

大数据与信息工程学院

文本分类实战

**学 院：** 大数据与信息工程

**班 级：** 信管2101

**学 号：**  2100890225

**学生姓名：** 钟佳迅

**指导教师：** 杜逆索

**完成时间：** 2023 年 11 月

目录

[1. 数据集的介绍 1](#_Toc20840)

[2. 实验方法的说明 1](#_Toc23615)

[2.1. K近邻算法 1](#_Toc9416)

[2.1.1. 算法介绍 1](#_Toc19120)

[2.1.2. 算法应用 1](#_Toc27231)

[2.1.3. 基本思想 2](#_Toc26446)

[2.1.4. 算法的优缺点 2](#_Toc27641)

[2.2. 决策树算法 3](#_Toc32063)

[2.2.1. 算法介绍 3](#_Toc30348)

[2.2.2. 算法应用 3](#_Toc8827)

[2.2.3. 决策树构建 3](#_Toc14748)

[2.2.4. 算法的优缺点 5](#_Toc15390)

[2.3. 逻辑回归算法 6](#_Toc21209)

[2.3.1. 算法介绍 6](#_Toc4626)

[2.3.2. 算法应用 6](#_Toc28568)

[2.3.3. 基本原理 6](#_Toc21798)

[2.3.4. 算法的优缺点 7](#_Toc31500)

[2.4. 词袋法 7](#_Toc21205)

[2.4.1. 介绍 7](#_Toc18296)

[2.4.2. 步骤 8](#_Toc11335)

[3. 实验目标及实验流程 8](#_Toc20903)

[3.1. 目标需求 8](#_Toc21071)

[3.2. 流程图及说明 9](#_Toc31086)

[4. 实验结果的说明与展示 10](#_Toc22138)

[4.1. 多次实验结果数据及解决方案 10](#_Toc7948)

[4.1.1. 结果展示（数据示例） 10](#_Toc24933)

[4.1.2. 多次实验取平均值 11](#_Toc21656)

[4.2. 各个模型的混淆矩阵 12](#_Toc24822)

[4.3. 算法的优劣比较 14](#_Toc3045)

[4.3.1. 总体来说 14](#_Toc21806)

[4.3.2. 细分来说 15](#_Toc11395)

[5. 参考文献 18](#_Toc6311)

[附录1:代码 19](#_Toc6851)

[附录2:多次实验结果数据 26](#_Toc10757)

# 数据集的介绍

表1：数据集概况

|  |  |  |
| --- | --- | --- |
|  | 样本数 | 标签 |
| 训练集 | 3305 | [女性，体育，文学，校园] |
| 测试集 | 200 | [女性，体育，文学，校园] |

# 实验方法的说明

## K近邻算法

### 算法介绍

近邻算法是数据挖掘分类算法中比较常用的一种方法。K-近邻是基于统计 的分类方法,是根据测试样本在特征空间中K 个最近邻样本中的多数样本 的类别来进行分类，因此具有直观、无需先验统计知识等特点，从而成为非参数分类的一种重要方法。

### 算法应用

k近邻算法广泛应用于模式识别、图像处理、数据挖掘等领域。例如，在图像处理中，可以使用k近邻算法对图像进行分类；在数据挖掘中，可以使用k近邻算法对数据进行分类和聚类。

### 基本思想

产生训练集，使得训练集按照已有的分类标准划分成离散型数值类，或者是连续型数值类输出。以训练集的分类为基础，对测试集每个样本寻找K个近邻，采用欧氏距离作为样本间的相似程度的判断依据，相似度大的即为最近邻。一般近邻可以选择1个或者多个。当类为连续型数值时，测试样本的最终输出为近邻的平均值；当类为离散值时，测试样本的最终输出为近邻类中个数最多的那一类。所有的例子都处于N维空间，一般每个例子x都被表示为特征向量<a₁(x),a₂(x)…,a i(x)>,这里a,(x)表示例子x的第i个属性值。那么两个例子x,x,之间的相似度量可以采用欧氏距离：

*（2.1）*

判断近邻就是使用欧氏距离测试两个例子之间的距离，距离值越小的表明相似性越大，反之则表明相似性越小。

### 算法的优缺点

**优点：**k近邻算法简单易懂，易于实现，对异常值不敏感，适用于多分类问题。

**缺点：**k近邻算法需要大量的存储空间，计算复杂度高，对于高维数据和样本不平衡的数据集表现不佳。

## 决策树算法

### 算法介绍

决策树是一种基本的分类与回归方法，是一种树形结构，其中每个内部节点表示一个属性上的判断，每个分支代表一个判断结果的输出，最后每个叶节点代表一种分类结果或者回归预测结果。决策树算法通过对数据进行分析，构建决策树模型，从而实现对数据的分类和预测。

