数据仓库与数据挖掘第二次作业实验报告

2018214157 金颖 2018214166 贾子洲

[一、实验环境 2](#_Toc532600598)

[二、任务一：银行直销数据分类 2](#_Toc532600599)

[2.1 数据预处理 2](#_Toc532600600)

[2.1.1 数据均衡化 2](#_Toc532600601)

[2.1.2 数据离散化 2](#_Toc532600602)

[2.1.3 数据正则化 3](#_Toc532600603)

[2 评价指标 3](#_Toc532600604)

[2.3 方法一：logistic回归分类器 4](#_Toc532600605)

[2.3.1 算法描述 4](#_Toc532600606)

[2.3.2 算法实现 4](#_Toc532600607)

[2.3.3 复杂度分析 5](#_Toc532600608)

[2.3.4 实验结果 5](#_Toc532600609)

[2.4 方法二：神经网络 6](#_Toc532600610)

[2.4.1 算法描述 6](#_Toc532600611)

[2.4.1 算法实现 6](#_Toc532600612)

[2.4.2 复杂度分析 7](#_Toc532600613)

[2.4.3 实验结果 7](#_Toc532600614)

[2.5 结果比较 8](#_Toc532600615)

[三、任务二：青蛙叫声聚类分析 8](#_Toc532600616)

[3.1距离度量与特征降维 8](#_Toc532600617)

[3.1.1 距离度量 8](#_Toc532600618)

[3.1.2 特征降维 8](#_Toc532600619)

[3.2 评价指标 9](#_Toc532600620)

[3.3 方法一：K-means 9](#_Toc532600621)

[3.3.1 算法描述 9](#_Toc532600622)

[3.3.2 算法实现 9](#_Toc532600623)

[3.3.3 复杂度分析 10](#_Toc532600624)

[3.4 方法二：层次聚类 10](#_Toc532600625)

[3.4.1 算法描述 10](#_Toc532600626)

[3.4.2 算法实现 10](#_Toc532600627)

[3.4.3 复杂度分析 10](#_Toc532600628)

[3.5 结果分析 10](#_Toc532600629)

[3.5.1 不同参数对应结果分析 10](#_Toc532600630)

[3.5.2 算法时间与聚类效果比较 12](#_Toc532600631)

[3.5.3 特征降维效果 13](#_Toc532600632)

[四、实验总结与心得体会 13](#_Toc532600633)

[4.1 实验总结 13](#_Toc532600634)

[4.1.2 分类算法实验总结 13](#_Toc532600635)

[4.1.2 聚类实验总结 13](#_Toc532600636)

[4.2 心得体会 14](#_Toc532600637)

# 一、实验环境

本次实验的实验环境如下：

编程语言：Python2.7.15

主要包及版本：Numpy1.15.2, torch0.4.1, torchvision0.2.1。

# 二、任务一：银行直销数据分类

本任务中，我们选择了全量数据集。

## 2.1 数据预处理

数据预处理主要包括数据均衡化、数据离散化和数据正则化，在特征选择方面，由于已经经过了离散化等处理，我们认为可以让算法去学习得到不同维数据对分类数据的影响大小，因此，在不做特殊说明的情况下，我们使用全量特征。

### 2.1.1 数据均衡化

我们统计了银行数据的标签情况，发现大部分的标签数据标记为no，少部分的标签数据标记为yes，正负样本的比例差距悬殊。在逻辑回归二分类问题中，差距过于悬殊的样本，容易导致分类器会倾向于向样本比例较大的一类预测，从而使得损失函数较小，正确率较高，然而分类器并没有学习到真正的正负类别的判别能力。

在这里，我们对数据进行了均衡化处理，对标记为yes的数据在原有已有数据的基础上，对标记为yes的数据进行重复采样，并将采样数据作为新的训练集样本加入到训练集中，或者对标记为no的数据进行随机采样，最终使得正负样本保持1:1的比例。

