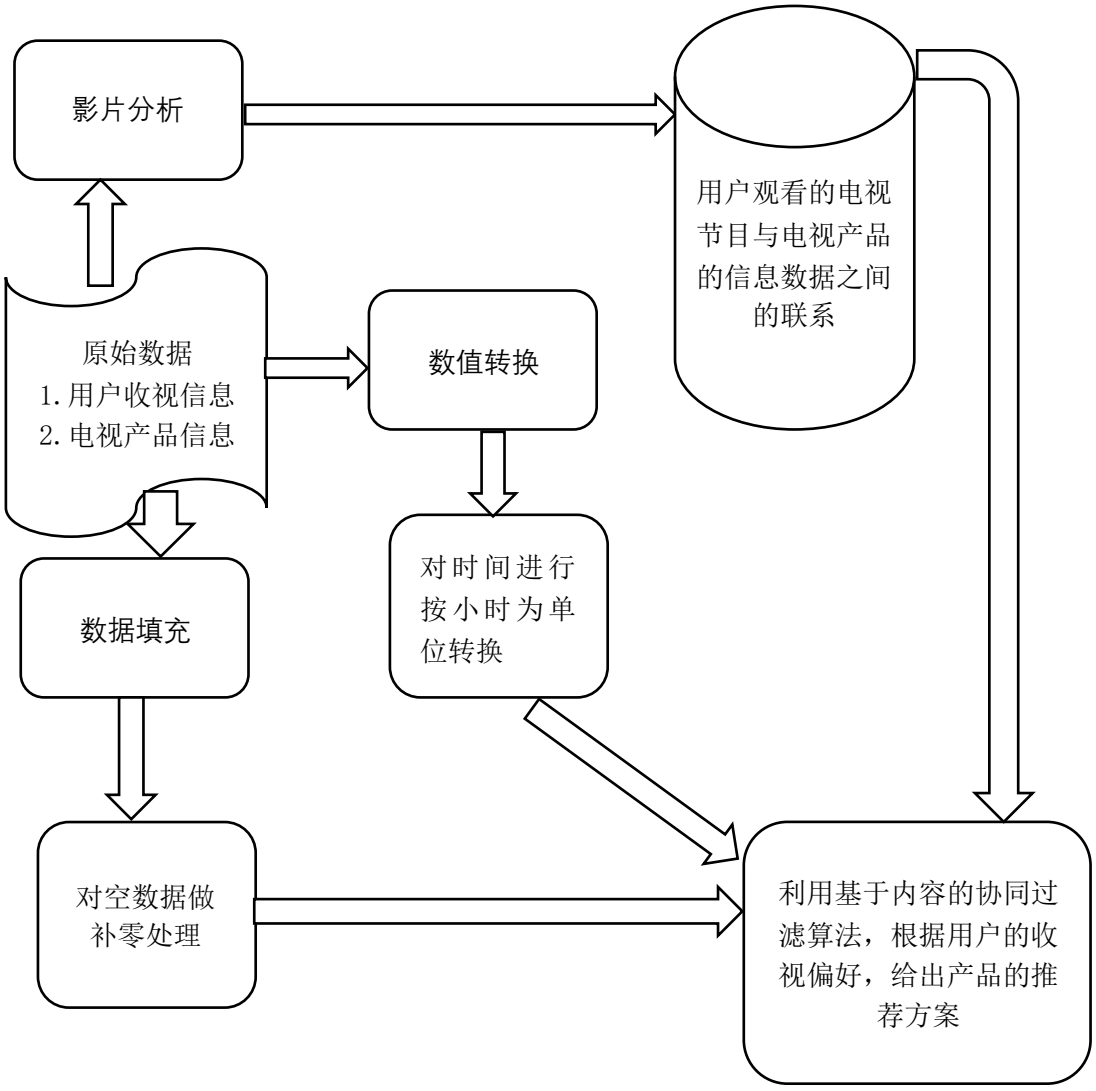
步骤一 数据预处理：数据填充，对数据中存在为空的地方用数值 0 替换；数值转换，对数据中的时间用浮点类型表示（以小时为单位）。影片分析，从产品的名称，所属年代，导演，主演，类别进行统计。

步骤二 数据分析：对数据进行处理后，在对用户推荐喜欢的同一导演的影片名称时，利用去重方法，对用户已观看过的影片进行去除，以便向用户荐新节目。采用 K-means++聚类分析算法对用户进行分类处理；

