《python数据分析与挖掘实战》笔记第5章

发布于2021-03-02 20:54:41阅读 3720

**文章目录**

* + [第5章：挖掘建模](https://cloud.tencent.com/developer?from=10680)
    - [5.1、分类与预测](https://cloud.tencent.com/developer?from=10680)
      * [5.1.1、实现过程](https://cloud.tencent.com/developer?from=10680)
      * [5.1.2、常用的分类与预测算法](https://cloud.tencent.com/developer?from=10680)
      * [5.1.3、回归分析](https://cloud.tencent.com/developer?from=10680)
      * [5.1.4、决策树](https://cloud.tencent.com/developer?from=10680)
      * [5.1.5、人工神经网络](https://cloud.tencent.com/developer?from=10680)
      * [5.1.7、 Python分类预测模型特点](https://cloud.tencent.com/developer?from=10680)
    - [5.2、聚类分析](https://cloud.tencent.com/developer?from=10680)
      * [5.2.1、常用聚类分析算法](https://cloud.tencent.com/developer?from=10680)
      * [5.2.2、 K-Means聚类算法](https://cloud.tencent.com/developer?from=10680)
      * [5.2.3、聚类分析算法评价](https://cloud.tencent.com/developer?from=10680)
      * [5.2.4、 Python主要聚类分析算法](https://cloud.tencent.com/developer?from=10680)
    - [5.3、关联规则](https://cloud.tencent.com/developer?from=10680)
      * [5.3.1、常用关联规则算法](https://cloud.tencent.com/developer?from=10680)
      * [5.3.2、 Apriori 算法](https://cloud.tencent.com/developer?from=10680)
    - [5.4、时序模式](https://cloud.tencent.com/developer?from=10680)

**第5章：挖掘建模**

**5.1、分类与预测**

分类和预测是预测问题的两种主要类型，分类主要是预测分类标号（离散属性），而预测 主要是建立连续值函数模型，预测给定自变量对应的因变量的值。

**5.1.1、实现过程**

(1）分类 分类是构造一个分类模型，输入样本的属性值，输岀对应的类别，将每个样本映射到预先定义好的类别。

(2)预测 预测是指建立两种或两种以上变量间相互依赖的函数模型，然后进行预测或控制。

**5.1.2、常用的分类与预测算法**

| 算法名称 | 算法描述 |
| --- | --- |
| 回归分析 | 回归分析是确定预测属性(数值型)与其他变量间相互依赖的定量关系最常用的统计学 方法。包括线性回归、非线性回归、Logistic回归、岭回归、主成分回归、偏最小二乘回 归等模型 |
| 决策树 | 决策树采用自顶向下的递归方式，在内部节点进行属性值的比较，并根据不同的属性值 从该节点向下分支，最终得到的叶节点是学习划分的类 |
| 人工神经网络 | 人工神经网络是一种模仿大脑神经网络结构和功能而建立的信息处理系统，表示神经网 络的输入与输出变量之间关系的模型 |
| 贝叶斯网络 | 贝叶斯网络又称信度网络，是Bayes方法的扩展，是目前不确定知识表达和推理领域最 有效的理论模型之一 |
| 支持向量机 | 支持向量机是一种通过某种非线性映射，把低维的非线性可分转化为高维的线性可分， 在高维空间进行线性分析的算法 |

**5.1.3、回归分析**

回归分析是通过建立模型来研究变量之间相互关系的密切程度、结构状态及进行模型 预测的一种有效工具，在工商管理、经济、社会、医学和生物学等领域应用十分广泛。

常用的回归模型见表5-2

* 表5-2主要回归模型分类

| 回归模型名称 | 适用条件 | 算法描述 |
| --- | --- | --- |
| 线性回归 | 因变量与自变量是线性 关系 | 对一个或多个自变量和因变量之间的线性关系进行建模，可用最 小二乘法求解模型系数 |
| 非线性回归 | 因变量与自变量之间不 都是线性关系 | 对一个或多个自变量和因变量之间的非线性关系进行建模。如果 非线性关系可以通过简单的函数变换转化成线性关系，用线性回归 的思想求解；如果不能转化，用非线性最小二乘方法求解 |
| Logistic 回归 | 因变量一般有1和0（是 否）两种取值 | 是广义线性回归模型的特例，利用Logistic函数将因变量的取值范 围控制在0和1之间，表示取值为1的概率 |
| 岭回归 | 参与建模的自变量之间具有多重共线性 | 是一种改进最小二乘估计的方法 |
| 主成分回归 | 参与建模的自变量之间具有多重共线性 | 主成分回归是根据主成分分析的思想提出来的.是对最小二乘法 的一种改进，它是参数估计的一种有偏估计。可以消除自变量之间 的多重共线性 |