### 2.1.2 数据离散化

我们观察了银行数据样本的每一维的数据组成，其中既包含有类型为整数的维度（例如年龄等），还包含有类型为浮点数的维度（例如价格等），另外还包含了字符串类型的数据。 在这其中，字符串数据本身即为离散数据，然而逻辑回归分类器主要基于数值计算对数据样本进行分类，字符串数据无法参与到到逻辑回归分类器的计算，对于此类数据，我们将每个特定的字符串归为一类，则对于一个维度具有n种不同字符串的一列，我们用n个浮点数0来替代原有的字符串表达，并将每个数据特定的字符串所属类的位置的浮点数0替换为浮点数1，这样就实现了将字符串数据转换为可计算的对象。

另外我们考察了数字类型的维度，这些数据例如年龄，持续时间等数据，不能简单看做为连续性的程度上的表征，其数值并不一定可以表示为某种强烈程度的度量，将这些数据直接当做连续数据处理并不完全恰当。另外为了增强逻辑回归分类器的表达能力，我们将数值类型的数据也进行了离散化处理，我们将同一个维度的所有数值，从数值的最小值到数值的最大值，我们将这个区间分成了10份，每个落入特定区间的数字，则对应10个类别的某一类，类似于对字符串类型数据的处理，原有的数字类型被一个10维的0和1组成的数据替代，其中对应类别位置的值为1，非对应类别位置的值为0。这种方法在逻辑回归分类器中经常可以增强分类器的分类能力，但在本数据集中，实验表明该方法没有有效实现分类器性能的增强。

### 2.1.3 数据正则化

对于连续的数字数据，如果不进行离散化操作，这些数据大小差会非常大，这可能会影响到逻辑回归分类器的梯度下降。在这里，我们对这些不标准的数据进行了正则化处理，将所有的数据都正则化成了一个均值为0的，方差为1的高斯分布，提高分类效果。

## 2 评价指标

分类任务中常用的评价指标有以下几种：

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 预测 | |
| 1 | 0 |
| 真实 | 1 | a | b |
| 0 | c | d |

上表中展示了一个二分类任务的分类结果，其中1代表真样本，0代表假样本。

准确率 = 

精度 = 

召回率 = 

本实验中，针对二分类问题，我们在测试集上计算了模型的**准确率**，**精度和召回率**作为分类模型的评价指标。

另外，为了更好的数据可视化效果，我们利用了**ROC曲线**来对模型进行进一步的评价，ROC曲线以真正率为y轴，假正率为x轴，绘制了**ROC曲线**。

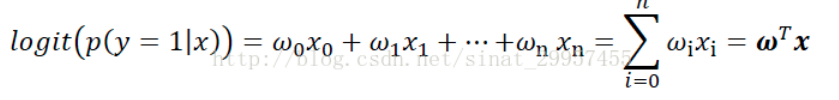
真正率 = 

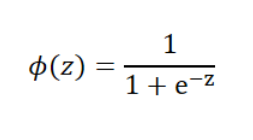
假正率 = 

## 2.3 方法一：logistic回归分类器

### 2.3.1 算法描述

逻辑回归分类器主要包含了两个单元，一个线性单元，由权重w和偏置b组成，输入x经过权重w加权后求和，然后加上偏置b，即可得到线性单元的输出。由于函数的曲线为线性，而在逻辑回归问题中，我们主要希望能够拟合0/1值，为了能够让输出尽可能接近0/1，我们在线性输出的基础上又增添了Sigmiod函数，使得输出保持在0到1之间，并且在整个函数空间范围内，除了0附近的位置趋向于0和-1之外，其他的位置均接近于0和1，这两部分的函数表示如下：

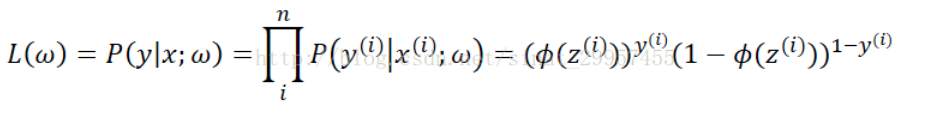




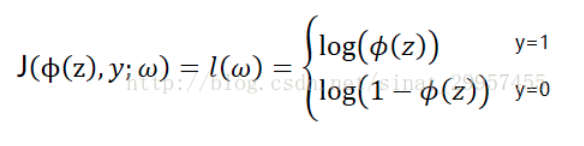
当逻辑回归函数输出的值接近于1时则认定，逻辑回归分类器将数据分类为正类，否则分类为负类。