步骤三 数据筛选与推荐：统计分类后的数据，筛选汇总，从用户的相似性，产品的相似性两个角度对用户给出相应的产品推荐方案。

2.3 问题 1 分析方法与过程

2.3.1 流程图



2.3.2 数据分析

在已有的数据中，将用户对该节目的偏好度=偏好度（年代）+偏好度（导演）+偏好度（演员）+偏好度（类目）。以词频多少等价为用户对产品的偏好程度。

第一步：从 1979 年一直到 2018 年，以五年为一个阶段，找出每个阶段观看电视节目的用户的数量。具体数据分析如下所示（只截取了部分数据）：

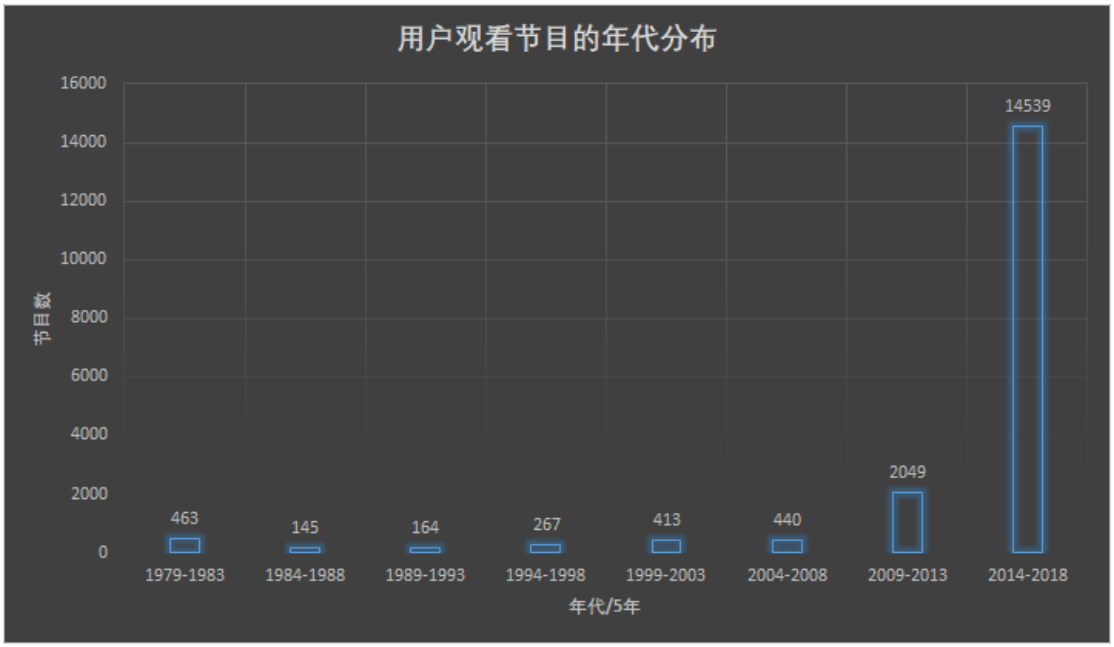


图 1 用户观看的节目的年代分布

第二步：找出每个电视节目的执行导演，并找出用户观看的电视节目中同一导演出现的次数，具体数据分析如下所示（只截取了部分数据）：

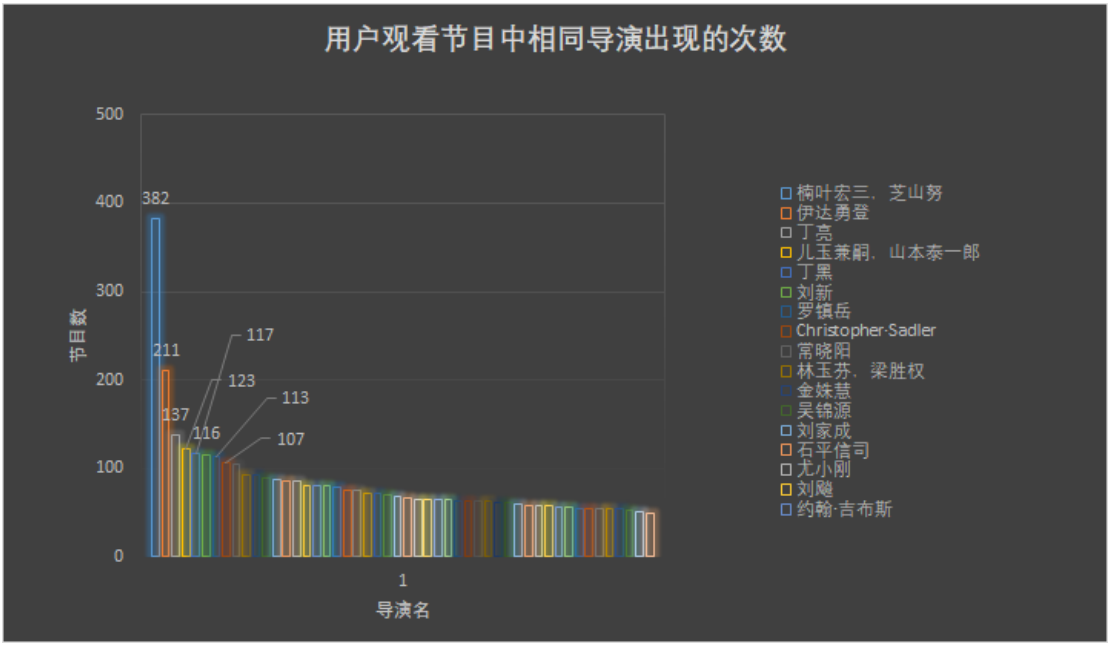