代码清单5-1\_逻辑回归代码

#-\*- coding: utf-8 -\*-

#逻辑回归 自动建模

import pandas as pd

#参数初始化

filename = '../data/bankloan.xls'

data = pd.read\_excel(filename)

x = data.iloc[:,:8].values

y = data.iloc[:,8].values

from sklearn.linear\_model import LogisticRegression as LR

from sklearn.linear\_model import RandomizedLogisticRegression as RLR

rlr = RLR() #建立随机逻辑回归模型，筛选变量

rlr.fit(x, y) #训练模型

get\_support=rlr.get\_support(indices=True) #获取特征筛选结果，也可以通过.scores\_方法获取各个特征的分数

print(u'通过随机逻辑回归模型筛选特征结束。')

print(u'有效特征为：%s' % ','.join(data.columns[get\_support]))

x = data[data.columns[get\_support]].values #筛选好特征

lr = LR() #建立逻辑货柜模型

lr.fit(x, y) #用筛选后的特征数据来训练模型

print(u'逻辑回归模型训练结束。')

print(u'模型的平均正确率为：%s' % lr.score(x, y)) #给出模型的平均正确率，本例为81.4%

复制

递归特征消除的主要思想是反复的构建模型（如SVM或者回归模型）然后选出最好 的（或者最差的）的特征（可以根据系数来选），把选出来的特征放到一边，然后在剩余的 特征上重复这个过程，直到遍历所有特征。这个过程中特征被消除的次序就是特征的排 序。因此，这是一种寻找最优特征子集的贪心算法。Scikit-Leam提供了 RFE包，可以用 于特征消除，还提供了 RFECV,可以通过交叉验证来对特征进行排序。

稳定性选择是一种基于二次抽样和选择算法相结合较新的方法，选择算法可以是回 归、SVM或其他类似的方法。它的主要思想是在不同的数据子集和特征子集上运行特征 选择算法，不断重复，最终汇总特征选择结果。比如，可以统计某个特征被认为是重要特 征的频率（被选为重要特征的次数除以它所在的子集被测试的次数）。在理想情况下，重 要特征的得分会接近100%。稍微弱一点的特征得分会是非0的数，而最无用的特征得分 将会接近于0。Scikit-Leam在随机Lasso和随机逻辑回归中有对稳定性选择的实现。

**5.1.4、决策树**

决策树是一树状结构，它的每一个叶节点对应着一个分类，非叶节点对应着在某个属性 上的划分，根据样本在该属性上的不同取值将其划分成若干个子集。对于非纯的叶节点，多 数类的标号给出到达这个节点的样本所属的类。构造决策树的核心问题是在每一步如何选择 适当的属性对样本做拆分。对一个分类问题，从已知类标记的训练样本中学习并构造出决策 树是一个自上而下，分而治之的过程。

常用的决策树算法见表5-4

| 决策树算法 | 算法描述 |
| --- | --- |
| ID3算法 | 其核心是在决策树的各级节点上，使用信息增益方法作为属性的选择标准.来帮助确定生成 每个节点时所应采用的合适属性 |
| C4.5算法 | C4.5决策树生成算法相对于ID3算法的重要改进是使用信息增益率来选择节点属性。C4.5算 法可以克服1D3算法存在的不足：ID3算法只适用于离散的描述属性，而C4.5算法既能够处理 离散的描述属性，也可以处理连续的描述属性 |
| CART算法 | CART决策树是一种十分有效的非参数分类和回归方法，通过构建树、修剪树、评估树来构 建一个二叉树。当终结点是连续变量时，该树为回归树；当终结点是分类变量，该树为分类树 |

**1. ID3算法简介及基本原理** ID3算法基于信息嫡来选择最佳测试属性。它选择当前样本集中具有最大信息增益值的 属性作为测试属性；样本集的划分则依据测试属性的取值进行，测试属性有多少不同取值 就将样本集划分为多少子样本集，同时决策树上相应于该样本集的节点长出新的叶子节点。 ID3算法根据信息论理论，采用划分后样本集的不确定性作为衡量划分好坏的标准，用信息 增益值度量不确定性：信息增益值越大，不确定性越小。因此，ID3算法在每个非叶节点选 择信息增益最大的属性作为测试属性，这样可以得到当前情况下最纯的拆分，从而得到较小 的决策树。

由于ID3决策树算法釆用了信息增益作为选择测试属性的标准，会偏向于选择取值较多 的，即所谓高度分支属性，而这类層性并不一定是最优的属性。同时ID3决策树算法只能处 理离散属性，对于连续型的属性，在分类前需要对其进行离散化。为了解决倾向于选择高度 分支属性的问题，人们采用信息增益率作为选择测试属性的标准，这样便得到C4.5决策树 算法。此外，常用的决策树算法还有CART算法、SLIQ算法、SPRINT算法和PUBLIC算 法等。