### 2.3.2 算法实现

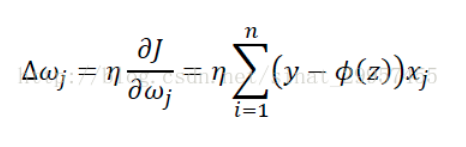
在训练上，逻辑回归分类器，将分类器的输出值看作为当前样本的概率，将参数当做为最大似然估计参数，使用了最大似然估计的方法，来构造了最大似然函数，则逻辑回归参数最有可能的值即为最大似然估计最大的值：



为了便于计算。将以上函数求其对数，并求其负值，即可得到逻辑回归分类的损失函数，即二项交叉熵损失：



由于该损失函数连续可导，则可以使用随机梯度下降的方法求取参数的最优值，梯度公式如下：



在实际的代码实现中，我们使用了一个和输入特征维度相同的数组代表了参数w，并用高斯分布初始化初始了该值。另外我们使用了一个浮点数b来代表方程的偏置，并初始化为0，在训练过程中，学习率选择为0.0001，同时由于数据量较小，在原有损失函数的基础上我们还给参数添加了L2损失以增强网络的泛化性能，其参数设置为1。

### 2.3.3 复杂度分析

该算法的时间复杂度需要分为训练实现复杂度和预测时间复杂度

假设每次训练的批次大小为bs，数据的维度为m，则每次前向运算需要的时间和m\*bs成正比，反向更新参数的时间消耗和m\*bs成正比。则每一个批次的训练的时间复杂度为m\*bs，假设数据需要训练t遍，每一遍的数据量为n，则每一遍共训练n/bs个批次，则每一遍的训练时间复杂度为 m\*n，则训练t遍的时间复杂度为：m\*n\*t

在预测时间复杂度上，假设预测的数据量为n，则前向传播的复杂度为 m\*n。

### 2.3.4 实验结果

在这里我们分别对正则化项为0, 0.1, 1.0 学习率为0.1, 0.01, 0.001分别进行了测试，并展示了不同评测标准下的评测结果，所有的结果均建立在阈值为0.5的基础上。

表1 逻辑回归分类实验结果（正则化项=0）

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy(%) | Precision(%) | Recall(%) |
| lr = 0.1 | 68.78 | 71.94 | 62.17 |
| lr = 0.01 | 68.09 | 76.82 | 53.74 |
| lr = 0.001 | 66.12 | 70.19 | 57.69 |

表2 逻辑回归分类实验结果（正则化项=0.1）

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy(%) | Precision(%) | Recall(%) |
| lr = 0.1 | 70.88 | 80.98 | 52.77 |
| lr = 0.01 | 68.27 | 74.91 | 56.65 |
| lr = 0.001 | 66.48 | 71.76 | 58.23 |

表3 逻辑回归分类实验结果（正则化项=1.0）

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy(%) | Precision(%) | Recall(%) |
| lr = 0.1 | 67.58 | 68.74 | 63.33 |
| lr = 0.01 | 67.86 | 71.97 | 61.09 |
| lr = 0.001 | 66.94 | 70.77 | 58.51 |

如上表所示，对于每一种参数，我们分别记录了三个指标，分别为正确率，精度和召回率。我们可以发现在正确率上学习率为0.1 正则化项为0.1 明显最优，在下面的对比实验中，则统一使用该组参数作为对比实验的固定参数。

接下来，我们对是否使用数据均衡化和数据离散化对模型精度的影响进行比较：

表4 数据均衡化与离散化对实验结果的影响

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 未离散 | | | 离散 | | | |
|  | Accuracy(%) | Precision(%) | Recall(%) | Accuracy(%) | Precision(%) | Recall(%) | |
| 未均衡 | 90.93 | 71.43 | 8.26 | 88.34 | 0.0 | 0.0 |
| 均衡 | 79.32 | 85.12 | 71.18 | 70.88 | 80.98 | 52.77 |

