**目 录**

目录

[案例一 混凝土强度数据标准化处理 3](#_Toc148798082)

[【1】案例背景 3](#_Toc148798083)

[【2】方法陈述 3](#_Toc148798084)

[【3】实验代码 3](#_Toc148798085)

[步骤一 制作数据集CSV 3](#_Toc148798086)

[步骤二 多分类下的逻辑回归（K=1） 4](#_Toc148798087)

[步骤三 KNN近邻算法（K=1） 6](#_Toc148798088)

[步骤四 未优化的决策树算法 7](#_Toc148798089)

[步骤五 超参数调整后的决策树模型 9](#_Toc148798090)

[步骤六 朴素贝叶斯模型 12](#_Toc148798091)

[【4】结果分析 13](#_Toc148798092)

# 案例一 混凝土强度数据标准化处理

## 【1】案例背景

本题当中提供了214份玻璃的数据，并提供了这些玻璃的分类，要求我们以70%的样本作为训练集，以30%的样本作为测试集，利用本章提供的分类算法，选取合适的指标进行评测。**本部分由冯梓原完成**

## 【2】方法陈述

共用了逻辑回归、KNN算法、决策树算法、朴素贝叶斯算法四种算法进行计算。

## 【3】实验代码

### 步骤一 制作数据集CSV

import pandas as pd

data = pd.read\_table('glass.data', sep=',', header=None, index\_col=None)  # 加载本地数据

data.to\_csv('glass.csv', sep=',', header=True, index=False)  # data转成csv并保存为glass.csv文件

# 读取生成的glass.csv文件进行验证

csv = pd.read\_csv('glass.csv', sep=',')

# 创建一个空字典来存储列名及其对应的序号

column\_names\_dict = {}

# 打开 .names 文件以读取列名

with open('filtered\_glass\_names.names', 'r') as file:

    for line in file:

        column\_name = line.strip()  # 去除前导/尾随空格

        if column\_name:  # 判断行是否非空

            # 提取序号和列名

            parts = column\_name.split('. ', 1)

            if len(parts) == 2:

                index, name = parts

                # 使用 strip() 去除前导空格并尝试将序号转换为整数

                try:

                    index = int(index.strip())

                    # 将序号和列名添加到字典中

                    column\_names\_dict[index] = name

                except ValueError:

                    pass  # 忽略无法转换为整数的行

# 对列名按照序号进行排序

sorted\_column\_names = [column\_names\_dict[i] for i in sorted(column\_names\_dict.keys())]

# 打印排序后的列名

print(sorted\_column\_names)

# 读取数据文件并保存为CSV文件，使用逗号作为分隔符

data = pd.read\_csv('glass.csv', sep=',', index\_col=None)

df = data

df.columns = sorted\_column\_names

df.to\_csv('glass.csv', sep=',', index=False)

# 打印DataFrame

print(df)

### 步骤二 多分类下的逻辑回归（K=1）

#多分类的逻辑回归模型 （Confusion Matrix + Roc）

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import confusion\_matrix, roc\_curve, roc\_auc\_score, auc

import matplotlib.pyplot as plt

import numpy as np

# 1. 读取数据

data = pd.read\_csv('glass.csv')

# 2. 划分特征和标签

X = data.iloc[:, 1:-1]  # 选择中间九列作为特征

y = data['Type of glass: (class attribute)']  # 类别标签

# 3. 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 4. 创建并训练逻辑回归模型

model = LogisticRegression(max\_iter=1000, multi\_class='multinomial', solver='lbfgs')

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

# 5. 模型预测

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

# 6. 评估模型性能

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

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

classification\_rep = classification\_report(y\_test, y\_pred)

# 输出结果

print("Accuracy:", accuracy)

print("Confusion Matrix:")

print(confusion)

print("Classification Report:")

print(classification\_rep)

# 绘制混淆矩阵

plt.imshow(confusion, interpolation='nearest', cmap=plt.get\_cmap('Blues'))

plt.title('Confusion Matrix(LogisticRegression)')

plt.colorbar()

plt.xticks(np.arange(len(np.unique(y))), np.unique(y))

plt.yticks(np.arange(len(np.unique(y))), np.unique(y))

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# 绘制 ROC 曲线

n\_classes = len(model.classes\_)