使用Scikit-Leam建立基于信息嫡的决策树模型，如代码清单5-2所示

代码清单5\_2\_决策树算法预测销量高低代码

#-\*- coding: utf-8 -\*-

#使用ID3决策树算法预测销量高低

import pandas as pd

#参数初始化

inputfile = '../data/sales\_data.xls'

data = pd.read\_excel(inputfile, index\_col = u'序号') #导入数据

#数据是类别标签，要将它转换为数据

#用1来表示“好”、“是”、“高”这三个属性，用-1来表示“坏”、“否”、“低”

data[data == u'好'] = 1

data[data == u'是'] = 1

data[data == u'高'] = 1

data[data != 1] = -1

x = data.iloc[:,:3].values.astype(int)

y = data.iloc[:,3].values.astype(int)

from sklearn.tree import DecisionTreeClassifier as DTC

dtc = DTC(criterion='entropy') #建立决策树模型，基于信息熵

dtc.fit(x, y) #训练模型

#导入相关函数，可视化决策树。

#导出的结果是一个dot文件，需要安装Graphviz才能将它转换为pdf或png等格式。

from sklearn.tree import export\_graphviz

from sklearn.externals.six import StringIO

x = pd.DataFrame(x)

with open("tree.dot", 'w') as f:

f = export\_graphviz(dtc, feature\_names = x.columns, out\_file = f)

复制

**5.1.5、人工神经网络**

人工神经网络(Artificial Neural Networks, ANN),是模拟生物神经网络进行信息处 理的一种数学模型。它以对大脑的生理研究成果为基础，其目的在于模拟大脑的某些机理与 机制，实现一些特定的功能。

人工神经网络的学习也称为训练，指的是神经网络在受到外部环境的刺激下调整神经网 络的参数，使神经网络以一种新的方式对外部环境作出反应的一个过程。在分类与预测中， 人工神将网络主要使用有指导的学习方式，即根据给定的训练样本，调整人工神经网络的参 数以使网络输出接近于已知的样本类标记或其他形式的因变量。

使用人工神经网络模型需要确定是各连接的拓扑结构、神经元的特征和学习规则等。目 前，已有近40种人工神经网络模型，常用的用来实现分类和预测的人工神经网络算法见表5-7。

* 表5-7人工神经网络算法

| 算法名称 | 算法描述 |
| --- | --- |
| BP神经网络 | 是一种按误差逆传播算法训练的多层前馈网络，学习算法是δ学习规则，是目前应用最广泛 的神经网络模型之一 |
| LM神经网络 | 是基于梯度下降法和牛顿法结合的多层前馈网络，特点：迭代次数少，收敛速度快，精确度高 |
| RBF径向基 神经网络 | RBF网络能够以任意精度逼近任意连续函数，从输入层到隐含层的变换是非线性的，而从隐 含层到输出层的变换是线性的，特别适合于解决分类问题 |
| FNN模糊神经网络 | FNN模糊神经网络是具有模糊权系数或者输入信号是模糊量的神经网络，是模糊系统与神经 网络相结合的产物，它汇聚了神经网络与模糊系统的优点，集联想、识别、自适应及模糊信息 处理于一体 |
| GMDH神经网络 | GMDH网络也称为多项式网络，它是前馈神经网络中常用的一种用于预测的神经网络。它的 特点是网络结构不固定，而且在训练过程中不断改变 |
| ANFIS自适 应神经网络 | 神经网络镶嵌在一个全部模糊的结构之中，在不知不觉中向训练数据学习，自动产生、修正 并高度概括出最佳的输入与输出变量的隶属函数以及模糊规则；另外，神经网络的各层结构与 参数也都具有了明确的、易于理解的物理意义 |

代码清单5-3神经网络算法预测销量高低

#-\*- coding: utf-8 -\*-

#使用神经网络算法预测销量高低

import pandas as pd

#参数初始化

inputfile = '../data/sales\_data.xls'

data = pd.read\_excel(inputfile, index\_col = u'序号') #导入数据

#数据是类别标签，要将它转换为数据

#用1来表示“好”、“是”、“高”这三个属性，用0来表示“坏”、“否”、“低”

data[data == u'好'] = 1

data[data == u'是'] = 1

data[data == u'高'] = 1

data[data != 1] = 0

x = data.iloc[:,:3].values.astype(int)

y = data.iloc[:,3].values.astype(int)

from keras.models import Sequential

from keras.layers.core import Dense, Activation

model = Sequential() #建立模型

model.add(Dense(input\_dim = 3, output\_dim = 10))

model.add(Activation('relu')) #用relu函数作为激活函数，能够大幅提供准确度

model.add(Dense(input\_dim = 10, output\_dim = 1))

model.add(Activation('sigmoid')) #由于是0-1输出，用sigmoid函数作为激活函数

model.compile(loss = 'binary\_crossentropy', optimizer = 'adam')