图 2 用户观看节目中的相同导演出现的次数

第三步：找出每个电视节目中的主演阵容，并找出用户在观看的节目中的主演出现的次数，具体数据分析如下所示（只截取了部分数据）：

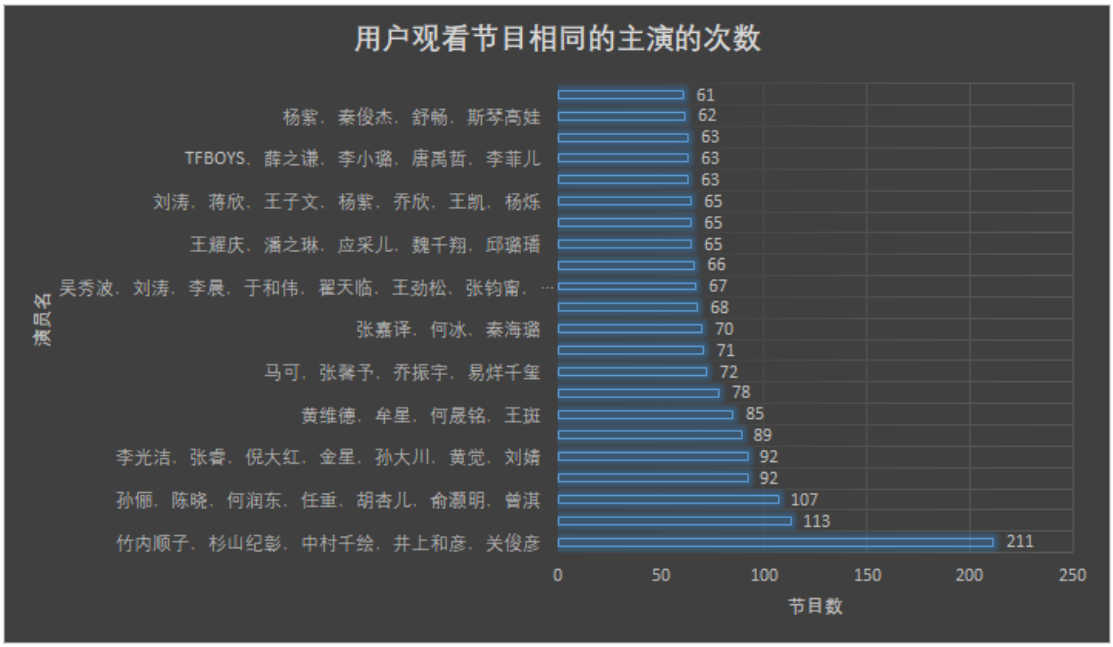


图 3 用户观看节目相同的主演的次数

第四步 :找出用户观看过的所有的电视节目的类别,再找出每个电视类别下观看用户的数量,具体数据分析如下所示：

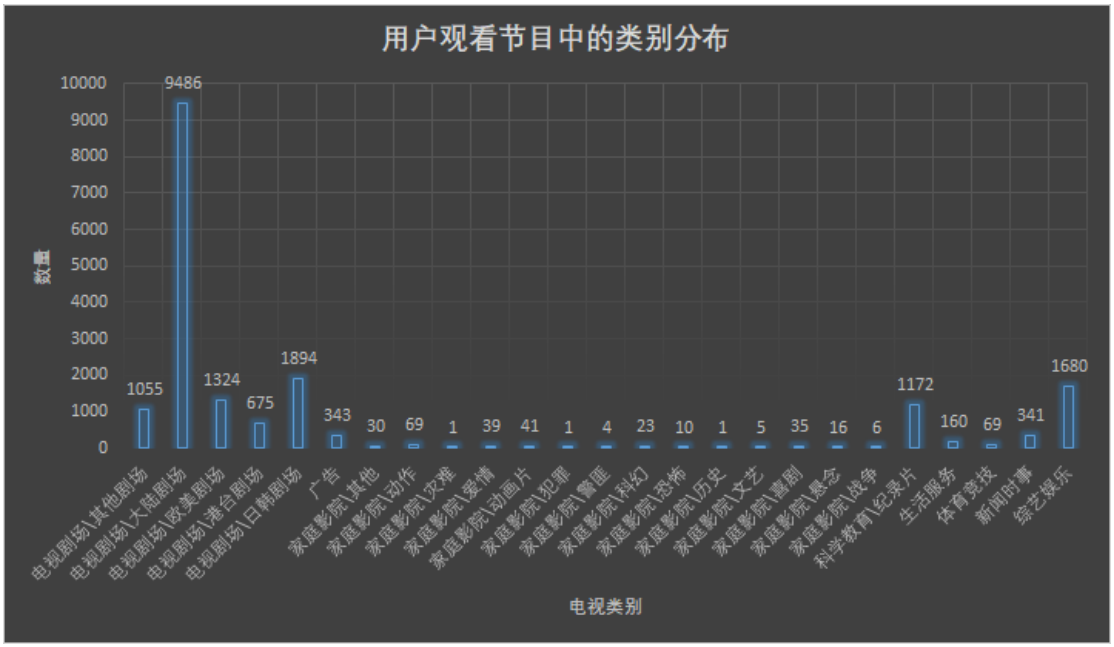


图 4 用户观看节目中的类别分布

2.3.3 基于内容的协同过滤算法与去重分析过程

第一步：根据用户对产品的偏好度，构建相应的用户_产品_偏好度矩阵。

第二步：利用构建的用户_产品矩阵，转置得出产品_用户_偏好度矩阵，计算用户的观看偏好，从而将用户喜好看的电视产品且用户未观看过的电视产品推荐给用户；

2.3.4 用户的收视偏好分析

