数据挖掘课程大作业

**王起凡**

**24B918010**

**航天学院力学系**

**2024年12月22日**

一、基础部分。

根据题目要求，现在首先计算出树状图各个节点的支持度，然后对其根据支持度及与子集的关系进行分类。之后，为了方便可视化处理，利用Office Powerpoint软件在原图片上标记出详细的分类如下图1所示，然后导出为PDF。为了保证图片的分辨率，使用PS软件将PDF转为高清晰PNG格式文件，这也是数据挖掘课程第一节课讲的数据处理的应用。把标记为F的频繁项集的支持度也一并写上了。如下图所示：

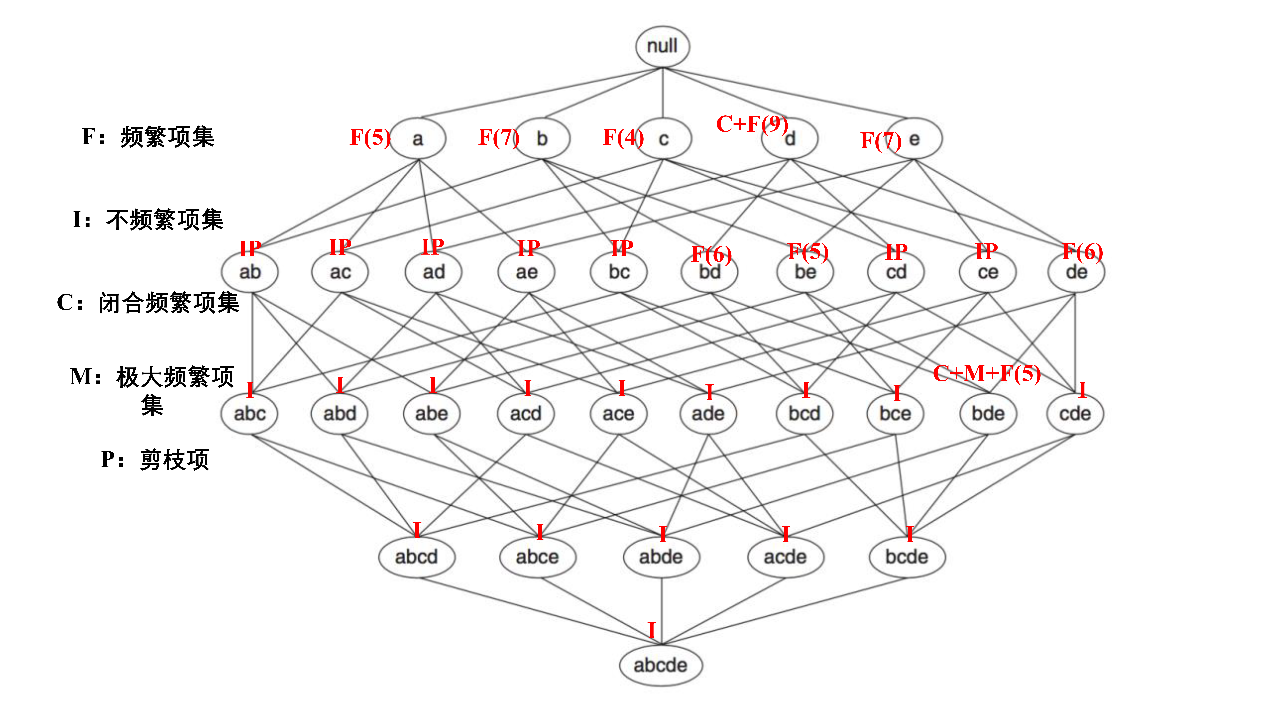


图 1节点标注示意图

为了更详细地反映各节点详细的支持度，把各个节点的支持度总结如下：

第1层（单项集）：

a: support=5 [F]

b: support=7 [F]

c: support=4 [F]

d: support=9 [F, C]

e: support=7 [F]