#编译模型。由于我们做的是二元分类，所以我们指定损失函数为binary\_crossentropy，以及模式为binary

#另外常见的损失函数还有mean\_squared\_error、categorical\_crossentropy等，请阅读帮助文件。

#求解方法我们指定用adam，还有sgd、rmsprop等可选

model.fit(x, y, nb\_epoch = 1000, batch\_size = 10) #训练模型，学习一千次

yp = model.predict\_classes(x).reshape(len(y)) #分类预测

from cm\_plot import \* #导入自行编写的混淆矩阵可视化函数

cm\_plot(y,yp).show() #显示混淆矩阵可视化结果

复制

cm\_plot.py

# -\*- coding: utf-8 -\*-

def cm\_plot(y, yp):

from sklearn.metrics import confusion\_matrix # 导入混淆矩阵函数

cm = confusion\_matrix(y, yp) # 混淆矩阵

import matplotlib.pyplot as plt # 导入作图库

plt.matshow(cm, cmap=plt.cm.Greens) # 画混淆矩阵图，配色风格使用cm.Greens，更多风格请参考官网。

plt.colorbar() # 颜色标签

for x in range(len(cm)): # 数据标签

for y in range(len(cm)):

plt.annotate(cm[x, y], xy=(x, y), horizontalalignment='center', verticalalignment='center')

plt.ylabel('True label') # 坐标轴标签

plt.xlabel('Predicted label') # 坐标轴标签

return plt

复制

**5.1.7、 Python分类预测模型特点**

* 表5-9常见的模型评价和在Python中的实现

| 模 型 | 模型特点 | 位 于 |
| --- | --- | --- |
| 逻辑回归 | 比较基础的线性分类模型，很多时候是简单有效的选择 | sklean. linear\_model |
| SVM | 强大的模型，可以用来回归、预测、分类等，而根据选取不同的核函 数。模型可以是线性的/非线性的 | sklean.svm |
| 决策树 | 基于“分类讨论、逐步细化”思想的分类模型，模型直观，易解释，如 前面5.1.4节中可以直接给出决策图 | sklean.tree |
| 随机森林 | 思想跟决策树类似，精度通常比决策树要高，缺点是由于其随机性，丧 失了决策树的可解释性 | sklean.ensemble |
| 朴素贝叶斯 | 基于概率思想的简单有效的分类模型，能够给出容易理解的概率解释 | sklean.naive\_bayes |
| 神经网络 | 具有强大的拟合能力，可以用于拟合、分类等，它有很多个增强版本， 如递神经网络、卷积神经网络、自编码器等，这些是深度学习的模型基础 | Keras |

**5.2、聚类分析**

**5.2.1、常用聚类分析算法**

与分类不同，聚类分析是在没有给定划分类别的情况下，根据数据相似度进行样本分组的一种方法。与分类模型需要使用有类标记样本构成的训练数据不同，聚类模型可以建立在无类标记的数据上，是一种非监督的学习算法。聚类的输入是一组未被标记的样本，聚类根据数据自身的距 离或相似度将其划分为若干组，划分的原则是组内距离最小化而组间（外部）距离最大化，如图5-11

表5-10常用聚类方法

| 类 别 | 包括的主要算法 |
| --- | --- |
| 划分（分裂）方法 | K-Means算法（K・平均）、K-MEDOIDS算法（K-中心点）、CLARANS算法（基于选择 的算法） |
| 层次分析方法 | BIRCH算法（平衡迭代规约和聚类）、CURE算法（代表点聚类）、CHAMELEON算法 （动态模型） |
| 基于密度的方法 | DBSCAN算法（基于高密度连接区域）、DENCLUE算法（密度分布函数）、OPTICS算 法（对象排序识别） |
| 基于网格的方法 | STING算法（统计信息网络）、CLIOUE算法（聚类高维空间）、WAVE-CLUSTER算法 （小波变换） |
| 基于模型的方法 | 统计学方法、神经网络方法 |

表5-11常用聚类分析算法

| 算法名称 | 算法描述 |
| --- | --- |
| K-Means | K-均值聚类也称为快速聚类法，在最小化误差函数的基础上将数据划分为预定的类数K。该算法 原理简单并便于处理大量数据 |
| K-中心点 | K-均值算法对孤立点的敏感性，K-中心点算法不采用簇中对象的平均值作为簇中心，而选用簇 中离平均值最近的对象作为簇中心 |
| 系统聚类 | 系统聚类也称为多层次聚类，分类的单位由高到低呈树形结构，且所处的位置越低，其所包含的 对象就越少，但这些对象间的共同特征越多。该聚类方法只适合在小数据量的时候使用，数据量大 的时候速度会非常慢 |

**5.2.2、 K-Means聚类算法**

K-Means算法是典型的基于距离的非层次聚类算法，在最小化误差函数的基础上将 数据划分为预定的类数K，采用距离作为相似性的评价指标，即认为两个对象的距离越近， 其相似度就越大。

**1、算法过程**

1 ）从N个样本数据中随机选取K个对象作为初始的聚类中心。 2） 分别计算每个样本到各个聚类中心的距离，将对象分配到距离最近的聚类中。 3） 所有对象分配完成后，重新计算K个聚类的中心。 4） 与前一次计算得到的K个聚类中心比较，如果聚类中心发生变化，转过程2），否则 转过程5）。 5） 当质心不发生变化时停止并输出聚类结果。