### 算法应用

决策树算法广泛应用于数据挖掘、机器学习、模式识别等领域，如金融风险评估、医学诊断、客户流失预测等。

### 决策树构建

**特征选择：**选取有较强分类能力的特征。

**决策树生成：**典型的算法有 ID3 和 C4.5， 它们生成决策树过程相似， ID3 是采用信息增益作为特征选择度量， 而 C4.5 采用信息增益比率。

**决策树剪枝：**剪枝原因是决策树生成算法生成的树对训练数据的预测很准确， 但是对于未知数据分类很差， 这就产生了过拟合的现象。常见的后剪枝算法有REP,PEP,MEP以及CCP。

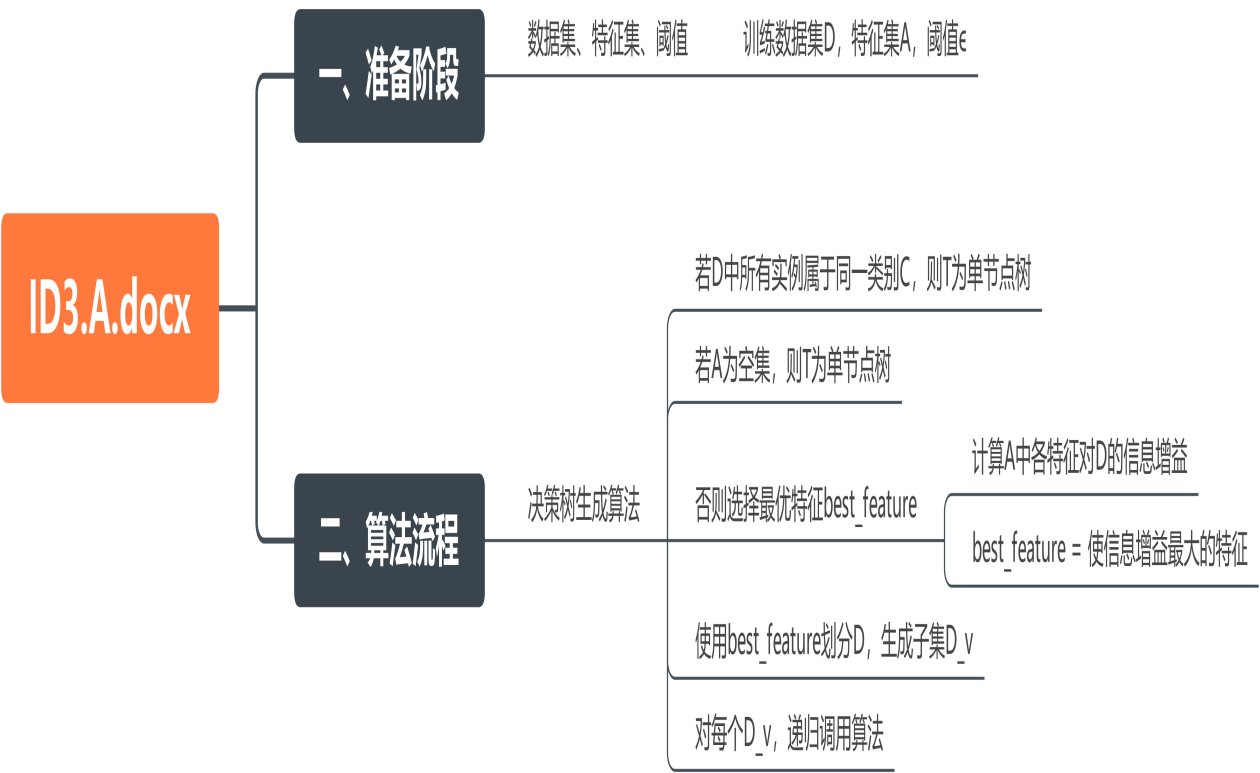


图1：ID3算法基本流程

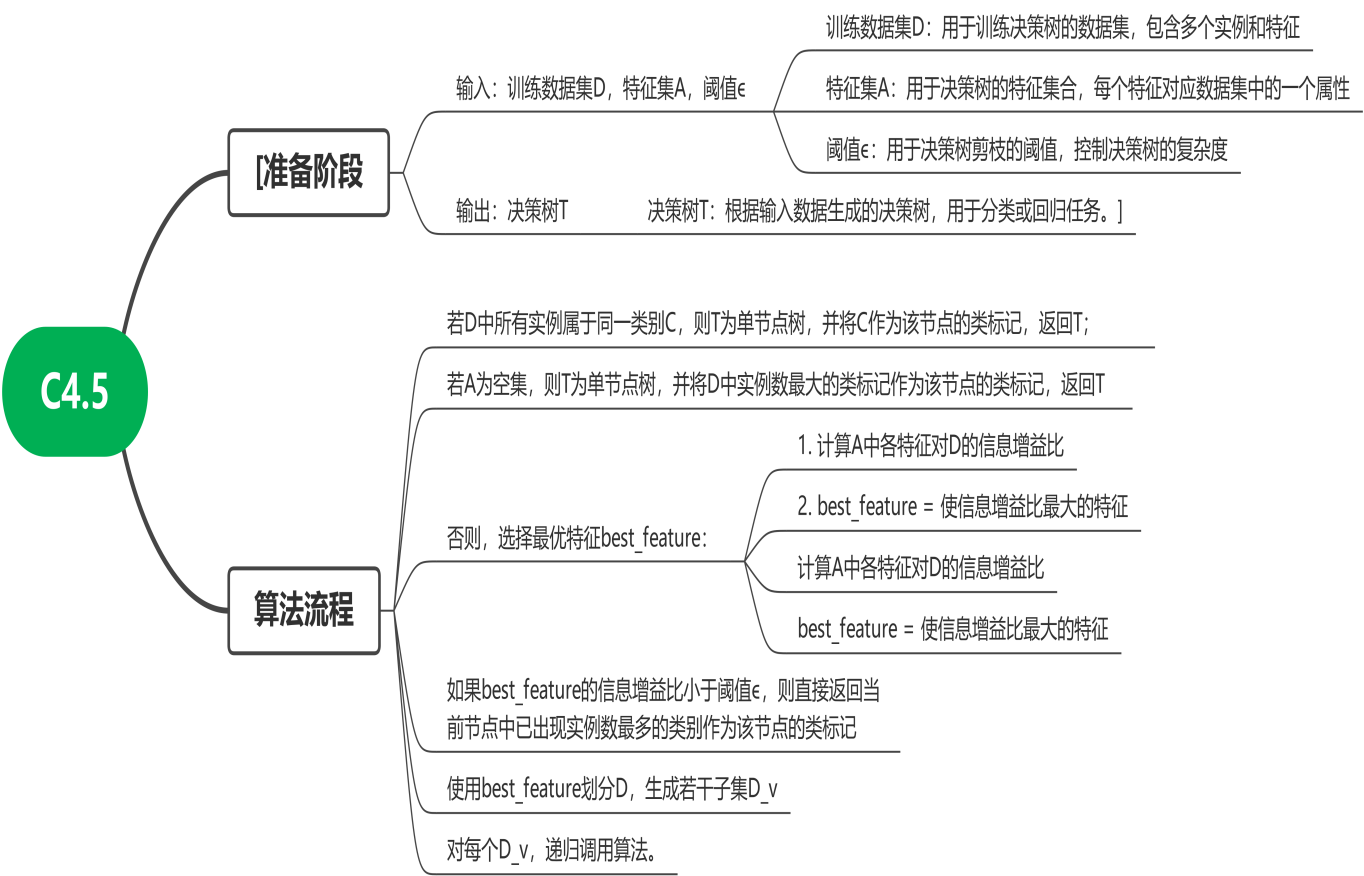


图2：C4.5基本流程

表2：各类后剪枝算法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **算法名称** | **剪枝方式** | **是否需要独立剪枝集** | **误差估计** | **计算复杂度** |
| REP | 自底向上 | 需要 | 剪枝集上的误差估计 | O(n) |
| PEP | 自顶向下 | 不需要 | 使用连续校正 | O(n) |
| MEP | 自底向上 | 不需要 | 基于m-概率估计 | O(n) |
| CCP | 自底向上 | 不需要 | CV方式或标准误差 | O(n²) |

### 算法的优缺点

**优点：**速度快:计算量相对较小，且容易转化成分类规则。只要沿着树根向下一直走到叶，沿途的分裂条件就能够唯一确定一条分类的谓词。准确性高:挖掘出的分类规则准确性高，便于理解，决策树可以清晰的显示哪些字段比较重要。非参数学习，不需要设置参数。

**缺点：**决策树很容易过拟合，很多时候即使进行后剪枝也无法避免过拟合的问题，因此可以通过设置树深或者叶节点中的样本个数来进行预剪枝控制；决策树属于样本敏感型，即使样本发生一点点改动，也会导致整个树结构的变化，可以通过集成算法来解决；

## 逻辑回归算法

### 算法介绍

逻辑回归是一种广泛应用于机器学习和数据分析的算法。它的主要思想是将线性回归的输出通过一个sigmoid函数映射到0到1之间，表示事件发生的概率。逻辑回归可以用于二分类问题，也可以通过一些技巧扩展到多分类问题。

