实验报告（4）

学号：202202206341 姓名：王子俊

1. **实验名称： 模型评估**
2. **实验目的**
3. 了解神经网络，决策树，knn等模型
4. 了解pca降维方法
5. 了解数据预处理和数据分析全过程
6. 了解不同模型之间的差异

**实验内容**

在给定的数据集上，训练神经网络，knn，决策树模型，输出预测结果及其准确度。

1. **实验步骤**

**模型评估与选择**

**一、选择数据集**

1. **数据集简介**

   - 数据集名称：UCI信用卡违约数据集（Credit Card Default Dataset）。

   - 样本数量：30,000。

   - 特征维度：23个特征，包括数值型和类别型特征。

1. **数据集基本特征分析**

   - 对每个特征进行统计分析（均值、标准差、缺失值比例等）。

   - 数据可视化：

     - 数值特征分布直方图。

     - 类别特征用柱状图表示分布情况。

     - 特征之间的相关性热力图。

**二、数据预处理**

1. **缺失值处理**

   - 对缺失值特征填充（均值、中位数或特定策略）。

1. **特征标准化**

   - 使用Min-Max缩放或Z-Score标准化所有数值特征。

1. **数据集划分**

   - 将数据集划分为训练集和测试集（8:2比例），并确保类别分布均匀。

1. **降维处理**

   - 使用PCA将维度从23降至适当维度（如2或3维）。

   - 保留未降维数据作为对照。

**三、分类器的分类**

**1. KNN分类**

   - 调整邻居数k（如3、5、7）。

   - 比较不同距离度量方式（欧几里得、曼哈顿等）。

**2. 神经网络分类**

   - 使用简单全连接神经网络：

     - 输入层：与特征维度一致。

     - 隐藏层：可设置为1-2层，节点数量灵活调整。

     - 输出层：1个节点（Sigmoid激活）用于二分类任务。

**3. 决策树分类**

   - 使用信息增益或基尼系数作为划分标准。

   - 设置最大深度或叶节点数以控制模型复杂度。

**四、模型评估与误差分析**

1. **纵向对比分析（未降维 vs. 降维）**

   - 每个分类器在未降维和降维数据上的**准确率**、**分类时间**等指标对比。

   - 绘制柱状图对比每个分类器在不同数据处理方式下的性能差异。

1. **横向对比分析（分类器之间的对比）**

   - 对比KNN、神经网络和决策树在未降维和降维数据上的**准确率**和**分类时间**。

   - 绘制条形图或折线图展示不同分类器性能。

1. **误差分析**

   - 绘制混淆矩阵，分析误分类情况。

   - 分析降维对分类器的实际影响：如信息丢失程度和计算效率变化。

1. **实验结果与分析**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

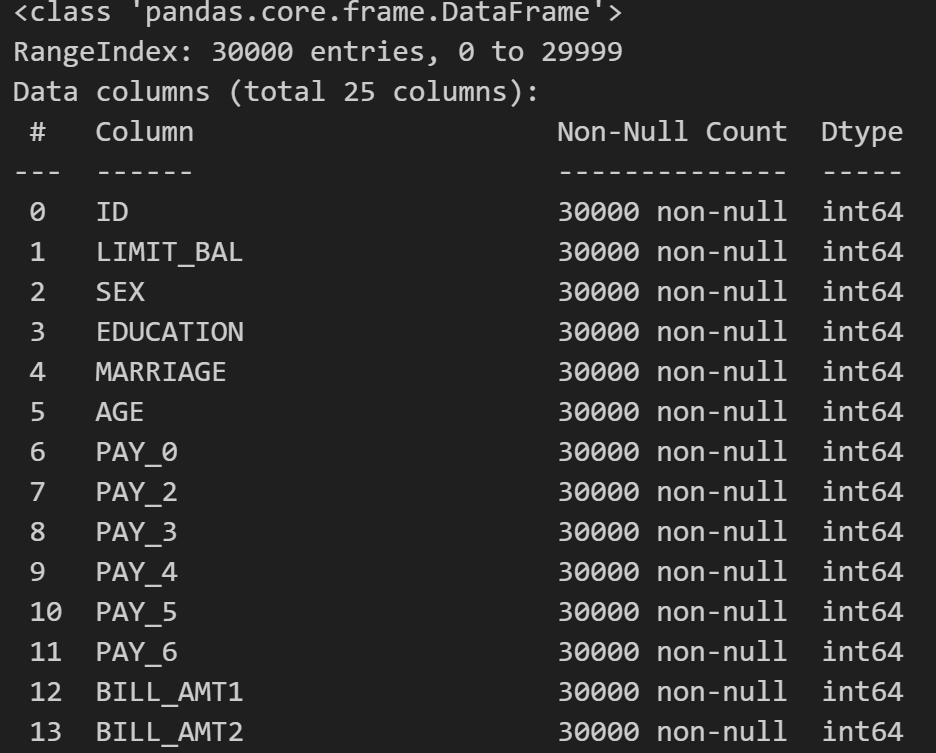
# 数据集加载

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%20clients.xls"

data = pd.read\_excel(url, header=1, skiprows=0)