聚类的结果可能依赖于初始聚类中心的随机选择，可能使得结果严重偏离全局最优分 类。实践中，为了得到较好的结果，通常选择不同的初始聚类中心，多次运行K-Means算 法。在所有对象分配完成后，重新计算K个聚类的中心时，对于连续数据，聚类中心取该簇的均值，但是当样本的某些属性是分类变量时，均值可能无定义，可以使用K-众数方法。

**2、数据类型与相似性的度量** （1）连续属性 对于连续属性，要先对各属性值进行零-均值规范，再进行距离的计算。在K-Means聚类算法中，一般需要度量样本之间的距离、样本与簇之间的距离以及簇与簇之间的距离。

(2)文档数据 对于文档数据使用余弦相似性度量，先将文档数据整理成文档-词矩阵格式。

采用K-Means聚类算法，设定聚类个数K为3,最大迭代次数为500次，距离函数取欧 氏距离。 K-Means聚类算法的Python代码如代码清单5-4所示。

代码清单5-4 K-Means聚类算法代码

#-\*- coding: utf-8 -\*-

#使用K-Means算法聚类消费行为特征数据

import pandas as pd

#参数初始化

inputfile = '../data/consumption\_data.xls' #销量及其他属性数据

outputfile = '../tmp/data\_type.xls' #保存结果的文件名

k = 3 #聚类的类别

iteration = 500 #聚类最大循环次数

data = pd.read\_excel(inputfile, index\_col = 'Id') #读取数据

data\_zs = 1.0\*(data - data.mean())/data.std() #数据标准化

def density\_plot(data): #自定义作图函数

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False #用来正常显示负号

p = data.plot(kind='kde', linewidth = 2, subplots = True, sharex = False)

[p[i].set\_ylabel(u'密度') for i in range(k)]

plt.legend()

return plt

if \_\_name\_\_ == '\_\_main\_\_':

from sklearn.cluster import KMeans

model = KMeans(n\_clusters=k, n\_jobs=4, max\_iter=iteration) # 分为k类，并发数4

model.fit(data\_zs) # 开始聚类

# 简单打印结果

r1 = pd.Series(model.labels\_).value\_counts() # 统计各个类别的数目

r2 = pd.DataFrame(model.cluster\_centers\_) # 找出聚类中心

r = pd.concat([r2, r1], axis=1) # 横向连接（0是纵向），得到聚类中心对应的类别下的数目

r.columns = list(data.columns) + [u'类别数目'] # 重命名表头

print(r)

# 详细输出原始数据及其类别

r = pd.concat([data, pd.Series(model.labels\_, index=data.index)], axis=1) # 详细输出每个样本对应的类别

r.columns = list(data.columns) + [u'聚类类别'] # 重命名表头

r.to\_excel(outputfile) # 保存结果

pic\_output = '../tmp/pd\_' #概率密度图文件名前缀

for i in range(k):

density\_plot(data[r[u'聚类类别']==i]).savefig(u'%s%s.png' %(pic\_output, i))

复制

事实上，Scikit-Leam中的K-Means算法仅仅支持欧氏距离，原因在于采用其他的距 离并不一定能够保证算法的收敛性。

**5.2.3、聚类分析算法评价**

聚类分析仅根据样本数据本身将样本分组。其目标是实现组内的对象相互之间是相似的 (相关的)，而不同组中的对象是不同的(不相关的)。组内的相似性越大，组间差别越大，聚类效果就越好。

**5.2.4、 Python主要聚类分析算法**

Python的聚类相关的算法主要在Scikit-Learn中，Python里面实现的聚类主要包括 K-Means聚类、层次聚类、FCM以及神经网络聚类，其主要相关函数如表5.16所示。

| 对象名 | 函数功能 | 所属工具箱 |
| --- | --- | --- |
| KMeans | K均值聚类 | sklean.cluster |
| AffinityPropagation | 吸引力传播聚类，2007年提出，几乎优于所有其他方法，不需要 指定聚类数，但运行效率较低 | sklean.cluster |
| MeanShift | 均值漂移聚类算法 | sklean.cluster |
| SpectralClustering | 谱聚类，具有效果比K均值好，速度比K均值快等特点 | sklean.cluster |
| AgglomerativeClustering | 层次聚类，给出一棵聚类层次树 | sklean.cluster |
| DBSCAN | 具有噪声的基于密度的聚类方法 | sklean.cluster |
| BIRCH | 综合的层次聚类算法，可以处理大规模数据的聚类 | sklean.cluster |

这些不同模型的使用方法是大同小异的，基本都是先用对应的函数建立模型，然后 用.fit()方法来训练模型，训练好之后，就可以用.label\_方法给出样本数据的标签，或者 用.predict()方法预测新的输入的标签。