从用户会看信息表和用户点播信息表中的数据可以知道：用户号与机顶盒设备号一一对应，故两者之间可以相互代替。

利 Excel 对数据进行统计处理与分析，并保存在相对路径 data/

从点播金额，单片点播时长，单片点播目录，频道，节目年代属性，节目导演者，节目主演者等不同的特征角度，分析用户的收视偏好。

2.3.5 产品的推荐方案

程序运行：user_item.py:

input: item_user_values

output: top_k(相似产品中，相似度最大的 k 个产品)

下面展示的是，我们的测试数据的截图，再输入用户的编号后，会直接展示推荐个用户的电视产品，再次输入推荐产品的个数后，就只会展示相似度最高的指定个数的推荐的产品：

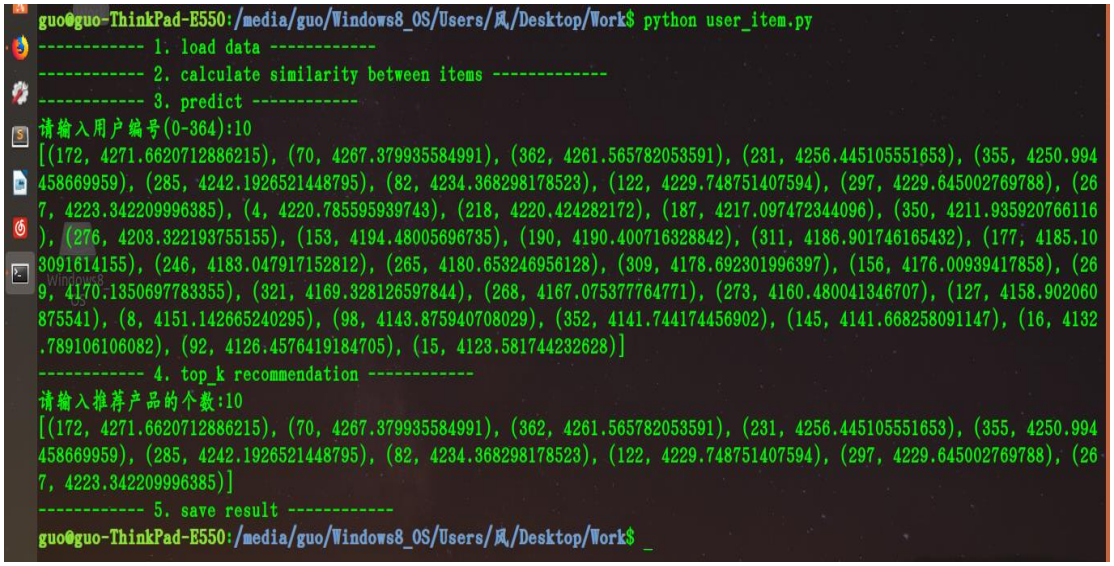
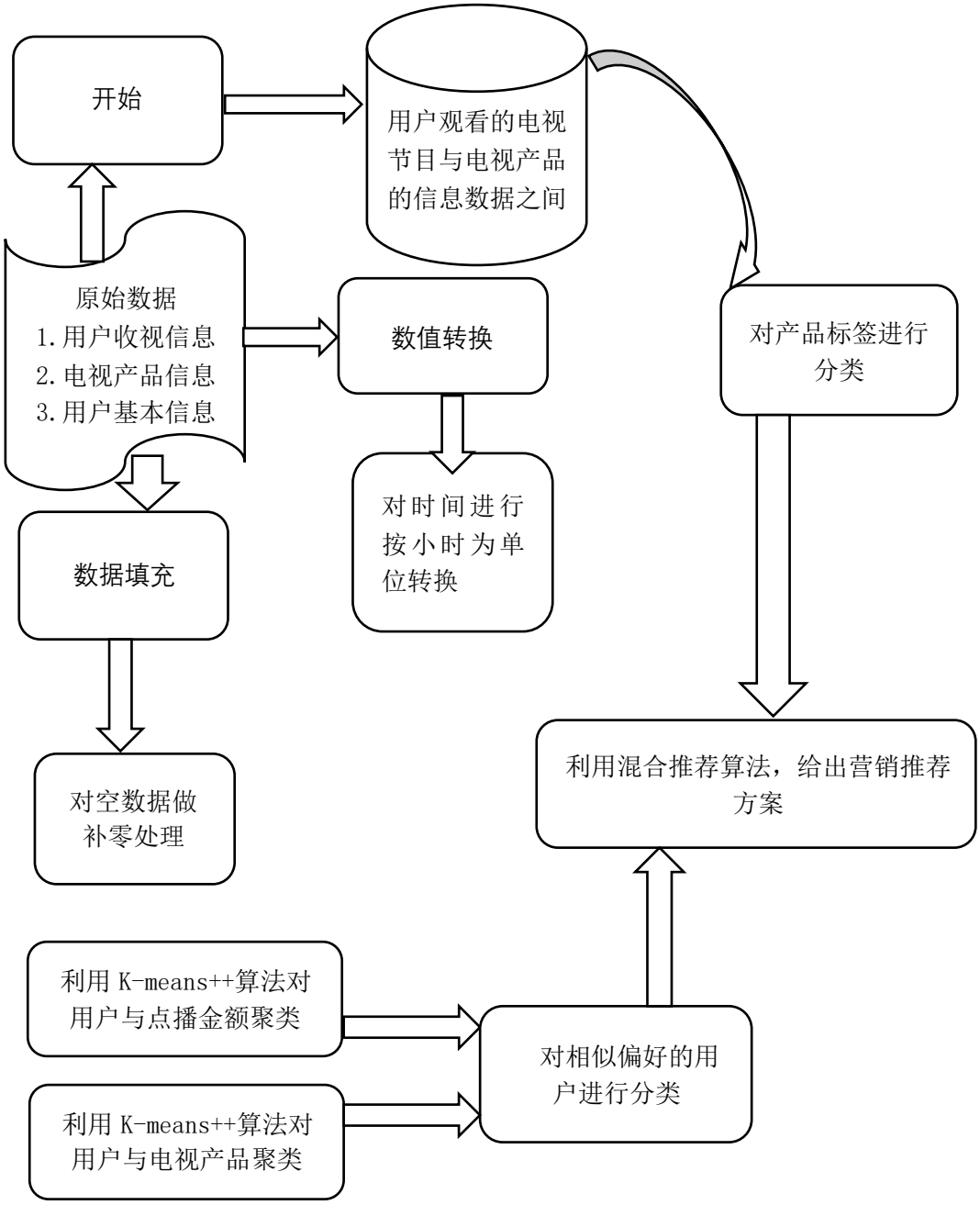