# 为每个类别计算 ROC 曲线和 AUC

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(1, n\_classes + 1):  # 类别标签从1到7

    fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, y\_prob[:, i - 1])

    roc\_auc[i] = auc(fpr[i], tpr[i])

# 绘制 ROC 曲线

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

for i in range(1, n\_classes + 1):

    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Multi-Class Classification(LogisticRegression)')

plt.legend(loc='lower right')

plt.show()

### 步骤三 KNN近邻算法（K=1）

#KNN（K近邻算法，k=1）

import pandas as pd

import numpy as np

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, roc\_curve, roc\_auc\_score, auc

import matplotlib.pyplot as plt

# 读取数据

data = pd.read\_csv('glass.csv')

# 划分特征和标签

X = data.iloc[:, 1:-1]  # 特征列

y = data['Type of glass: (class attribute)']  # 标签列

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 创建K近邻模型

k = 1  # 设置K值

knn = KNeighborsClassifier(n\_neighbors=k)

# 拟合模型

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

# 预测测试集

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

# 6. 评估模型性能

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

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

classification\_rep = classification\_report(y\_test, y\_pred)

# 输出结果

print("Accuracy:", accuracy)

print("Confusion Matrix:")

print(confusion)

print("Classification Report:")

print(classification\_rep)

# 计算混淆矩阵

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

# 绘制混淆矩阵

plt.imshow(conf\_matrix, interpolation='nearest', cmap=plt.get\_cmap('Blues'))

plt.title('Confusion Matrix(KNN,k=1)')

plt.colorbar()

plt.xticks(np.arange(len(np.unique(y))), np.unique(y))

plt.yticks(np.arange(len(np.unique(y))), np.unique(y))

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

### 步骤四 未优化的决策树算法

#决策树（未优化）

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import roc\_curve, auc

# 1. 读取数据

data = pd.read\_csv('glass.csv')

# 2. 划分特征和标签

X = data.iloc[:, 1:-1]  # 选择中间九列作为特征

y = data['Type of glass: (class attribute)']  # 类别标签

# 3. 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 4. 创建并训练决策树模型

model = DecisionTreeClassifier()

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

# 5. 模型预测

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

# 6. 评估模型性能

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

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

classification\_rep = classification\_report(y\_test, y\_pred)

# 输出结果

print("Accuracy:", accuracy)

print("Confusion Matrix:")

print(confusion)

print("Classification Report:")

print(classification\_rep)

# 绘制混淆矩阵

plt.imshow(confusion, interpolation='nearest', cmap=plt.get\_cmap('Blues'))

plt.title('Confusion Matrix (Decision Tree)')

plt.colorbar()

# 获取类别的唯一标签和数量

unique\_labels, counts = np.unique(y, return\_counts=True)

# 将标签和数量一起显示在x轴和y轴

plt.xticks(np.arange(len(unique\_labels)), [f'{l} ({c})' for l, c in zip(unique\_labels, counts)], rotation=45)

plt.yticks(np.arange(len(unique\_labels)), [f'{l} ({c})' for l, c in zip(unique\_labels, counts)])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# 计算并绘制ROC曲线

n\_classes = len(np.unique(y))

y\_prob = model.predict\_proba(X\_test)

# 为每个类别计算ROC曲线和AUC

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(1, n\_classes + 1):  # 类别标签从1到7

    fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, y\_prob[:, i - 1])

    roc\_auc[i] = auc(fpr[i], tpr[i])

# 绘制ROC曲线

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

for i in range(1, n\_classes + 1):

    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Multi-Class Classification (Decision Tree)')

plt.legend(loc='lower right')

plt.show()

### 步骤五 超参数调整后的决策树模型

#超参数调整后的决策树

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import roc\_curve, auc

# 1. 读取数据

data = pd.read\_csv('glass.csv')

# 2. 划分特征和标签

X = data.iloc[:, 1:-1]  # 选择中间九列作为特征

y = data['Type of glass: (class attribute)']  # 类别标签

# 3. 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 4. 超参数调整

param\_grid = {

    'max\_depth': [None, 10, 20, 30],  # 调整树的最大深度

    'min\_samples\_split': [2, 5, 10],  # 调整分割节点的最小样本数

    'min\_samples\_leaf': [1, 2, 4]  # 调整叶节点的最小样本数

}

grid\_search = GridSearchCV(DecisionTreeClassifier(), param\_grid, cv=5, scoring='accuracy')