此外，Scipy库也提供了一个聚类子库scipy.cluster，里边提供了一些聚类算法，如层次 聚类等，但没有Scikit-Lean那么完善和丰富。scipy.cluster的好处是它的函数名和功能基本 跟Python是一一对应的(Scpiy致力于让Python称为Python般强大)，如层次聚类的linkage, dendrogram等，因此已经熟悉Python的朋友，可以尝试使用Scipy提供的聚类库

**下面介绍一个聚类结果可视化的工具——TSNE**

TSNE 是 Laurens van der Maaten 和 Geoffrey Hintton 在 2008 年提出的，它的定位是高维 数据的可视化。我们总喜欢能够直观地展示研究结果，聚类也不例外。然而，通常来说输入 的特征数是高维的(大于3维)，一般难以直接以原特征对聚类结果进行展示。而TSNE提供 了一种有效的数据降维方式，让我们可以在2维或者3维的空间中展示聚类结果。

下面我们用TSNE对上述KMeans聚类的结果以二维的方式展示出来。

代码清单5-5 用TSNE进行数据降维并展示聚类结果

#-\*- coding: utf-8 -\*-

#接k\_means.py，用TSNE进行数据降维并展示聚类结果

import pandas as pd

#参数初始化

inputfile = '../data/consumption\_data.xls' #销量及其他属性数据

outputfile = '../tmp/data\_type.xls' #保存结果的文件名

k = 3 #聚类的类别

iteration = 500 #聚类最大循环次数

data = pd.read\_excel(inputfile, index\_col = 'Id') #读取数据

data\_zs = 1.0\*(data - data.mean())/data.std() #数据标准化

if \_\_name\_\_ == '\_\_main\_\_':

from sklearn.cluster import KMeans

model = KMeans(n\_clusters = k, n\_jobs = 4, max\_iter = iteration) #分为k类，并发数4

model.fit(data\_zs) #开始聚类

#简单打印结果

r1 = pd.Series(model.labels\_).value\_counts() #统计各个类别的数目

r2 = pd.DataFrame(model.cluster\_centers\_) #找出聚类中心

r = pd.concat([r2, r1], axis = 1) #横向连接（0是纵向），得到聚类中心对应的类别下的数目

r.columns = list(data.columns) + [u'类别数目'] #重命名表头

print(r)

#详细输出原始数据及其类别

r = pd.concat([data, pd.Series(model.labels\_, index = data.index)], axis = 1) #详细输出每个样本对应的类别

r.columns = list(data.columns) + [u'聚类类别'] #重命名表头

r.to\_excel(outputfile) #保存结果

# 用TSNE进行数据降维并展示聚类结果

from sklearn.manifold import TSNE

tsne = TSNE()

tsne.fit\_transform(data\_zs) #进行数据降维

tsne = pd.DataFrame(tsne.embedding\_, index = data\_zs.index) #转换数据格式

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False #用来正常显示负号

#不同类别用不同颜色和样式绘图

d = tsne[r[u'聚类类别'] == 0]

plt.plot(d[0], d[1], 'r.')

d = tsne[r[u'聚类类别'] == 1]

plt.plot(d[0], d[1], 'go')

d = tsne[r[u'聚类类别'] == 2]

plt.plot(d[0], d[1], 'b\*')

plt.show()

复制

**5.3、关联规则**

**5.3.1、常用关联规则算法**

表5-17常用关联规则算法

| 算法名称 | 算法描述 |
| --- | --- |
| Apriori | 关联规则最常用也是最经典的挖掘频繁项集的算法，其核心思想是通过连接产生候选项及其支持 度然后通过剪枝生成频繁项集 |
| FP-Tree | 针对Apriori算法的固有的多次扫描事务数据集的缺陷，提出的不产生候选频繁项集的方法。 Apriori和FP-Tree都是寻找频繁项集的算法 |
| Eclat算法 | Eclat算法是一种深度优先算法，采用垂直数据表示形式，在概念格理论的基础上利用基于前缀 的等价关系将搜索空间划分为较小的子空间 |
| 灰色关联法 | 分析和确定各因素之间的影响程度或是若千个子因素(子序列)对主因素(母序列)的贡献度而 进行的一种分析方法 |

**5.3.2、 Apriori 算法**

Apriori 算法是最经典的挖掘频繁项集的算法，第一次实现了在[大数据](https://cloud.tencent.com/solution/bigdata?from=10680)集上可行的关联规则提取，其 核心思想是通过连接产生候选项与其支持度，然后通过剪枝生成频繁项集。

代码清单5-6 Apriori算法调用代码

#-\*- coding:utf-8 -\*-

#使用Apriori算法挖掘菜品订单关联规则

from \_\_future\_\_ import print\_function

import pandas as pd

from apriori import \* #导入自行编写的apriori函数

inputfile = '../data/menu\_orders.xls'

outputfile = '../tmp/apriori\_rules.xls' #结果文件

data = pd.read\_excel(inputfile, header = None)