图 5 第一问的电视产品推销方案

2.4 问题 2 分析方法与过程

2.4.1 流程图



2.4.2 数据预处理

节目的偏好度=偏好度（年代）+偏好度（导演）+偏好度（演员）+偏好度（类目）。

从数据表中分离统计出文件：机顶盒设备号_频道_观看时长.csv；机顶盒设备号_点播金额 csv；机顶盒设备号_分类.csv；机顶盒设备号_分类_时长.csv；产品_类目.csv；产品_金额.csv；产品_集数.csv；产品_出品年代.csv；

产品_导演.csv；产品_演员.csv；以及 user_item_values 矩阵

2.4.3 K-means++算法对用户的分析过程

第一：对用户进行分类

2.4.4 基于用户的协同过滤算法分析过程

第一步：对具有相似偏好的用户给出合适的电视产品的推荐方案；

程序运行：user_user_item.py

input: user_item_values

output: top_k（相似度最高的 k 个产品）

输入一个用户的编号，得出与此用户爱好相似的所有用户及其喜爱的电视产品，再次输入推荐相似用户的人数后，便会展示与之相似度最高的对应的人数及其电视产品，截图如下所示：

图 6 第二问的电视产品推销方案（1）

图 7 第二问的电视产品推销方案（2）

第二步：对具有相似消费记录的用户给出合适的电视套餐的推荐方案。

分析用户与电视产品年代之间的关系，数据分析如下所示：

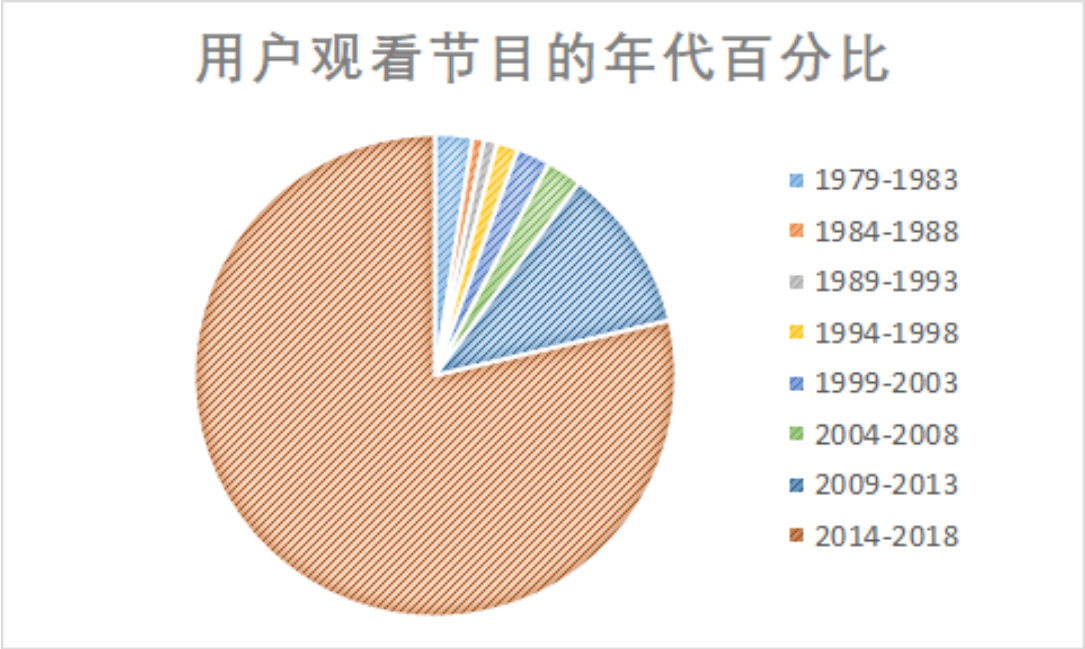


图 8 用户观看节目的年代百分比

分析用户与观看的节目导演出现情况之间的关系，数据分析如下所示：

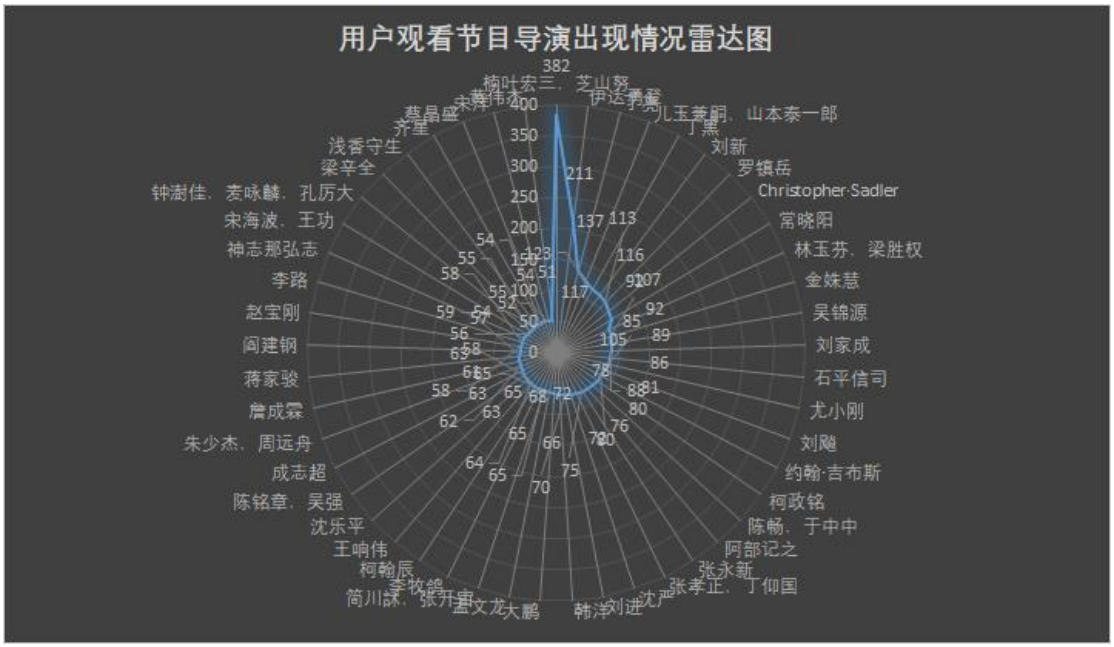


图 9 用户观看节目导演出现情况雷达图

分析用户与观看的节目主演出现情况之间的关系，数据分析如下所示：

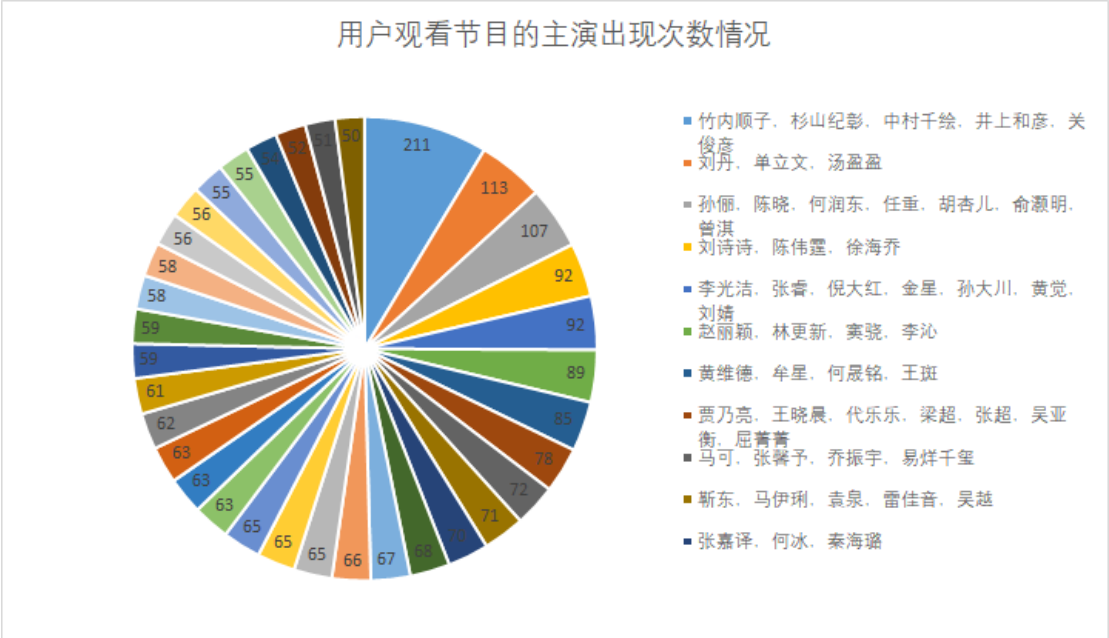


图 10 用户观看节目的主演出现次数

分析用户与观看节目的类比之间的关系，数据分析如下所示：

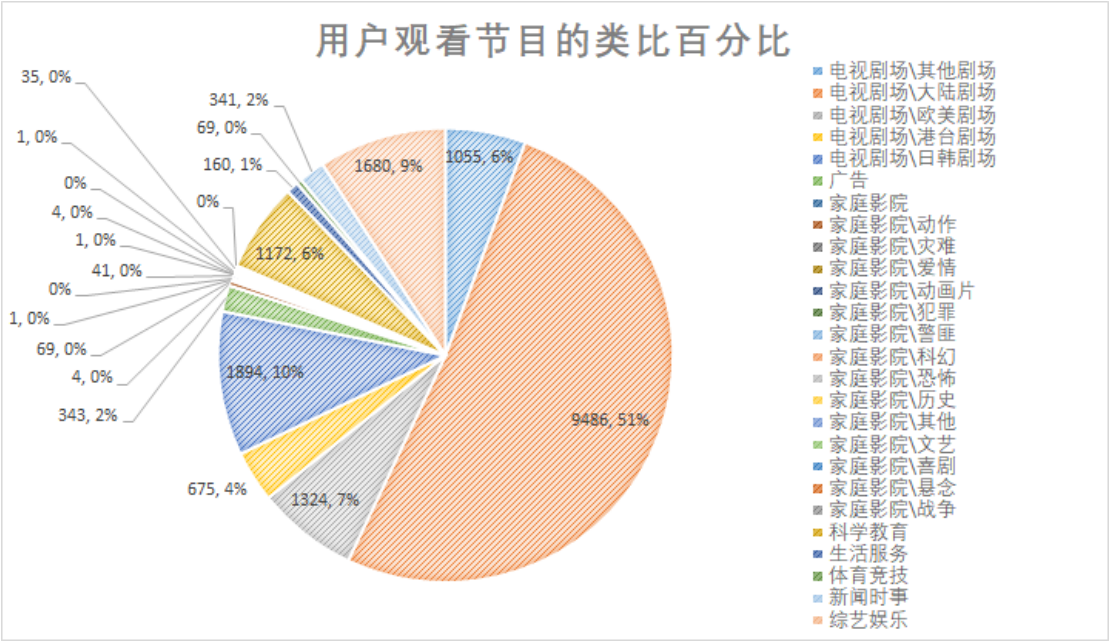


图 11 用户与观看节目的类比百分比

2.4.5 聚类中心分析结果

K-means++聚类的算法步骤如下：

（1）在数据集中随机选择一个样本点作为第一个初始化的聚类中心；

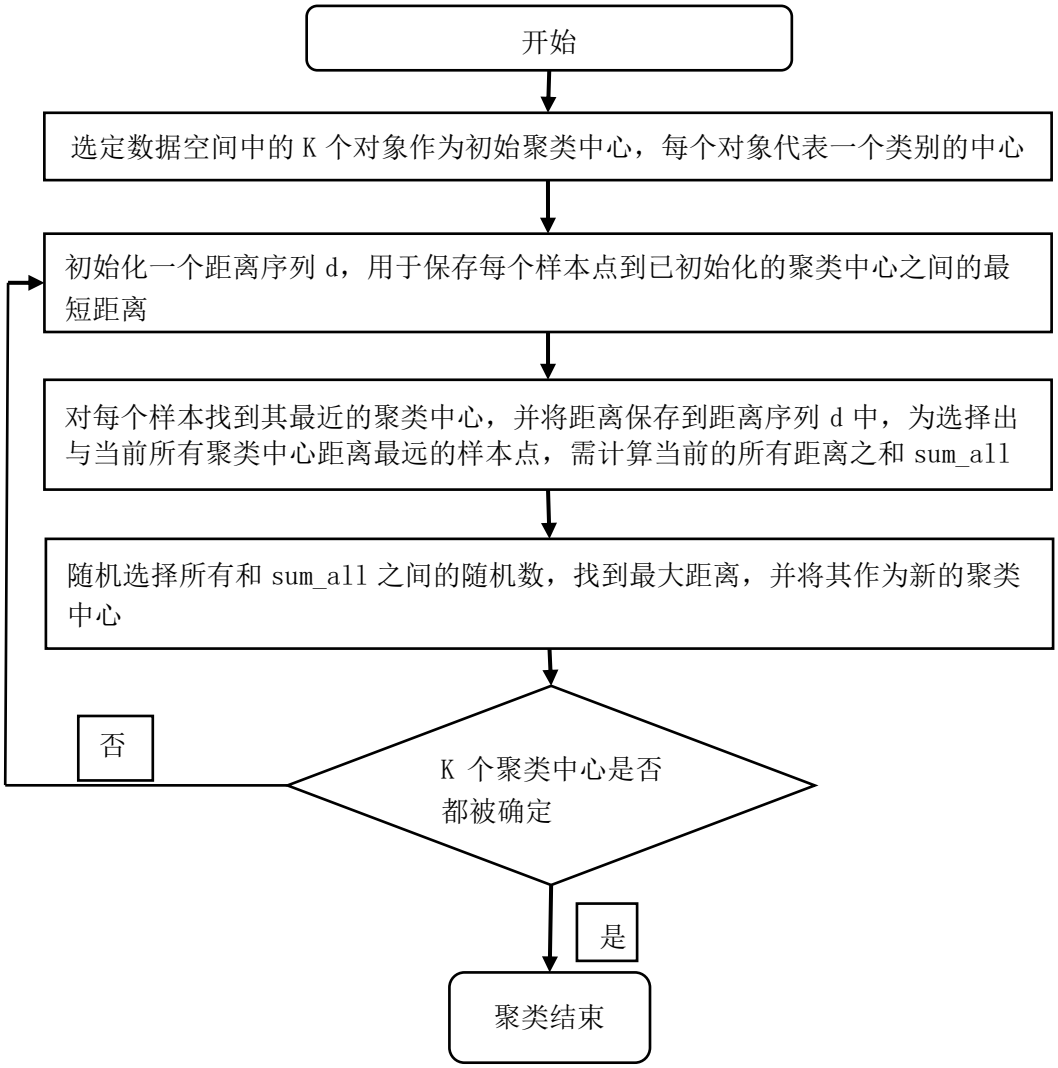