### 算法应用

逻辑回归可以应用于很多领域，例如金融、医疗、电商等。在金融领域，逻辑回归可以用于信用评估、欺诈检测等；在医疗领域，逻辑回归可以用于疾病诊断、药物疗效预测等；在电商领域，逻辑回归可以用于用户行为分析、广告推荐等。

### 基本原理

**基本假设：**假设输入特征与输出标签之间存在线性关系。输出标签服从二项分布（二分类问题）。

**线性模型：**对于二分类问题，假设线性模型为：

（2.2）

其中z表示模型的输出，为模型的参数，为输入特征。

**非线性映射：**为了将线性模型的输出转化为概率值，在逻辑回归中使用了sigmoid函数（也称为逻辑函数）进行非线性映射。sigmoid函数定义如下：

（2.3）

**概率计算：**将线性模型的输出通过sigmoid函数映射后，得到该样本属于正例的概率值概率值范围在0到1之间。另一类的概率值可以通过计算得到。

**决策边界：**根据预测的概率值，可以根据设定的阈值进行分类决策。通常，当，认为样本属于正例；当，认为样本属于另一类。

**损失函数与参数估计：**逻辑回归使用的损失函数是对数损失函数（也称为交叉熵损失函数）。参数估计采用最大似然估计（MLE）的方法，通过最大化似然函数来求解模型的参数。

### 算法的优缺点

**优点：**计算简单，易于实现；可以处理高维数据；可以得到事件发生的概率，而不仅仅是分类结果。

**缺点：**逻辑回归假设特征之间是线性关系，无法处理非线性关系；对异常值比较敏感；逻辑回归的表现受特征选择的影响较大。

## 词袋法

### 介绍

CountVectorizer是[自然语言处理](https://so.csdn.net/so/search?q=%E8%87%AA%E7%84%B6%E8%AF%AD%E8%A8%80%E5%A4%84%E7%90%86&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/AI_dataloads/article/details/_blank)中常用的文本特征提取方法之一，用于将文本数据转换为词袋（Bag of Words，BoW）表示。它将文本文档转换为词汇表中的单词计数向量。