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)]) #转换0-1矩阵的过渡函数

# b = map(ct, data.as\_matrix()) #用map方式执行

b = map(ct, data.values) #用map方式执行

data = pd.DataFrame(list(b)).fillna(0) #实现矩阵转换，空值用0填充

print(u'\n转换完毕。')

del b #删除中间变量b，节省内存

support = 0.2 #最小支持度

confidence = 0.5 #最小置信度

ms = '---' #连接符，默认'--'，用来区分不同元素，如A--B。需要保证原始表格中不含有该字符

find\_rule(data, support, confidence, ms).to\_excel(outputfile) #保存结果

复制

apriori.py

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import print\_function

import pandas as pd

#自定义连接函数，用于实现L\_{k-1}到C\_k的连接

def connect\_string(x, ms):

x = list(map(lambda i:sorted(i.split(ms)), x))

l = len(x[0])

r = []

for i in range(len(x)):

for j in range(i,len(x)):

if x[i][:l-1] == x[j][:l-1] and x[i][l-1] != x[j][l-1]:

r.append(x[i][:l-1]+sorted([x[j][l-1],x[i][l-1]]))

return r

#寻找关联规则的函数

def find\_rule(d, support, confidence, ms = u'--'):

result = pd.DataFrame(index=['support', 'confidence']) #定义输出结果

support\_series = 1.0\*d.sum()/len(d) #支持度序列

column = list(support\_series[support\_series > support].index) #初步根据支持度筛选

k = 0

while len(column) > 1:

k = k+1

print(u'\n正在进行第%s次搜索...' %k)

column = connect\_string(column, ms)

print(u'数目：%s...' %len(column))

sf = lambda i: d[i].prod(axis=1, numeric\_only = True) #新一批支持度的计算函数

#创建连接数据，这一步耗时、耗内存最严重。当数据集较大时，可以考虑并行运算优化。

d\_2 = pd.DataFrame(list(map(sf,column)), index = [ms.join(i) for i in column]).T

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

column = list(support\_series\_2[support\_series\_2 > support].index) #新一轮支持度筛选

support\_series = support\_series.append(support\_series\_2)

column2 = []

for i in column: #遍历可能的推理，如{A,B,C}究竟是A+B-->C还是B+C-->A还是C+A-->B？

i = i.split(ms)

for j in range(len(i)):

column2.append(i[:j]+i[j+1:]+i[j:j+1])

cofidence\_series = pd.Series(index=[ms.join(i) for i in column2]) #定义置信度序列

for i in column2: #计算置信度序列

cofidence\_series[ms.join(i)] = support\_series[ms.join(sorted(i))]/support\_series[ms.join(i[:len(i)-1])]

for i in cofidence\_series[cofidence\_series > confidence].index: #置信度筛选

result[i] = 0.0

result[i]['confidence'] = cofidence\_series[i]

result[i]['support'] = support\_series[ms.join(sorted(i.split(ms)))]

result = result.T.sort(['confidence','support'], ascending = False) #结果整理，输出

print(u'\n结果为：')

print(result)

return result

复制

结果为：

support confidence

e---a 0.3 1.000000

e---c 0.3 1.000000

c---a 0.5 0.714286

a---c 0.5 0.714286

a---b 0.5 0.714286

c---b 0.5 0.714286

b---a 0.5 0.625000

b---c 0.5 0.625000

复制

其中，e—a表示e发生能够推出a发生，置信度为100%,支持度为30% ； b—c—a表 示b、c同时发生时能够推出a发生，置信度为60%,支持度为30%等。搜索出来的关联规 则不一定具有实际意义，需要根据问题背景筛选适当的有意义的规则，并赋予合理的解释。

**2. Ariori算法:使用候选产生频繁项集** Apriori算法的主要思想是找出存在于事务数据集中的最大的频繁项集，在利用得到的最大频繁项集与预先设定的最小置信度阈值生成强关联规则。

**5.4、时序模式**

本文参与[腾讯云自媒体分享计划](https://cloud.tencent.com/developer/support-plan) ，欢迎热爱写作的你一起参与！

本文分享自作者个人站点/博客：https://blog.csdn.net/xc\_zhou/[复制](javascript:void(0);)

如有侵权，请联系 cloudcommunity@tencent.com 删除。

[展开阅读全文](javascript:;)

[编程算法](https://cloud.tencent.com/developer/tag/10663?entry=article)[聚类算法](https://cloud.tencent.com/developer/tag/10925?entry=article)[决策树](https://cloud.tencent.com/developer/tag/10721?entry=article)[深度学习](https://cloud.tencent.com/developer/tag/10333?entry=article)[AI 人工智能](https://cloud.tencent.com/developer/tag/10539?entry=article)

[举报](javascript:;)