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

best\_params = grid\_search.best\_params\_

print("Best Hyperparameters:", best\_params)

# 5. 创建并训练决策树模型

model = DecisionTreeClassifier(\*\*best\_params)  # 使用最佳超参数组合

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

# 6. 模型预测

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

# 7. 评估模型性能

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

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

classification\_rep = classification\_report(y\_test, y\_pred)

# 输出结果

print("Accuracy:", accuracy)

print("Confusion Matrix:")

print(confusion)

print("Classification Report:")

print(classification\_rep)

# 绘制混淆矩阵

plt.imshow(confusion, interpolation='nearest', cmap=plt.get\_cmap('Blues'))

plt.title('Confusion Matrix (Decision Tree)')

plt.colorbar()

# 获取类别的唯一标签和数量

unique\_labels, counts = np.unique(y, return\_counts=True)

# 将标签和数量一起显示在x轴和y轴

plt.xticks(np.arange(len(unique\_labels)), [f'{l} ({c})' for l, c in zip(unique\_labels, counts)], rotation=45)

plt.yticks(np.arange(len(unique\_labels)), [f'{l} ({c})' for l, c in zip(unique\_labels, counts)])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# 计算并绘制ROC曲线

n\_classes = len(np.unique(y))

y\_prob = model.predict\_proba(X\_test)

# 为每个类别计算ROC曲线和AUC

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(1, n\_classes + 1):  # 类别标签从1到7

    fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, y\_prob[:, i - 1])

    roc\_auc[i] = auc(fpr[i], tpr[i])

# 绘制ROC曲线

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

for i in range(1, n\_classes + 1):

    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Multi-Class Classification (Decision Tree)')

plt.legend(loc='lower right')

plt.show()

### 步骤六 朴素贝叶斯模型

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix, roc\_curve, auc

# 1. 读取数据

data = pd.read\_csv('glass.csv')

# 2. 划分特征和标签

X = data.iloc[:, 1:-1]  # 选择中间九列作为特征

y = data['Type of glass: (class attribute)']  # 类别标签

# 3. 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 4. 创建并训练朴素贝叶斯模型

model = GaussianNB()

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

# 5. 模型预测

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

y\_prob = model.predict\_proba(X\_test)

# 6. 评估模型性能

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

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

classification\_rep = classification\_report(y\_test, y\_pred)

# 输出结果

print("Accuracy:", accuracy)

print("Confusion Matrix:")

print(confusion)

print("Classification Report:")

print(classification\_rep)

# 绘制混淆矩阵

plt.imshow(confusion, interpolation='nearest', cmap=plt.get\_cmap('Blues'))

plt.title('Confusion Matrix (Naive Bayes)')

plt.colorbar()

# 修改如下两行代码来设置刻度标签

tick\_marks = np.arange(len(np.unique(y)))

plt.xticks(tick\_marks, [1, 2, 3, 5, 6, 7])

plt.yticks(tick\_marks, [1, 2, 3, 5, 6, 7])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# 绘制 ROC 曲线

n\_classes = len(np.unique(y))

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

for i in range(1, n\_classes + 1):

    fpr, tpr, \_ = roc\_curve(y\_test == i, y\_prob[:, i - 1])

    roc\_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Multi-Class Classification (Naive Bayes)')

plt.legend(loc='lower right')

plt.show()