从上表中可以看出，未均衡化的数据正确率很高，但召回率非常低，这说明由于数据未均衡化，则数据明显倾向于预测为负类，大部分正例样本被遗漏。

对比均衡化数据这一行，则发现未离散数据相较于离散数据各项指标更好，则说明数据离散化在该问题中没有实际帮助。

最后，我们在数据未离散的基础上对数据进行了normalize处理，得到了数据最优异的结果，说明了normalize的重要作用，结果如下表

表5 正则化对实验结果的影响

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy(%) | Precision(%) | Recall(%) |
| normalize | 86.06 | 86.23 | 85.01 |

下面是我们绘制的ROC曲线：

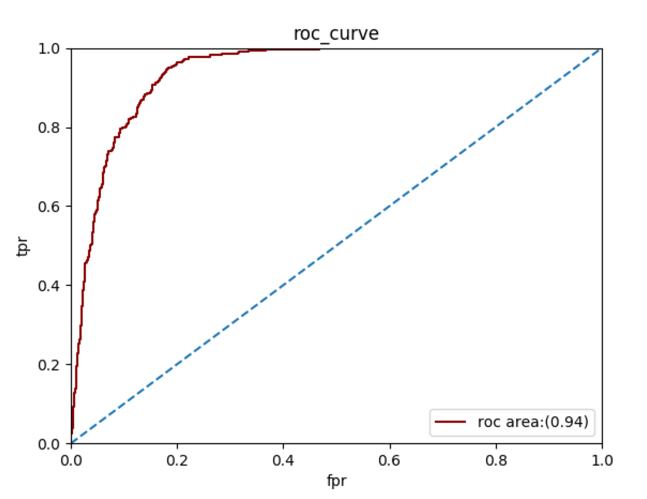


图1 logistic回归分类ROC曲线

## 2.4 方法二：神经网络

### 2.4.1 算法描述

神经网络的输入是每个样例的特征属性向量，供给到输入层，经过权值加权计算前馈到隐藏层，最终输出层输出网络预测结果，是一种非线性的模型。

它的训练过程概括如下：

1）随机初始化权值。

2）前向传播输入的训练数据。

3）反向传播预测错误信息。

4）直到收敛。

### 2.4.1 算法实现

神经网络在分类任务上有着广泛的应用，具有强鲁棒性，高预测精度，和强适应性，其比较明显的缺点有：参数众多，训练时间长，模型的可解释性差。

本实验中，我们实现了一个浅层的神经网络进行数据分类，其结构如下图。

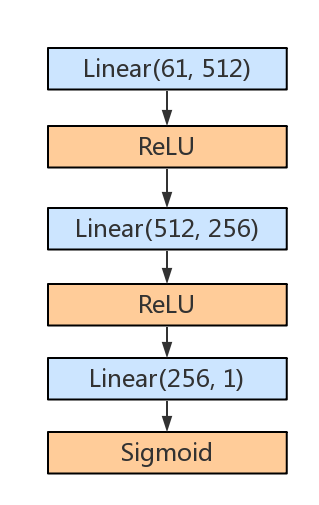


图2 神经网络结构

神经网络由三层线性层组成，前两层以ReLU作为激活函数，最后做sigmoid操作。

### 2.4.2 复杂度分析

神经网络涉及的参数量都十分庞大，比如在本实验的神经网络中，一个样本的前向传播计算需要数量级为T = 61\*512 + 512\*256 + 256 \* 1 次运算，那么对于n个样本，它的复杂度为 O(nT)，但由于T的值很大，神经网络的复杂性是较高的。另外，神经网络的耗时多少还与训练的epoch数量有关。

### 2.4.3 实验结果

对于样本不平衡的问题，这里我们采取随机采样的方法，使正负样例达到平衡，将筛选之后的数据集作为我们的实验数据集。在2.3的实验中已经证明了数据正则化对实验结果的提升作用，因此，我们的训练数据也进行了normalization操作。

由于神经网络本身具有特征学习的能力，作为对特征学习能力的一种体验，我们不再20种属性进行选取，仅仅进行正则化操作。我们将(n \* 61)维度的数据作为输入数据，其中随机抽取70%作为训练集，30%作为测试集。神经网络训练的batch\_size = 200, epoch = 10。