（2）选择出其余的聚类中心：

①计算样本中每一个样本点与已经初始化的聚类中心之间的距离，并选择其中最短的距离，即为d_i；

②以概率选择距离最大的样本最为新的聚类中心，重复上述过程，直到 k 个聚类中心都被确定。

（3）对 k 个初始化的聚类中心，利用 K-means++算法计算最终的聚类中心。

K-means++聚类的算法流程图如下：



2.4.6 对相似偏好的用户的分析

通过题目一中所得的数据，可以构建用户_用户矩阵，然后利用基于用户的协同过滤算法，计算用户之间的相似性。选取相似度最高的 10 名用户作为推荐产品推荐用户。

2.4.7 对产品分类打包的分析

通过问题一中的数据特征提取后，我们可以得到不同角度的用户分组信息，而不同角度的用户分组依据即可作为产品的类别，从而为产品打上时长，类别，点播金额等角度的不同标签。

2.4.8 产品的推荐方案

先输入用户的编号，便会个此用户展示出所有的电视产品的推荐，再通过对前十名相似用户的偏好分析，再次进行相似度计算，为不同偏好的用户进行个性化的打包产品推荐，截图如下所示：

3 参考文献

[1]赵志勇，Python 机器学习算法，2017 年 7 月

[2]Tony Ojeda Sean Patrick Murphy Benjamin Bengfort Abhijit Dasgupta 著 郝志恒 王佳玮 谢时光 刘梦馨 译，数据科学实战手册（R+Python）

[3]张良均 陈俊德 刘名军 陈荣 著，数据挖掘使用案例分析

4 附录

图表目录：

程序代码：

file.py //原始数据读取文件

```
#coding:utf-8

import pandas as pd

file_path="Bdata/附件1：用户收视信息.xlsx"
data1_1=pd.read_excel(file_path,u"用户收视信息",encoding='gbk')
data1_2=pd.read_excel(file_path,u"用户回看信息")
data1_3=pd.read_excel(file_path,u"用户点播信息")
data1_4=pd.read_excel(file_path,u"用户单片点播信息")

data_2=pd.read_csv("Bdata/附件2：电视产品信息.csv",encoding='gbk')
```

```
data_3=pd.read_csv("Bdata/附件3：用户基本信息.csv",encoding='gbk')
```