点赞 1分享

相关文章

* 《python数据分析与挖掘实战》笔记第4章

数据预处理一方面是要提高数据的质量，另一方面是要让 数据更好地适应特定的挖掘技术或工具。统计发现，在数据挖掘的过程中，数据预处理工作量占到了整个过程的60%。

[周小董](https://cloud.tencent.com/developer/user/4908043)

* 《python数据分析与挖掘实战》笔记第2章

python有4个内建的数据结构–list（列表）、tuple（元组）、dictionary（字典）以及set（集合），它们可以统称为容器。

[周小董](https://cloud.tencent.com/developer/user/4908043)

* 《python数据分析与挖掘实战》笔记第3章

数据质量分析是数据挖掘中数据准备过程的重要一环，是数据预处理的前提，也是数据挖掘分析结论有效性和准确性的基础，没有可信的数据，数据挖掘构建的模型将是空中楼阁。

[周小董](https://cloud.tencent.com/developer/user/4908043)

* 《python数据分析与挖掘实战》笔记第1章

数据挖掘是从大量数据（包括文本）中挖掘出隐含的、先前未知的、对决策有潜在价值的关系、模式和趋势，并用这些知识和规则建立用于决策支持的模型，提供预测性决策支持的方...

[周小董](https://cloud.tencent.com/developer/user/4908043)

* Python数据分析与实战挖掘

基础篇 书推荐：《用python做科学计算》 ? 扩展库 简介 Numpy数组支持，以及相应的高效处理函数 Scipy矩阵支持，以及相应的矩阵数值计算模块 Ma...

[企鹅号小编](https://cloud.tencent.com/developer/user/1008345)

* Python数据分析与挖掘实战.pdf

[学一学大数据](https://cloud.tencent.com/developer/user/1685822)

* 关于《Python数据挖掘入门与实战》读书笔记二（亲和性分析）

首先确保最新的版本吧，python最让人头痛和无解的就是版本兼容性问题，不过既然选择了就不要怕麻烦，不断的纠错不断的尝试，在频繁的错误中提高自己。

[python与大数据分析](https://cloud.tencent.com/developer/user/7190337)

* 关于《Python数据挖掘入门与实战》读书笔记七（主成分分析二）

主成分分析算法（Principal Component Analysis，PCA）的目的是找到能用较少信息描述数据集的特征组合。它意在发现彼此之间没有相关性、能...

[python与大数据分析](https://cloud.tencent.com/developer/user/7190337)

* 关于《Python数据挖掘入门与实战》读书笔记六（主成分分析一）

特征抽取是数据挖掘任务最为重要的一个环节，一般而言，它对最终结果的影响要高过数据挖掘算法本身。但怎样选取好的特征，还没有严格、快捷的规则可循，这也是数据挖掘科学...

[python与大数据分析](https://cloud.tencent.com/developer/user/7190337)

* Python业务分析实战｜共享单车数据挖掘

自行车共享系统是传统自行车租赁的新一代，从注册会员、租赁到归还的整个过程都是自动化的。通过这些系统，用户可以很容易地从一个特定的位置租用自行车，并在另一个位置归...

[数据STUDIO](https://cloud.tencent.com/developer/user/8756457)

* Python数据分析与挖掘实战 (大数据技术丛书) - 张良均 等著

10余位数据挖掘领域资深专家和科研人员，10余年大数据挖掘咨询与实施经验结晶。从数据挖掘的应用出发，以电力、航空、医疗、互联网、生产制造以及公共服务等行业真实案...

[用户3157710](https://cloud.tencent.com/developer/user/3157710)

* Python 爬取淘宝商品数据挖掘分析实战

? 作者 孙方辉 本文为CDA志愿者投稿作品，转载需授权 项目内容 本案例选择>> 商品类目：沙发； 数量：共100页 4400个商品； 筛选条件：天猫...

[CDA数据分析师](https://cloud.tencent.com/developer/user/1283928)

* 关于《Python数据挖掘入门与实战》读书笔记一（数据挖掘、机器学习、深度学习、人工智能）

Python数据挖掘、Python机器学习、Python深度学习的书籍买了不少本了，但真正读下来的却很少，为何？涉及高等数学，图论、离散数学等等，对我来说和天文...

[python与大数据分析](https://cloud.tencent.com/developer/user/7190337)

* 慕课网数据分析与挖掘实战（一）-数据获取

其中，urllib，urllib2，requests，scrapy主要用于数据抓取 xpath主要用于分析提取有价值的内容 phantomjs，beauti...

[Meet相识](https://cloud.tencent.com/developer/user/3090887)

* 《数据挖掘与数据化运营实战 思路、方法、技巧与应用》—— 读书笔记

[用户1154259](https://cloud.tencent.com/developer/user/1154259)

* 关于《Python数据挖掘入门与实战》读书笔记三（估计器和近邻算法）

Scikit-learn库，实现了一系列数据挖掘算法，提供通用编程接口、标准化的测试和调参工具，便于用户尝试不同算法对其进行充分测试和查找优参数值。

[python与大数据分析](https://cloud.tencent.com/developer/user/7190337)