下面是神经网络的实验结果。

表6 神经网络预测结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy(%) | Precision(%) | Recall(%) |
| lr = 0.1 | 56.02 | 56.02 | 100 |
| lr = 0.03 | 87.47 | 85.94 | 93.13 |
| lr = 0.01 | 87.67 | 84.79 | 94.70 |
| lr = 0.003 | 85.9 | 87.01 | 87.82 |
| lr = 0.001 | 87.35 | 85.79 | 92.35 |
| lr = 0.0003 | 86.75 | 86.62 | 90.66 |
| lr = 0.0001 | 75.62 | 81.36 | 72.83 |

神经网络的平均训练时间在6s左右。从上表可以看出，在学习率为0.0003到0.03之间时，神经网络的预测表现都不错。

下面是绘制的ROC曲线：

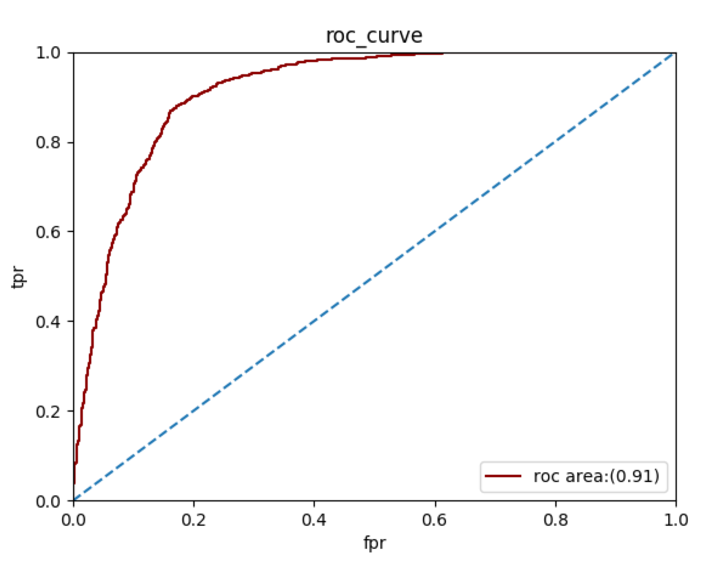


图3 神经网络ROC曲线

之后，又对每一层添加了batch normalization操作，进行了实验，发现平均训练时间在24s左右，但效果没有显著提升，因此不另外展示实验结果。

## 2.5 结果比较

基于以上实验数据，可以发现神经网络的效果还是要略好于传统的logistic回归分类，并且本实验中使用的神经网络结构还比较简单，如果采用更加复杂的结构，效果可能会更好，但参数量也会剧烈增长。

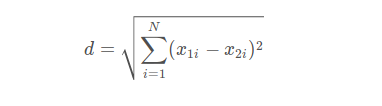
总体来说，神经网络的性能要优于logistic回归分类，而logistic回归分类的效率更高。

# 三、任务二：青蛙叫声聚类分析

## 3.1距离度量与特征降维

### 3.1.1 距离度量

在这个问题中，青蛙声音中的特征已经被有效的提取为MFCC特征类型，并分为了21一个维度，每个维度的值的分布情况相差不大，且都为浮点数类型。在这里我们选择了经典的欧式距离作为了青蛙叫声特征距离的衡量标准。其计算公式如下：



### 3.1.2 特征降维

在聚类问题中，我们实现了PCA特征降维方法，将原有的21组特征分别降维到了5、10、20等维度，并使用降维后的特征与原有的未降维的特征进行了比较。在PCA降维方法的具体实现上，我们计算每个维度特征的均值，然后计算了特征的协方差矩阵，并求得了矩阵的特征值和特征向量，然后以前k大的特征值对应的特征向量作为基底，对原有的特征向量进行了矩阵乘法，从而得到了新的k维的特征向量。

## 3.2 评价指标

我们共实现了三种评价指标：**正确率、purity、F-score**。

其中**正确率**不关注样本是否平衡等问题，仅考虑当前样本是否被分类正确；**purity**则将每个类中占比最大实际类的样本作为正确指标，求算占比最大类占整个类的百分比，并对所有的聚类百分比做平均值，主要考考虑每个类的主体类的占比情况 ；**F-score** 则对每个数据原有的类进行考量，分别计算了其命中率和召回率指标，并进行了调和平均，充分考量了原有标签每个类的分类情况，并尽量考虑了正负样本不平均等问题。另外需要说明的是在实际处理中，我们将每个聚类中，样本实际类别占比最多的类作为该聚类的预测类别，这使得预测成为可能。