Kmeans++.py // kmeans 算法模型

```
#!/usr/bin/python
#encoding:utf-8


import numpy as np
import pandas as pd

from file import data1_3,data1_4,data_2,data_3

column_names=[u"正题名",u"导演",u"演员",u"出品年代",u"分类名称",u"连续剧分类"]
l=len(column_names)

data=data_2[column_names]
data=data.replace(to_replace='无',value=np.nan)
data=data.dropna(how='any')

#导入train_test_split对数据进行分割
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(data[column_names[1:l-1]],data[column_names[l-1]],test_size=0.25,random_state=33)
print "训练样本的类别和数量"
print y_train.value_counts()

print "测试样本的类别和数量"
print y_test.value_counts()


#导入KMeans模型
from sklearn.cluster import KMeans
#初始化，设置聚类中心数量为10
kmeans=KMeans(10, init='k-means++', algorithm='auto')#full, elkan
kmeans.fit(x_train)
#逐条判断每条数据所属的聚类中心
y_predict=kmeans.predict(x_test)

label_pred = kmeans.labels_ #获取聚类标签
print "聚类标签\n",label_pred
print "聚类标签长度:",len(label_pred)
centroids = kmeans.cluster_centers_ #获取聚类中心
```

```
print "聚类中心\n",centroids
inertia = kmeans.inertia_ # 获取聚类准则的总和
print "聚类准则的总和\n",inertia

#使用ARI进行算法的聚类性能评估
from sklearn import metrics
print "ARI进行算法的聚类性能评
估:",metrics.adjusted_rand_score(y_test, y_predict)

#模型保存
from sklearn.externals import joblib
joblib.dump(kmeans,  'data1/doc_cluster.pkl')
kmeans = joblib.load('data1/doc_cluster.pkl')
clusters = kmeans.labels_.tolist()
```

type.py //实现分类的文件

```
#coding:utf-8

import math
import codecs
import random

#k-means和k-means++聚类，第一列是label标签，其它列是数值型数据
class KMeans:

    #一列的中位数
    def getColMedian(self,colList):
        tmp = list(colList)
        tmp.sort()
        alen = len(tmp)
        if alen % 2 == 1:
            return tmp[alen // 2]
        else:
            return (tmp[alen // 2] + tmp[(alen // 2) - 1]) / 2

    #对数值型数据进行归一化，使用绝对标准分[绝对标准差->asd=sum(x-
u)/len(x),x的标准分->(x-u)/绝对标准差，u是中位数]
    def colNormalize(self,colList):
        median = self.getColMedian(colList)
        asd = sum([abs(x - median) for x in colList]) / len(colList)
        result = [(x - median) / asd for x in colList]
        return result
```

```
    '''
    1.读数据
    2.按列读取
    3.归一化数值型数据
    4.随机选择k个初始化中心点
    5.对数据离中心点距离进行分配
    '''
    def __init__(self,filePath,k):
        self.data={}#原始数据
        self.k=k#聚类个数
        self.iterationNumber=0#迭代次数
        #用于跟踪在一次迭代改变的点
        self.pointsChanged=0
        #误差平方和
        self.SSE=0
        line_1=True
        #bianmaxiugai
        with codecs.open(filePath,'r','gbk') as f:
            for line in f:
                 # 第一行为描述信息
                if line_1:
                    line_1=False
                    header=line.split(',')
                    self.cols=len(header)
                    self.data=[[] for i in range(self.cols)]
                else:
                    instances=line.split(',')
                    column_0=True
                    for ins in range(self.cols):
                        if column_0:
                            self.data[ins].append(instances[ins])# 0列数
据
                            column_0=False
                        else:

    self.data[ins].append(float(instances[ins]))# 数值列
        self.dataSize=len(self.data[1])#多少实例
        self.memberOf=[-1 for x in range(self.dataSize)]

        #归一化数值列
        for i in range(1,self.cols):
            self.data[i]=self.colNormalize(self.data[i])

        #随机从数据中选择k个初始化中心点
```

```
        random.seed()
        #1.下面是kmeans随机选择k个中心点
        #self.centroids=[[self.data[i][r] for i in range(1,self.cols)]
        #                for r in
random.sample(range(self.dataSize),self.k)]
        #2.下面是kmeans++选择K个中心点
        self.selectInitialCenter()

        self.assignPointsToCluster()

    #离中心点距离分配点，返回这个点属于某个类别的类型
    def assignPointToCluster(self,i):
        min=10000
        clusterNum=-1
        for centroid in range(self.k):
            dist=self.distance(i,centroid)
            if dist<min:
                min=dist
                clusterNum=centroid
        #跟踪改变的点
        if clusterNum!=self.memberOf[i]:
            self.pointsChanged+=1
        #误差平方和
        self.SSE+=min**2
        return clusterNum


    #将每个点分配到一个中心点，memberOf=[0,1,0,0,...]，0和1是两个类
别，每个实例属于的类别
    def assignPointsToCluster(self):
            self.pointsChanged=0
            self.SSE=0
            self.memberOf=[self.assignPointToCluster(i) for i in
range(self.dataSize)]

    # 欧氏距离,d(x,y)=math.sqrt(sum((x-y)*(x-y)))
    def distance(self,i,j):
        sumSquares=0
        for k in range(1,self.cols):
            sumSquares+=(self.data[k][i]-self.centroids[j][k-1])**2
        return math.sqrt(sumSquares)

    #利用类中的数据点更新中心点，利用每个类中的所有点的均值
    def updateCenter(self):
```

```
        members=[self.memberOf.count(i) for i in
range(len(self.centroids))]#得到每个类别中的实例个数
        self.centroids=[
            [sum([self.data[k][i] for i in range(self.dataSize)
                if self.memberOf[i]==centroid])/members[centroid]
            for k in range(1,self.cols)]
            for centroid in range(len(self.centroids))]

    '''迭代更新中心点（使用每个类中的点的平均坐标），
    然后重新分配所有点到新的中心点，直到类中成员改变的点小于1%(只有不
到1%的点从一个类移到另一类中)
    '''
    def cluster(self):
        done=False
        while not done:
            self.iterationNumber+=1#迭代次数
            self.updateCenter()
            self.assignPointsToCluster()
            #少于1%的改变点，结束
            if float(self.pointsChanged)/len(self.memberOf)<0.01:
                done=True
        print("误差平方和（SSE）: %f" % self.SSE)

    #打印结果
    def printResults(self):
        for centroid in range(len(self.centroids)):
            print('\n\nCategory %i\n=========' % centroid)
            for name in [self.data[0][i] for i in range(self.dataSize)
                if self.memberOf[i]==centroid]:
                print(name)

    #kmeans++方法与kmeans方法的区别就是初始化中心点的不同
    def selectInitialCenter(self):
        centroids=[]
        total=0
        #首先随机选一个中心点
        firstCenter=random.choice(range(self.dataSize))
        centroids.append(firstCenter)
        #选择其它中心点，对于每个点找出离它最近的那个中心点的距离
        for i in range(0,self.k-1):
            weights=[self.distancePointToClosestCenter(x,centroids)
                   for x in range(self.dataSize)]
            total=sum(weights)
            #归一化0到1之间
```

```
            weights=[x/total for x in weights]

            num=random.random()
            total=0
            x=-1
            while total<num:
                x+=1
                total+=weights[x]
            centroids.append(x)
        self.centroids=[[self.data[i][r] for i in range(1,self.cols)]
for r in centroids]

    def distancePointToClosestCenter(self,x,center):
        result=self.eDistance(x,center[0])
        for centroid in center[1:]:
            distance=self.eDistance(x,centroid)
            if distance<result:
                result=distance
        return result

    #计算点i到中心点j的距离
    def eDistance(self,i,j):
        sumSquares=0
        for k in range(1,self.cols):
            sumSquares+=(self.data[k][i]-self.data[k][j])**2
        return  math.sqrt(sumSquares)

if __name__=='__main__':
    #filePath='data/1-1设备号-观看时长.csv'
    filePath="data/1-4设备号-观看时长.csv"


    kmeans=KMeans(filePath,5)
    kmeans.cluster()  #误差平方和（SSE）
    kmeans.printResults()   #用户分类输出
```