### 步骤

**导入库：**from sklearn.feature\_extraction.text import CountVectorizer

**创建CountVectorizer对象：**vectorizer = CountVectorizer()

**拟合和转换数据：**X = vectorizer.fit\_transform(text\_data)

**获取词汇表：**vocabulary = vectorizer.get\_feature\_names\_out()

**获得词频矩阵：**word\_counts = X.toarray()

**使用文本向量：**将这些词频矩阵作为特征输入到机器学习模型中，用于文本分类、聚类等任务。

# 实验目标及实验流程

## 目标需求

1. 批量读取文档文件.txt；
2. 文本向量化使用词袋法或者TFIDF；
3. 比较各个算法的性能优劣，包括模型消耗用时、模型准确率。

## 流程图及说明

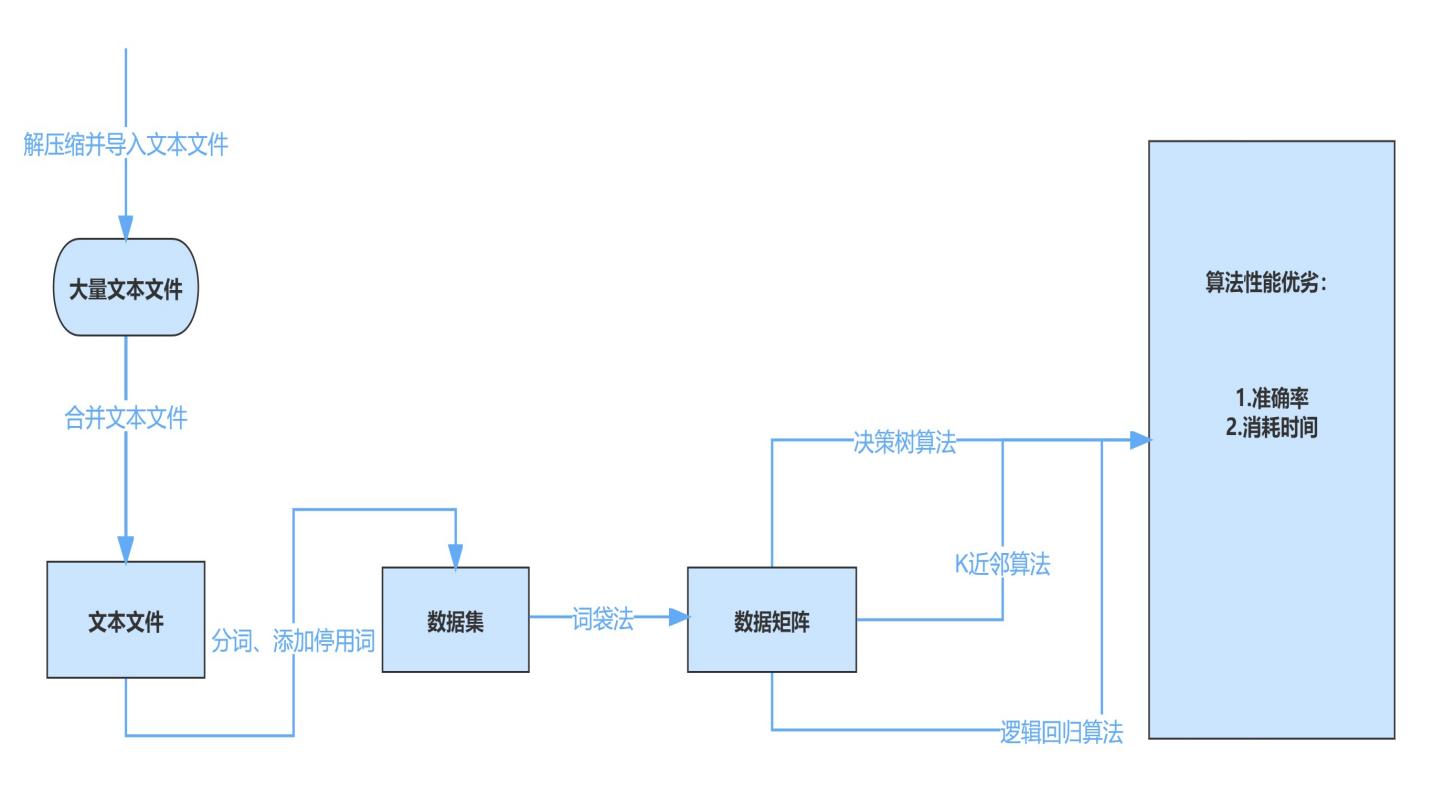


图3：流程图

如图3所示，简单来说，先读取解压缩后的test\_classifaction文件，并对该文件夹中的诸多文件进行合并，然后进行分词和去除停用词得到新的数据集，用词袋法将文本向量化得到数据矩阵，再分别使用决策树算法，K近邻算法，逻辑回归算法进行训练，并通过准确率以及消耗时间这两项指标比较算法的性能优劣。