## 3.3 方法一：K-means

### 3.3.1 算法描述

K-means是一种极其常用的聚类算法，它先随机产生K个聚类中心，然后进行如下迭代操作进行聚类。

1）计算每个点到聚类中心的距离

2）将每个点聚类到特定类中

3）更新聚类中心的值

### 3.3.2 算法实现

我们对K-means算法在全量数据上进行了实现，首先按照等间隔的原则在原有的数据集中选择k个点作为初始的聚类中心，然后给定聚类中心，开始迭代操作。

1）计算每个点到聚类中心的距离

在这一步操作中，我们遍历了数据中的每一个点，然后对每个中心分别通过欧式距离的度量方法来计算和每个中心的距离，并将距离值进行保存

2）将每个点聚类到特定类中

在这一步中，我们已知了每个点到k个聚类中心的距离，我们选取距离最小的那个聚类中心的编号作为该点的所属类，例如当前点距离第i个聚类中心点的距离较其他点最近，则将当前的点的类别标记为i-1，在实际运算中，该过程可以在计算聚类中心距离这一过程中一起进行操作

3）更新距类中心的值

在这一步中，我们将第二步中已经分配为同类的点的每一个维度分别求和并除以点的数量，得到一个平均中心点，并使用该平均中心点来更新旧的聚类中心

经过若干次迭代操作聚类逐渐变得稳定，得到了稳定的聚类结果。

### 3.3.3 复杂度分析

假设数据样本量为n，聚类的中心数为k，迭代次数为t，则可以知道每一轮迭代的时间复杂度为 n\*k，总共要进行t轮迭代，则总的时间复杂度为 n\*k\*t。

## 3.4 方法二：层次聚类

### 3.4.1 算法描述

这里，我们选择了自底向上的层次聚类，其主要步骤如下：

1）一开始，每一条数据记录为一个簇。

2）每次合并两个最相似的聚簇

3）直到簇的个数到达预设的某个值后，聚类停止。

### 3.4.2 算法实现

我们利用PCA将特征维度降到10，在全量数据集上实现了自底向上的层次聚类算法。这次，我们采用了最小法来度量两个簇之间的相似度。

1）初始化

将每条数据看做一个簇，计算两两之间的欧式距离，作为各个簇之间的相似性度量。

2）两两合并

对计算出的距离进行排序，找出其中距离最小，且尚未合并的两个簇进行合并。

3）聚类终止

当簇数目达到预设值时，停止聚类。

### 3.4.3 复杂度分析

层次聚类算法需要计算簇两两之间的相似度矩阵，复杂度为O(n3)。层次聚类算法的复杂度较高，为了尽可能优化性能，在实现代码时，采用了并查集等效率最优的解法，但耗时仍然较长，具体可见3.5。

## 3.5 结果分析

### 3.5.1 不同参数对应结果分析

#### 3.5.1.1 K-means

在这里以100次迭代作为上限，分别对k=5 k=10 k=20 k=40进行了实验，分别对正确率，purity度量和F-score度量进行了计算，从不同角度评估了k-means算法的表现效果，数据如下表，其中 F-score 指标占有4列，分别对4个类的F-score进行了分别计算。

表7 K-means聚类结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Purity | F-score c1 | F-score c2 | F-score c3 | F-score c4 |
| k=5 | 80.47 | 81.60 | 86.37 | 65.70 | 76.54 | 0.0 |
| k=10 | 83.67 | 83.77 | 90.95 | 0.0 | 80.41 | 0.0 |
| k=20 | 92.43 | 89.83 | 96.35 | 81.84 | 88.80 | 0.0 |
| k=40 | 95.44 | 94.35 | 98.02 | 88.38 | 92.90 | 65.92 |