## 【4】结果分析

本案例是一个小数据集，且数据集比较密，现附上所有模型下的混淆矩阵的可视化图，以便后续结果说明

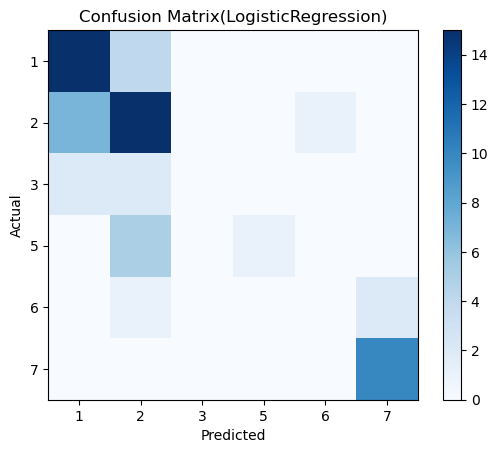
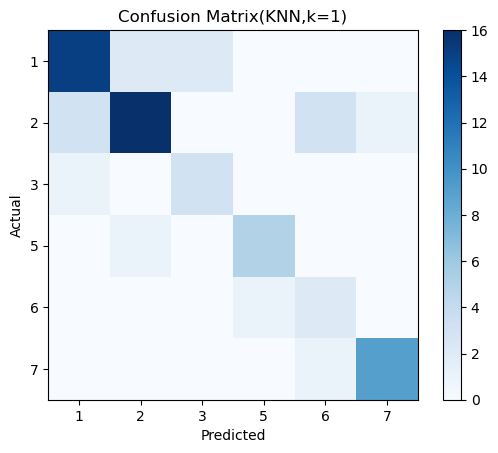


Figure 2

Figure 1

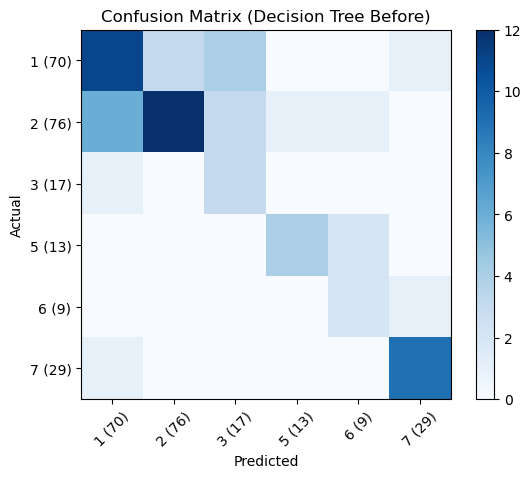


Figure 3

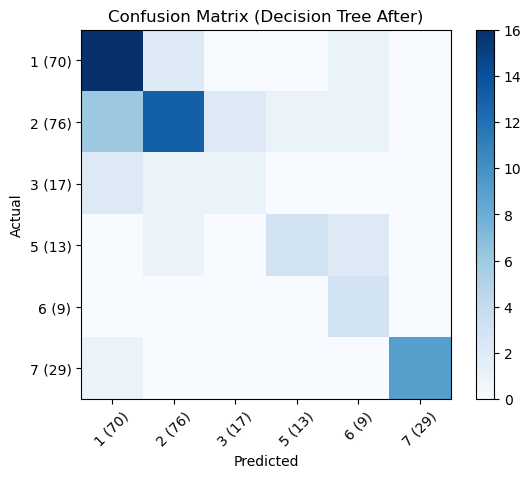


Figure 4

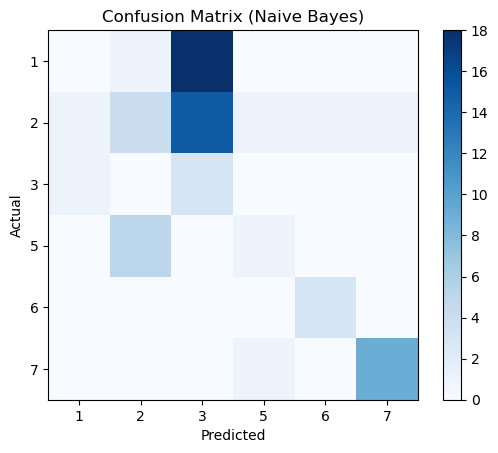
从图中的结果可以很清晰的看出，我首先运用的逻辑回归方法原用于二分类问题，通过修改参数用于了多分类问题，但是在小规模样本的数据分析上并不理想，而在较大样本的分析上符合分类的预期，最终的预测准确度为0.63。（Figure1）而KNN近邻的方法，在面对小规模数据集，数据非常密集的多分类问题上效果非常好，在很小规模的样本训练基础上，还能有一定预测准确率，最终的准确率为0.77（Figure2）。决策树模型一开始的预测结果非常一般，准确度只有0.63（Figure3），但是我引入了CV，通过选择最佳的超参数，更改树的深度，分支数量等，最终的预测准确度为0.69（Figure4），相比原模型有6%的提升。而朴素贝叶斯的效果最差，分类准确度只有0.31（Figure5）。

Figure 5