第2层（2项集）：

ab: support=2 [I, P]

ac: support=2 [I, P]

ad: support=4 [I, P]

ae: support=4 [I, P]

bc: support=2 [I, P]

bd: support=6 [F]

be: support=5 [F]

cd: support=4 [I, P]

ce: support=2 [I, P]

de: support=6 [F]

第3层（3项集）：

abc: support=1 [I]

abd: support=2 [I]

abe: support=2 [I]

acd: support=2 [I]

ace: support=1 [I]

ade: support=3 [I]

bcd: support=2 [I]

bce: support=1 [I]

bde: support=5 [F, M, C]

cde: support=2 [I]

第4层（4项集）：

abcd: support=1 [I]

abce: support=0 [I]

abde: support=2 [I]

acde: support=2 [I]

bcde: support=1 [I]

第5层（5项集）：

abcde: support=0 [I]

二、扩展部分

选择的论文是三篇文章中的第一篇Qiang Huang, Pingyi Luo, and Anthony K. H. Tung. 2023. A New Sparse Data Clustering Method based on Frequent Items. Proc. ACM Manag. Data 1, 1, Article 5 (May 2023), 28 pages. <https://doi.org/10.1145/3588685>。下面是整理的阅读报告：

这篇论文介绍了一种针对高维稀疏数据的创新聚类方法。从课程知识角度来看，该研究深入探讨了数据挖掘中的聚类分析技术，涵盖了包括基于划分的聚类算法、稀疏数据处理、Jaccard距离度量、局部敏感哈希(LSH)和频繁项集挖掘等重要知识点。论文特别关注了稀疏数据的特殊性质（高维度但大多数维度值为0），并围绕这一特点设计了相应的解决方案。

老师课堂上详细讲解了k-means算法的发展与详细原理，并且还介绍了一些改进的聚类算法，比如层次聚类算法与DBSCAN聚类算法。并且在第三次实验课上，老师利用make\_blob和make\_moon这两个数据集作为实验内容训练了我们对于k-means算法以及其改进算法的运用熟练度。这篇论文与前两个改进方法不同，在聚类中心表示与聚类算法方面均做出了创新。

在研究内容和创新点方面，论文主要提出了三个关键贡献：基于频繁项的聚类算法k-FreqItems、新型聚类中心表示方法FreqItem，以及基于LSH的种子选择方法SILK。这些创新不仅包括算法层面的改进（如使用Jaccard距离替代传统的欧氏距离），还涉及实现层面的优化（如分布式并行计算框架的设计）。这些创新共同构建了一个完整的稀疏数据聚类解决方案。下面分别来介绍老师课堂上讲解的k-means算法与文章提出的改进算法。

k-means作为经典的基于划分的聚类算法，其基本思想是通过迭代优化将数据划分为k个簇。算法首先随机选择k个初始中心点，然后重复执行两个步骤：将每个数据点分配给最近的中心(使用欧氏距离)，然后重新计算每个簇的中心点(计算均值)，直到中心点位置不再发生显著变化。然而，k-means在处理高维稀疏数据时存在局限性，主要体现在中心表示方式和距离度量的选择上。

针对这些问题，论文提出了k-FreqItems算法和SILK初始化技术。k-FreqItems保持了类似k-means的迭代框架，但引入了新的FreqItem中心表示方式和Jaccard距离度量，能更好地处理稀疏数据。具体来说，FreqItem通过选择频率超过特定阈值的维度作为中心的非零维度，有效保持了中心的稀疏性。而SILK则进一步优化了初始化过程，它利用两级MinHash技术将相似数据聚集到相同的桶中，然后从这些桶中识别频繁出现的数据作为初始中心点。这种方法避免了传统初始化方法中的顺序迭代问题，提供了更好的初始中心，从而加快了算法的收敛速度。