user_item.py //基于问题一的基于内容的协同过滤算法的实现

```
#coding:utf-8
#基于内容的协同过滤算法：根据用户的观看记录向用户进行相关推荐
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#相似度矩阵的计算
```

```
def similarity(data):
    """计算矩阵中任意两行之间的相似度
    imput:data(mat) :任意矩阵
    output:w(mat)  :任意两行之间的相似度"""
    m=np.shape(data)[0]   #用户的数量
    #初始化相似度矩阵
    w=np.mat(np.zeros((m,m)))
    for i in xrange(m):
        for j in xrange(i,m):
            if j!=i:
                #计算任意两行之间的相似度
                w[i,j]=cos_sim(data[i,],data[j,])
                w[j,i]=w[i,j]
            else:
                w[i,j]=0
    return w

#导入用户-商品数据
def load_data(file_path):
    """导入用户-商品数据
       input: file_path: 数据文件的路径
       output: data(mat):用户商品数据"""
    f=open(file_path)
    data=[]

    for line in f.readlines():
        lines=line.strip().split(",")
        tmp=[]
        for x in lines:
            if x!="\t":
                tmp.append(float(x))  #直接存储用户对商品的打分
            else:
                tmp.append(0)
        data.append(tmp)
    f.close()
    return np.mat(data)

def item_based_recommend(data, w, user):
    '''基于商品相似度为用户user推荐商品
    input:  data(mat):商品用户矩阵
            w(mat):商品与商品之间的相似性
            user(int):用户的编号
    output: predict(list):推荐列表
    '''
```

```
    m, n = np.shape(data) # m:商品数量 n:用户数量
    interaction = data[:,user].T # 用户user的互动商品信息

    # 1、找到用户user没有互动的商品
    not_inter = []
    for i in xrange(n):
        if interaction[0, i] == 0: # 用户user未打分项
            not_inter.append(i)

    # 2、对没有互动过的商品进行预测
    predict = {}
    for x in not_inter:
        item = np.copy(interaction) # 获取用户user对商品的互动信息
        for j in xrange(m): # 对每一个商品
            if item[0, j] != 0: # 利用互动过的商品预测
                if x not in predict:
                    predict[x] = w[x, j] * item[0, j]
                else:
                    predict[x] = predict[x] + w[x, j] * item[0, j]
    # 按照预测的大小从大到小排序
    return sorted(predict.items(), key=lambda d:d[1], reverse=True)

def top_k(predict, k):
    '''为用户推荐前k个商品
    input:  predict(list):排好序的商品列表
            k(int):推荐的商品个数
    output: top_recom(list):top_k个商品
    '''
    top_recom = []
    len_result = len(predict)
    if k >= len_result:
        top_recom = predict
    else:
        for i in xrange(k):
            top_recom.append(predict[i])
    return top_recom

def cos_sim(x, y):
    '''余弦相似性
    input:  x(mat):以行向量的形式存储，可以是用户或者商品
            y(mat):以行向量的形式存储，可以是用户或者商品
    output: x和y之间的余弦相似度
    '''
    numerator = x * y.T  # x和y之间的额内积
```

```
    denominator = np.sqrt(x * x.T) * np.sqrt(y * y.T)
    return (numerator / denominator)[0, 0]

if __name__ == "__main__":
    # 1、导入用户商品数据
    print "------------ 1. load data ------------"
    file_path="data1/user_item_values.txt"
    data = load_data(file_path)
    # 将用户商品矩阵转置成商品用户矩阵
    data = data.T
    print data

    # 2、计算商品之间的相似性
    print "------------ 2. calculate similarity between items -------
------"
    w = similarity(data)
    # 3、利用用户之间的相似性进行预测评分
    print "------------ 3. predict ------------"
    lenth=data.shape[1]
    num=input("请输入用户编号(0-"+str(lenth)+"):")
    predict = item_based_recommend(data, w, num)
    print predict
    # 4、进行Top-K推荐
    print "------------ 4. top_k recommendation ------------"
    item_num=input("请输入推荐产品的个数:")
    top_recom = top_k(predict, item_num)
    print top_recom

    print "------------ 5. save result ------------"
    np.savetxt('data2/top_result1.txt',top_recom,fmt='%f',delimiter='
',newline='\r\n')

    #画图
```

user_user_item.py //基于问题二的基于用户的协同过滤算法的实现

```
#encoding:utf-8
from user_item import load_data,similarity,top_k
import numpy as np
def user_based_recommend(data,w,user):
    """基于用户相似性为用户use进行推荐
    input:data(mat)  用户商品矩阵
          w(mat)  用户之间的相似度
          user(int)  用户的编号
```

```
    output:predict(list)  推荐列表
    """
    m,n=np.shape(data)
    interction=data[user,]  #用户user与商品信息
    #找到用户user没有互动过的商品
    not_inter=[]
    for i in xrange(n):
        if interction[0,i]==0 : #没有互动的商品
            not_inter.append(i)

    #对没有互动过的商品进行预测
    predict={}
    for x in not_inter:
        item=np.copy(data[:,x])  #找到所有用户对商品x的互动信息
        for i in xrange(m):  #对每一个用户
            if x not in predict:
                predict[x]=w[user,i]*item[i,0]
            else:
                predict[x]=predict[x]+w[user,i]*item[i,0]
    #排序
    return sorted(predict.items(),key=lambda d:d[1],reverse=True)

if __name__ == "__main__":
    # 1、导入用户商品数据
    print "------------ 1. load data ------------"
    file_path="data1/user_item_values.txt"
    data = load_data(file_path)


    # 2、计算商品之间的相似性
    print "------------ 2. calculate similarity between items -------
------"
    w = similarity(data)
    # 3、利用用户之间的相似性进行预测评分
    print "------------ 3. predict ------------"
    lenth=data.shape[0]
    num=input("请输入用户编号(0-"+str(lenth)+"):")
    predict = user_based_recommend(data, w, num)
    print (predict,"\n")
    # 4、进行Top-K推荐
    print "------------ 4. top_k recommendation ------------"
    item_num=input("请输入推荐相似用户的人数:")
    top_recom = top_k(predict, item_num)
    print top_recom
```

```
    print "------------ 5. save result ------------"
    np.savetxt('data2/top_result1.txt',top_recom,fmt='%f',delimiter='
',newline='\r\n')
```