接下来分别绘制了Purity和F-score（这里取c1为例）与k值变化的关系图如下。

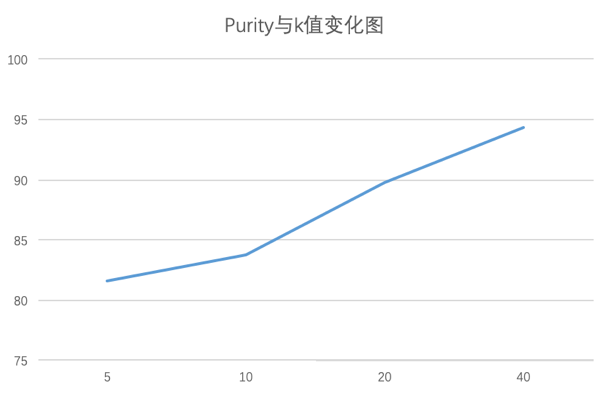


图4 purity与k值变化图

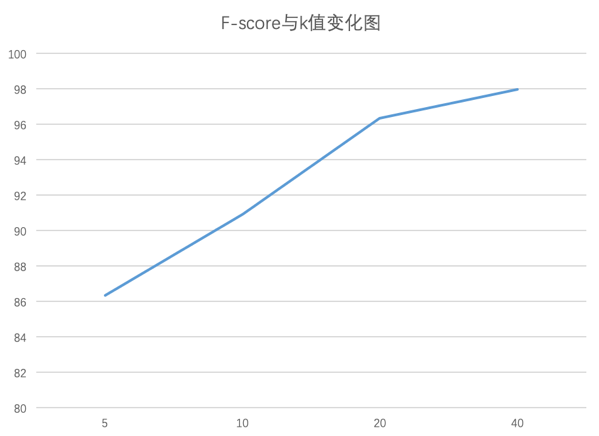


图5 F-score与k值变化图

从整体上来看，k的值越大则聚类效果越好，我们分析这既和指标计算方式有关，同时和聚类表达能力有关，在一定范围内k越大则聚类的表达能力越强。

另外一方面，我们观察四组F-score，我们发现不同的类的聚类难度并不相同，其中1,3类聚类难度较低，仅使用较小的k值即可实现良好的聚类效果，而第2,4类则当k值较小时，甚至无法形成代表性聚类。当k值较大时，才出现代表性的簇。

#### 3.5.1.2 层次聚类

分别对聚类阈值cluster=4、10、20、40、100、400、700、1000、2000进行了实验，分别对正确率，purity度量和F-score度量进行了计算，数据如下表，其中 F-score 指标占有4列，分别对4个类的F-score进行了分别计算。

表8 层次聚类结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Purity | F-score c1 | F-score c2 | F-score c3 | F-score c4 |
| cluster=4 | 61.45 | 90.36 | 76.12 | 0.0 | 0.1 | 0.0 |
| cluster=10 | 61.53 | 96.15 | 76.15 | 0.0 | 0.6 | 0.0 |
| cluster=20 | 61.63 | 98.08 | 76.2 | 0.0 | 1.19 | 2.90 |
| cluster=40 | 61.82 | 97.79 | 76.3 | 0.0 | 2.28 | 8.45 |
| cluster=100 | 62.96 | 99.12 | 76.84 | 0.0 | 9.33 | 11.11 |
| cluster=400 | 76.66 | 99.89 | 84.00 | 72.62 | 53.22 | 23.37 |
| cluster=700 | 97.67 | 99.91 | 98.76 | 100 | 96.10 | 25.64 |
| cluster=1000 | 98.96 | 99.91 | 99.7 | 100 | 98.28 | 27.85 |
| cluster=2000 | 99.86 | 99.98 | 99.88 | 100 | 99.77 | 100 |