**具体如下：**

**数据准备：**从指定路径中读取训练集和测试集数据；训练集包含四个分类标签：女性、体育、校园、文学；测试集与训练集相同

**数据预处理：**使用jieba库对文本进行分词；加载停用词语料，并去除停用词；

使用CountVectorizer对文本进行特征提取，转换为词频矩阵

**实验算法选择：**选择K近邻分类器(KNeighborsClassifier)、决策树分类器(DecisionTreeClassifier)、逻辑回归分类器(LogisticRegression)

**实验过程：**

对每个算法进行以下步骤：

（1）算法训练：使用训练集进行模型训练；

（2）算法预测：使用测试集进行模型预测；

（3）性能评估：计算准确率、混淆矩阵和召回率。

# 实验结果的说明与展示

## 多次实验结果数据及解决方案

### 结果展示（数据示例）

表3：第一次实验

|  |  |  |
| --- | --- | --- |
| **分类器** | **准确率** | **消耗时间/s** |
| KNeighborsClassifier | 0.665 | 51.84 |
| DecisionTreeClassifier | 0.770 | 21.97 |
| LogisticRegression | 0.890 | 8.24 |

表4：第二次实验

|  |  |  |
| --- | --- | --- |
| **分类器** | **准确率** | **消耗时间/s** |
| KNeighborsClassifier | 0.665 | 45.00 |
| DecisionTreeClassifier | 0.770 | 21.58 |
| LogisticRegression | 0.890 | 8.41 |

表5：第三次实验

|  |  |  |
| --- | --- | --- |
| **分类器** | **准确率** | **消耗时间/s** |
| KNeighborsClassifier | 0.665 | 45.87 |
| DecisionTreeClassifier | 0.745 | 18.74 |
| LogisticRegression | 0.890 | 7.49 |

表6：第N次实验

|  |  |  |
| --- | --- | --- |
| **分类器** | **准确率** | **消耗时间/s** |
| KNeighborsClassifier | 0.665 | 41.41 |
| DecisionTreeClassifier | 0.735 | 19.83 |
| LogisticRegression | 0.890 | 7.07 |

### 多次实验取平均值

通过50次实验可以看出，KNeighborsClassifier和LogisticRegression的准确率在多次运行中保持不变，分别为0.665,0.890，而DecisionTreeClassifier的准确率在0.715——0.775之间不断变化。三者的消耗时间一直处于变化状态。

这可能是由于决策树算法本身的特性所导致的。决策树在构建过程中会根据特征的不同选择最佳分割点，因此每次运行时得到的决策树结构可能会有所不同，进而影响模型的准确率。这也意味着决策树算法对数据的细微变化比较敏感。故此我们对DecisionTreeClassifier分类器的准确率求平均：

（4.1）

DecisionTreeClassifier分类器的准确率求平均得= 0.747

三个算法模型消耗时间取平均值依次为= 41.60(s),= 7.93(s)

= 20.21(s)。

## 各个模型的混淆矩阵

选取与准确率平均值最相近的一次实验的混淆矩阵为例：

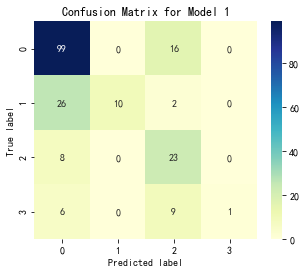
.