这项研究对数据挖掘技术的发展具有重要意义。首先，它提供了处理大规模高维稀疏数据的新方法，显著改善了传统k-means算法在稀疏数据上的表现。其次，通过提高聚类算法的可扩展性，为实际应用提供了可行的解决方案。最后，该研究在分布式数据挖掘技术的发展方面也做出了重要贡献。总的来说，这篇论文通过改进传统聚类算法，很好地解决了高维稀疏数据的聚类问题，展现了重要的理论价值和实际应用意义。

三、实践部分

使用 adaboost 对MAGIC gamma telescope数据集进⾏分类评估。为了方便审核，把以往的实验1、2、3的数据和实践部分的代码都开源到了github平台，地址为：

1、加载数据集，并对特征列进行编码添加列名，代码如下

import pandas as pd

import numpy as np

from sklearn import datasets

import urllib.request

# 下载MAGIC gamma telescope数据集

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"

data = pd.read\_csv(url, header=None)

# 添加有意义的列名

feature\_names = [

    'fLength', 'fWidth', 'fSize', 'fConc', 'fConc1',

    'fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist'

]

data.columns = feature\_names + ['class']

# 查看数据集的前几行

print("MAGIC Gamma Telescope数据集预览:")

print(data.head())

print("\n数据集基本信息:")

print(f"样本数量: {len(data)}")

print(f"特征数量: {len(feature\_names)}")

print(f"类别: {data['class'].unique()}")

print(f"\n每个类别的样本数量:\n{data['class'].value\_counts()}")

2、分割数据集，把原始数据预处理并且分为训练集和测试集。由于原始数据集比较复杂，因此首先利用预处理对特征列进行数字编码方便下文中分析。

from sklearn.model\_selection import train\_test\_split

import numpy as np

# 获取特征和标签

X = data[feature\_names]  # 选择特征列

y = (data['class'] == 'g').astype(int)  # 将'g'编码为1,'h'编码为0

# 修正后的类别平衡处理

gamma\_indices = np.where(y == 1)[0]

hadron\_indices = np.where(y == 0)[0]

# 确定较小的类别

min\_class\_size = min(len(gamma\_indices), len(hadron\_indices))

# 从两个类别中分别随机选择相同数量的样本

selected\_gamma\_indices = np.random.choice(gamma\_indices, min\_class\_size, replace=False)

selected\_hadron\_indices = np.random.choice(hadron\_indices, min\_class\_size, replace=False)

# 合并选择的样本索引

balanced\_indices = np.concatenate([selected\_gamma\_indices, selected\_hadron\_indices])

X\_balanced = X.iloc[balanced\_indices]

y\_balanced = y.iloc[balanced\_indices]

# 将数据集分割成训练集和测试集，70% 用于训练，30% 用于测试

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X\_balanced,

    y\_balanced,

    test\_size=0.3,  # 30%用于测试

    random\_state=42,  # 设置随机种子

    stratify=y\_balanced  # 确保划分后类别比例保持一致

)

print("\n平衡后的数据集大小:")

print(f"训练集大小: {X\_train.shape[0]}, 测试集大小: {X\_test.shape[0]}")

print(f"\n训练集类别分布:\n{pd.Series(y\_train).value\_counts()}")

print(f"\n测试集类别分布:\n{pd.Series(y\_test).value\_counts()}")

3、选择分类器，这里和老师作业里面使用一样的决策树分类器。利用scikit-learn分类器进行决策分类。

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

base\_clf = DecisionTreeClassifier(max\_depth=3)  # 限制深度以防止过拟合

# 创建AdaBoost分类器

clf = AdaBoostClassifier(

    base\_estimator=base\_clf,

    n\_estimators=100,  # 弱分类器的数量

    learning\_rate=1.0,  # 学习率

    random\_state=42  # 随机种子

)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

print("AdaBoost模型信息:")

print(f"基学习器数量: {clf.n\_estimators}")

print(f"学习率: {clf.learning\_rate}")

print(f"基学习器类型: DecisionTreeClassifier(max\_depth=3)")