接下来分别绘制了Purity和F-score（这里取c1为例）与cluster值变化（取100~2000之间的值）的关系图如下。

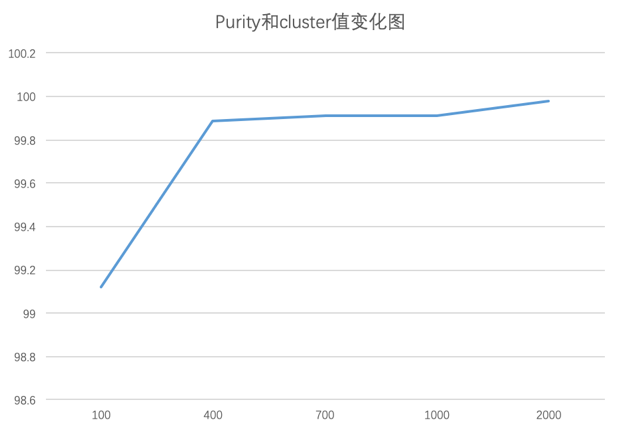


图6 purity与cluster值变化图

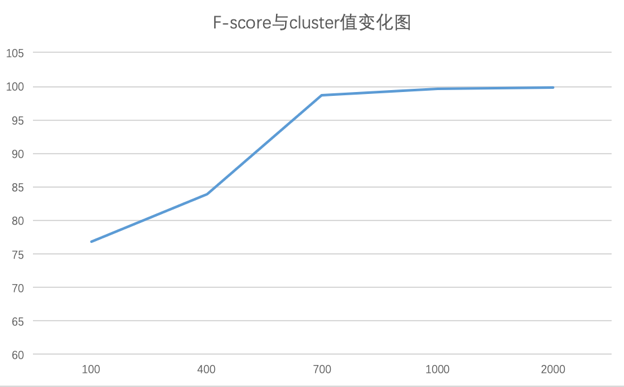


图7F-score与cluster值变化图

可以看出cluster的值越大则聚类效果越好。

另外观察四组F-score，我们发现cluster较小时，一些簇无法形成，这种情况相比K-means中出现的现象还要更加严重一些，一定程度上也体现了层次聚类的特点。

### 3.5.2 算法时间与聚类效果比较

实验耗时方面，K-means每轮迭代的耗时在几秒，但层次聚类的耗时在420秒左右，这一结果还是在代码已经经过性能优化的前提下达到的，体现出层次聚类的时间复杂度很高，与理论的时间复杂度相符。

在聚类效果方面，两种方法大致相当，但层次聚类更加不稳定，也有出现连接效应的风险，但在设置了合适的cluster的情况下，效果要好于K-means。而K-means的表现会更加稳定一些。

综上，在聚类效果差别不大的情况下，我们认为效率较高的K-means算法要优于层次聚类。

### 3.5.3 特征降维效果

另外，我们也对特征降维的效果进行了测试，在K-means实验中，我们以k=40 为基准，对PCA降维进行了测试，分别测试了降维到5,10,15和不降维的情况，并将结果统计如下：

表9 特征降维效果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | accuracy | purity | F-score c1 | F-score c2 | F-score c3 | F-score c4 |
| 5 | 94.83 | 92.93 | 96.67 | 89.33 | 92.60 | 78.62 |
| 10 | 95.37 | 94.57 | 97.99 | 88.27 | 93.13 | 64.80 |
| 15 | 95.47 | 94.49 | 98.09 | 88.57 | 93.10 | 66.29 |
| 无降维 | 95.44 | 94.35 | 98.02 | 88.38 | 92.90 | 65.92 |

我们发现，经过PCA降维和不经过PCA降维在accuracy和purity整体指标上变化不显著，但我们发现，在k=5时，第四类的值相较于无降维情况有明显的提升，增强了聚类的鲁棒性。这说明PCA降维可能对于难以聚类的类可能会有所帮助。

# 四、实验总结与心得体会

## 4.1 实验总结

本次实验主要分为两个部分：分类算法和聚类算法，下面对两部分分别进行总结。

### 4.1.2 分类算法实验总结

1）对全量银行数据集，我们进行了均衡化，离散化和正则化等预处理操作。

2）我们实现了logistic回归分类和神经网络分类两种算法。

3）我们计算了准确率，精度和召回率，绘制了ROC曲线来评价分类模型的性能。

4）分析实验结果，我们认为神经网络具有更强的拟合能力，而logistic回归分类的效率更高，可解释性更强。

### 4.1.2 聚类实验总结

1）对于青蛙叫声数据，我们采用欧氏距离作为距离度量，利用PCA进行了特征降维。

2）我们实现了K-means和层次聚类两种聚类算法。

3）我们计算了准确率accuracy，纯度purity 和F-score来评价聚类效果。给出了不同参数下每个算法的聚类效果比较。

4）通过比较两种算法，我们认为层次聚类的效率较低，而K-means是一种相对效率较高，性能也较好的聚类算法。

## 4.2 心得体会