图4：KNN模型的混淆矩阵热力图

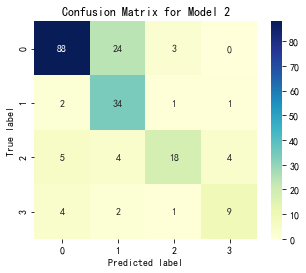


图5：决策树模型的混淆矩阵热力图

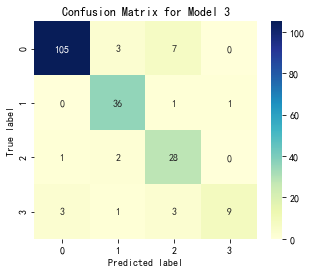


图6：决策树模型的混淆矩阵热力图

由混淆矩阵我们可以计算总体平均准确率，个体的准确率，精确率，召回率以及F1值等，例如根据图4可得KNN算法模型的性能：

，

0.71，

同理可得，0.46，，

,

同理可得，0.74，，

,

同理可得，0.57，。

## 算法的优劣比较

### 总体来说

根据混淆矩阵计算出的每一个平均准确率绘制直方图如下：

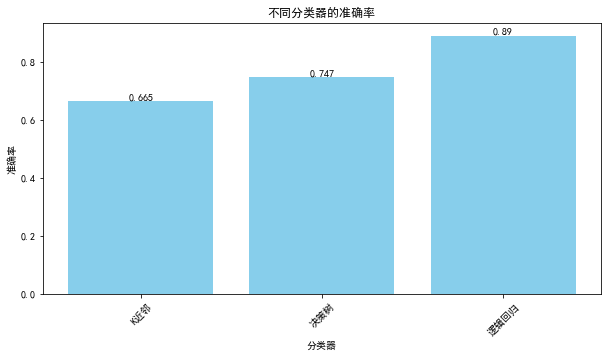


图7：各个算法的准确率

根据平均消耗时间绘制直方图如下：

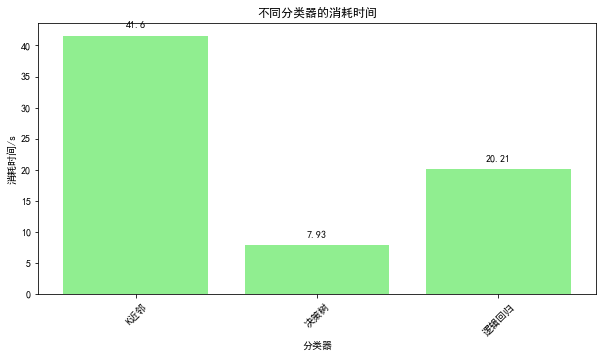


图8：各个算法训练数据集消耗的时间

由图7,8可看出，logisticRegression的准确率高，且训练数据耗费时间短，故训练此数据从准确率和消耗时间这两个因素来判断算法优劣，比较得出logisticRegression优于DecisionTreeClassifier,DecisionTreeClassifier优于KNeighborsClassifier。

### 细分来说

根据混淆矩阵计算出的P、R、F1值绘制如下：

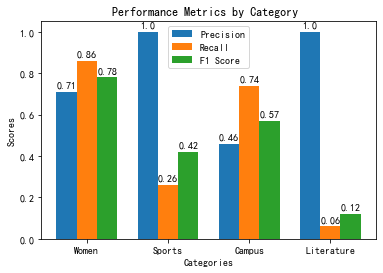


图9：KNN

由图可得，在使用KNN算法分类器时，"Sports" 类别的精度较高，但召回率相对较低，可能有一部分样本被错误分类或漏掉。"Campus" 类别的召回率较高，但精度较低，可能有一部分非该类别的样本被错误地划分为该类别。而"Women" 类别在精度、召回率和F1分数上都表现较好，而"Literature" 类别在召回率上较低，可能有大量该类别的样本未被正确找回。

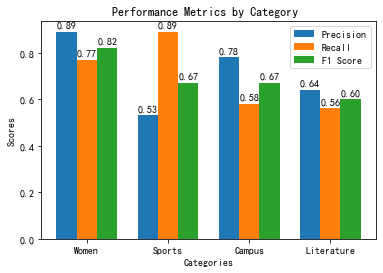


图10:决策树

由图可知，在使用决策树分类器时，"Women" 类别在精度、召回率和F1分数上表现较好，而"Sports" 类别的召回率相对较高，但精度较低。"Campus" 类别在精度和F1分数上表现较好，但召回率相对较低。而"Literature" 类别在三个指标上都相对较低。

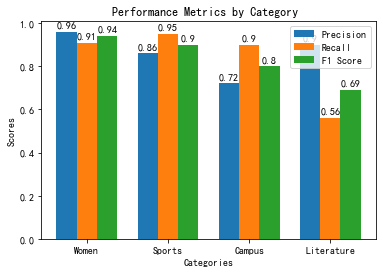


图11：逻辑回归

由图可知，在使用逻辑回归分类器针对不同的类别，我们需要采用不同的策略来提高分类模型的性能。对于性能最好的类别 "Women"，可以尝试进一步提升模型的精度和召回率，以达到更高的F1分数；对于性能较差的类别 "Literature"，则需要考虑增加数据量、改进特征工程或尝试其他算法等方式来提高性能。

# 参考文献

1. 湛燕. K-近邻、K-均值及其在文本分类中的应用[D].河北大学,2004.
2. 冯少荣.决策树算法的研究与改进[J].厦门大学学报(自然科学版),2007(04):496-500.
3. 魏红宁.决策树剪枝方法的比较[J].西南交通大学学报,2005(01):44-48.
4. 王黎明. 决策树学习及其剪枝算法研究[D].武汉理工大学,2007.

# 附录1:代码

1. **import** os
2. **import** jieba
3. **import** time
4. **import** warnings
5. **import** numpy as np
6. **import** pandas as pd
7. **import** matplotlib.pyplot as plt
8. **from** matplotlib.font\_manager **import** FontProperties
9. **from** sklearn **import** metrics
10. **from** sklearn.tree **import** DecisionTreeClassifier
11. **from** sklearn.neighbors **import** KNeighborsClassifier
12. **from** sklearn.linear\_model **import** LogisticRegression
13. **from** sklearn.ensemble **import** AdaBoostClassifier
14. **from** sklearn.preprocessing **import** LabelEncoder
15. **from** sklearn.feature\_extraction.text **import** CountVectorizer
16. **from** sklearn.feature\_extraction.text **import** TfidfVectorizer
18. warnings.filterwarnings('ignore')
20. path =  r'C:\Users\LENOVO\text\_classification'
22. # 设置字体
23. plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置字体为黑体或者其他支持中文的字体
24. plt.rcParams['axes.unicode\_minus'] = False  # 用来正常显示负号