4、模型评估方面，和老师文档里的示例代码一样，使用准确率和混淆矩阵来进行上面的决策树分类器的模型评估。

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# 设置中文字体 不然显示不出来

plt.rcParams['font.family'] = ['Microsoft YaHei']

plt.rcParams['axes.unicode\_minus'] = False  # 解决负号显示问题

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"准确率: {accuracy:.2f}")

cm = confusion\_matrix(y\_test, y\_pred)

print("\n混淆矩阵:")

print(cm)

# 可视化

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title('混淆矩阵可视化', fontsize=12)

plt.xlabel('预测类别', fontsize=10)

plt.ylabel('真实类别', fontsize=10)

labels = ['Hadron', 'Gamma']

plt.xticks([0.5, 1.5], labels)

plt.yticks([0.5, 1.5], labels)

plt.tick\_params(labelsize=10)

plt.tight\_layout()  # 自动调整布局

plt.show()

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred, target\_names=['Hadron', 'Gamma']))

5、超参数调整。利用GridSearchCV方法进行交叉验证。并且单纯文字表达不够直观反映超参数的影响，这里同样增加了线性图表来反映超参数的变化情况。

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

# 创建参数字典

param\_grid = {

    'base\_estimator\_\_max\_depth': [2, 3, 4, 5],  # 基学习器的深度

    'n\_estimators': [50, 100, 150],  # 弱分类器数量

    'learning\_rate': [0.1, 0.5, 1.0]  # 学习率

}

# 创建基础决策树分类器

base\_clf = DecisionTreeClassifier(random\_state=42)

# 创建AdaBoost分类器

ada\_clf = AdaBoostClassifier(

    base\_estimator=base\_clf,

    random\_state=42

)

# 创建GridSearchCV

grid\_search = GridSearchCV(

    ada\_clf,

    param\_grid,

    cv=5,

    scoring='accuracy',

    n\_jobs=-1

)

# 参数优化

grid\_search.fit(X\_train, y\_train)

# 输出最佳参数和得分

print("最佳超参数:", grid\_search.best\_params\_)

print(f"最佳交叉验证分数: {grid\_search.best\_score\_:.3f}")

# 最佳参数

best\_model = grid\_search.best\_estimator\_

y\_pred = best\_model.predict(X\_test)

# 评估最终模型

from sklearn.metrics import accuracy\_score, classification\_report

print("\n测试集准确率:", accuracy\_score(y\_test, y\_pred))

print("\n分类报告:")

print(classification\_report(y\_test, y\_pred))

# 可视化

import pandas as pd

import matplotlib.pyplot as plt

# 将cv\_results\_转换为DataFrame

results = pd.DataFrame(grid\_search.cv\_results\_)

plt.figure(figsize=(12, 6))

plt.plot(results['mean\_test\_score'], 'o-')

plt.title('不同参数组合的交叉验证性能', fontsize=12)

plt.xlabel('参数组合索引', fontsize=10)

plt.ylabel('准确率', fontsize=10)

plt.grid(True)

plt.show()

6、利用classification\_report来输出简单的分类报告。

from sklearn.metrics import classification\_report

# 输出分类报告

print(classification\_report(y\_test, y\_pred, target\_names=['Hadron', 'Gamma']))

7、混淆矩阵前面做过了，下面采用精确率、召回率与F1分数来作为评估指标分析模型性能。

from sklearn.metrics import precision\_score, recall\_score, f1\_score, confusion\_matrix

# 标签

y\_true = [0, 1, 1, 0, 1, 0, 1, 0, 1, 1]

y\_pred = [0, 1, 0, 0, 1, 0, 1, 1, 1, 0]

precision = precision\_score(y\_true, y\_pred)

recall = recall\_score(y\_true, y\_pred)

f1 = f1\_score(y\_true, y\_pred)

print(f"精确率: {precision:.2f}")

print(f"召回率: {recall:.2f}")

print(f"F1 分数: {f1:.2f}")