27. **def** read\_text(path, text\_list):
28. '''''
29. path: 必选参数，文件夹路径
30. text\_list: 必选参数，文件夹 path 下的所有 .txt 文件名列表
31. return: 返回值
32. features 文本(特征)数据，以列表形式返回;
33. labels 分类标签，以列表形式返回
34. '''
36. features, labels = [], []
37. **for** text **in** text\_list:
38. **if** text.split('.')[-1] == 'txt':
39. **try**:
40. with open(path + text, encoding='gbk') as fp:
41. features.append(fp.read())          # 特征
42. labels.append(path.split('/')[-2])  # 标签
43. **except** Exception as erro:
44. **print**('\n>>>发现错误, 正在输出错误信息...\n', erro)
46. **return** features, labels
48. **def** merge\_text(train\_or\_test, label\_name):
49. '''''
50. train\_or\_test: 必选参数，train 训练数据集 or test 测试数据集
51. label\_name: 必选参数，分类标签的名字
52. return: 返回值
53. merge\_features 合并好的所有特征数据，以列表形式返回;
54. merge\_labels   合并好的所有分类标签数据，以列表形式返回
55. '''
57. **print**('\n>>>文本读取和合并程序已经启动, 请稍候...')
59. merge\_features, merge\_labels = [], []  # 函数全局变量
60. **for** name **in** label\_name:
61. path = r'C:\Users\LENOVO\text\_classification/'+ train\_or\_test +'/'+ name +'/'
62. text\_list = os.listdir(path)
63. features, labels = read\_text(path=path, text\_list=text\_list)  # 调用函数
64. merge\_features += features  # 特征
65. merge\_labels   += labels    # 标签
67. # 可以自定义添加一些想要知道的信息
68. **print**('\n>>>你正在处理的数据类型是...\n', train\_or\_test)
69. **print**('\n>>>[', train\_or\_test ,']数据具体情况如下...')
70. **print**('样本数量\t', len(merge\_features), '\t类别名称\t', set(merge\_labels))
71. **print**('\n>>>文本读取和合并工作已经处理完毕...\n')
73. **return** merge\_features, merge\_labels
75. # 获取训练集
77. train\_or\_test = 'train'
79. label\_name = ['女性', '体育', '校园', '文学']
80. X\_train, y\_train = merge\_text(train\_or\_test, label\_name)
82. # 获取测试集
84. train\_or\_test = 'test'
86. label\_name = ['女性', '体育', '校园', '文学']
87. X\_test, y\_test = merge\_text(train\_or\_test, label\_name)
89. # 训练集
91. X\_train\_word = [jieba.cut(words) **for** words **in** X\_train]
92. X\_train\_cut = [' '.join(word) **for** word **in** X\_train\_word]
94. # 测试集
96. X\_test\_word = [jieba.cut(words) **for** words **in** X\_test]
97. X\_test\_cut  = [' '.join(word) **for** word **in** X\_test\_word]
99. # 加载停止词语料
101. stoplist = [word.strip() **for** word **in** open(r'C:\Users\LENOVO\text\_classification\stop\stopword.txt',encoding='utf-8').readlines()]
103. le = LabelEncoder()
105. y\_train\_le = le.fit\_transform(y\_train)
106. y\_test\_le  = le.fit\_transform(y\_test)
108. count = CountVectorizer(stop\_words=stoplist)
109. count.fit(list(X\_train\_cut) + list(X\_test\_cut))
110. X\_train\_count = count.transform(X\_train\_cut)
111. X\_test\_count  = count.transform(X\_test\_cut)
113. X\_train\_count = X\_train\_count.toarray()
114. X\_test\_count  = X\_test\_count.toarray()
116. **print**(X\_train\_count.shape, X\_test\_count.shape)
118. # 用于存储所有算法的名字，准确率和所消耗的时间
119. estimator\_list, score\_list, time\_list = [], [], []
121. **def** get\_text\_classification(estimator, X\_train, y\_train, X\_test, y\_test):
122. '''''
123. estimator: 必选参数，分类器模型
124. X\_train: 必选参数，训练集特征数据
125. y\_train: 必选参数，训练集标签数据
126. X\_test: 必选参数，测试集特征数据
127. y\_test: 必选参数，测试集标签数据
128. return: 返回值
129. clf\_name 分类器名称
130. clf 分类器模型
131. accuracy 准确率
132. time\_consume 消耗时间
133. '''
135. clf\_name = estimator.\_\_class\_\_.\_\_name\_\_  # 获取分类器名称
137. start\_time = time.time()  # 开始计时
138. estimator.fit(X\_train, y\_train)  # 训练模型
139. y\_pred = estimator.predict(X\_test)  # 预测测试集
141. accuracy = metrics.accuracy\_score(y\_test, y\_pred)  # 计算准确率
143. end\_time = time.time()  # 结束计时
144. time\_consume = end\_time - start\_time  # 计算消耗时间
146. **return** clf\_name, estimator, accuracy, time\_consume

149. **for** i **in** range(50):
150. **print**(f"\n>>>第{i+1}次实验开始...")
152. knc = KNeighborsClassifier()
153. result = get\_text\_classification(knc, X\_train\_count, y\_train\_le, X\_test\_count, y\_test\_le)
154. estimator\_list.append(result[1])
155. score\_list.append(result[2])
156. time\_list.append(result[3])
158. dtc = DecisionTreeClassifier()
159. result = get\_text\_classification(dtc, X\_train\_count, y\_train\_le, X\_test\_count, y\_test\_le)
160. estimator\_list.append(result[1])
161. score\_list.append(result[2])
162. time\_list.append(result[3])
164. lgr = LogisticRegression()
165. result = get\_text\_classification(lgr, X\_train\_count, y\_train\_le, X\_test\_count, y\_test\_le)
166. estimator\_list.append(result[1])
167. score\_list.append(result[2])
168. time\_list.append(result[3])
170. df = pd.DataFrame()
171. df['分类器'] = estimator\_list
172. df['准确率'] = score\_list
173. df['消耗时间/s'] = time\_list
175. # 保存结果表格
176. save\_path = r'C:\Users\LENOVO\Desktop\多次实验取平均.xlsx'
177. df.to\_excel(save\_path, index=False)
179. **print**(f"\n>>>多次实验的平均结果已保存至'{save\_path}'")
181. classifiers = ['K近邻', '决策树', '逻辑回归']
182. accuracy = [0.665, 0.747, 0.890]
183. time\_consumed = [41.60,7.93, 20.21]
185. # 总体分析柱形图
186. plt.figure(figsize=(10, 10))
187. plt.bar(classifiers, accuracy, color='skyblue')
188. plt.title('不同分类器的准确率')
189. plt.xlabel('分类器')
190. plt.ylabel('准确率')
191. plt.xticks(rotation=45)
193. # 添加数值标签
194. **for** i **in** range(len(classifiers)):
195. plt.text(i, accuracy[i], str(accuracy[i]), ha='center', va='bottom')
197. plt.show()
199. # 绘制消耗时间柱状图
200. plt.figure(figsize=(10, 5))
201. plt.bar(classifiers, time\_consumed, color='lightgreen')
202. plt.title('不同分类器的消耗时间')
203. plt.xlabel('分类器')
204. plt.ylabel('消耗时间/s')
205. plt.xticks(rotation=45)
207. # 添加数值标签
208. **for** i **in** range(len(classifiers)):
209. plt.text(i, time\_consumed[i]+1, str(time\_consumed[i]), ha='center', va='bottom')

212. plt.show()
214. # 类别和指标数据
215. categories = ['Women', 'Sports', 'Campus', 'Literature']
216. precision = [0.89, 0.53, 0.78, 0.64]
217. recall = [0.77, 0.89, 0.58, 0.56]
218. f1\_score = [0.82, 0.67, 0.67, 0.60]
220. x = np.arange(len(categories))  # 柱状图的 x 坐标位置
222. # 个体分析柱形图
223. fig, ax = plt.subplots()
224. bar\_width = 0.25  # 柱状图的宽度
225. b1 = ax.bar(x, precision, bar\_width, label='Precision')
226. b2 = ax.bar(x + bar\_width, recall, bar\_width, label='Recall')
227. b3 = ax.bar(x + 2\*bar\_width, f1\_score, bar\_width, label='F1 Score')
229. # 在柱形图上方标出数值
230. **def** autolabel(bars):
231. **for** bar **in** bars:
232. height = bar.get\_height()
233. ax.annotate(f'{height:.2f}', # 标签内容，保留两位小数
234. xy=(bar.get\_x() + bar.get\_width() / 2, height), # 标签位置
235. xytext=(0, 3),  # 文本偏移量
236. textcoords="offset points",
237. ha='center', va='bottom') # 水平居中，垂直居下
239. autolabel(b1)
240. autolabel(b2)
241. autolabel(b3)
243. # 设置标签和标题
244. ax.set\_xlabel('Categories')
245. ax.set\_ylabel('Scores')
246. ax.set\_title('Performance Metrics by Category ')
247. ax.set\_xticks(x + bar\_width)
248. ax.set\_xticklabels(categories)
249. ax.legend()
251. plt.show()
253. #混淆矩阵热力图
254. **import** seaborn as sns
255. **import** matplotlib.pyplot as plt
256. **from** sklearn.metrics **import** confusion\_matrix
257. **import** numpy as np
259. # 假设confusion\_matrices是一个包含每个模型混淆矩阵的列表，顺序为KNN、决策树、逻辑回归
260. confusion\_matrices = [
261. np.array( [[99,0,16,0],[26,10,2,0],[ 8,0,23,0],[6,0,9,1]]),
262. np.array([[88,24,3,0],[2,34,1,1],[5,4,18,4],[4,2,1,9]]),
263. np.array( [[105,3,7,0],[0,36,1,1],[1,2,28,0],[3,1,3,9]])
264. ]
266. # 绘制每个模型的混淆矩阵热力图
267. **for** i **in** range(len(confusion\_matrices)):
268. plt.figure(figsize=(5, 4))
269. ax = sns.heatmap(confusion\_matrices[i], annot=True, cmap="YlGnBu", fmt='g')
270. ax.set\_title('Confusion Matrix for Model {}'.format(i+1))
271. ax.set\_xlabel('Predicted label')
272. ax.set\_ylabel('True label')
273. plt.show()

# 附录2:多次实验结果数据

详情见Excel文件“多次实验取平均”

# (IQB%FBX_9`ULNPAH6[OSGY