# 查看数据的基本信息

print(data.info())



# 计算每个特征的基本统计量

summary\_stats = data.describe().transpose()

# 计算每个特征的缺失值比例

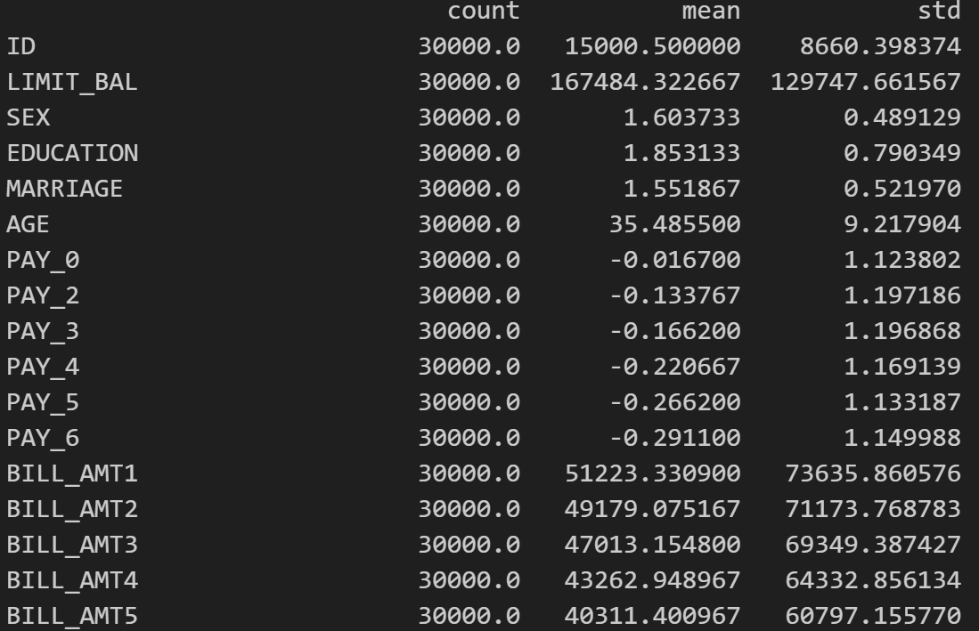
missing\_values = data.isnull().mean() \* 100

# 合并统计量和缺失值比例

summary\_stats['missing\_percentage'] = missing\_values

# 输出统计分析结果

print(summary\_stats)



# 数据清理

data.rename(columns=lambda x: x.strip(), inplace=True)  # 清理列名

data.rename(columns={"default payment next month": "default"}, inplace=True)

# 数据集信息

print(data.info())

print(data.describe())

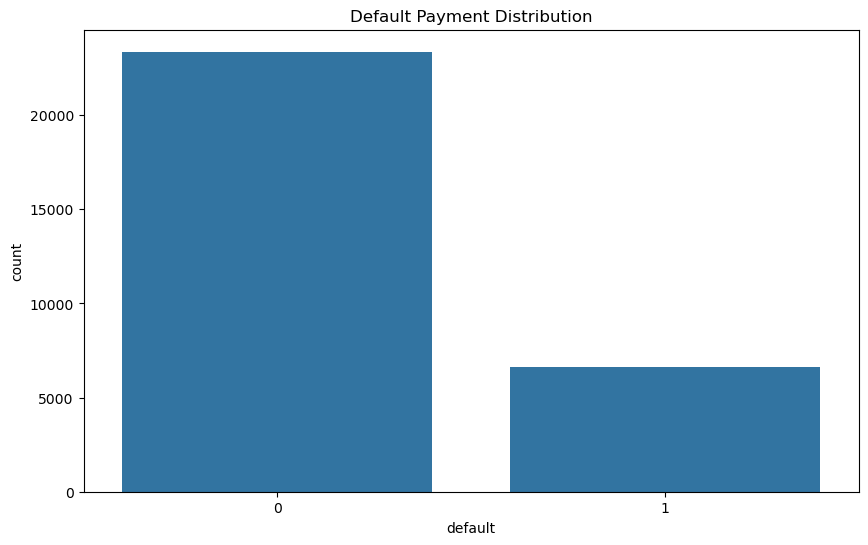
# 数据分布可视化

plt.figure(figsize=(10, 6))

sns.countplot(x="default", data=data)

plt.title("Default Payment Distribution")

plt.show()



# 数值特征的分布直方图

numeric\_features = data.select\_dtypes(include=['float64', 'int64']).columns

plt.figure(figsize=(12, 10))

for i, feature in enumerate(numeric\_features):

    plt.subplot(3, 3, i + 1)

    sns.histplot(data[feature], kde=True, bins=30)

    plt.title(f'Distribution of {feature}')

plt.tight\_layout()

plt.show()

# 类别特征的分布柱状图

categorical\_features = data.select\_dtypes(include=['object']).columns

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

for i, feature in enumerate(categorical\_features):

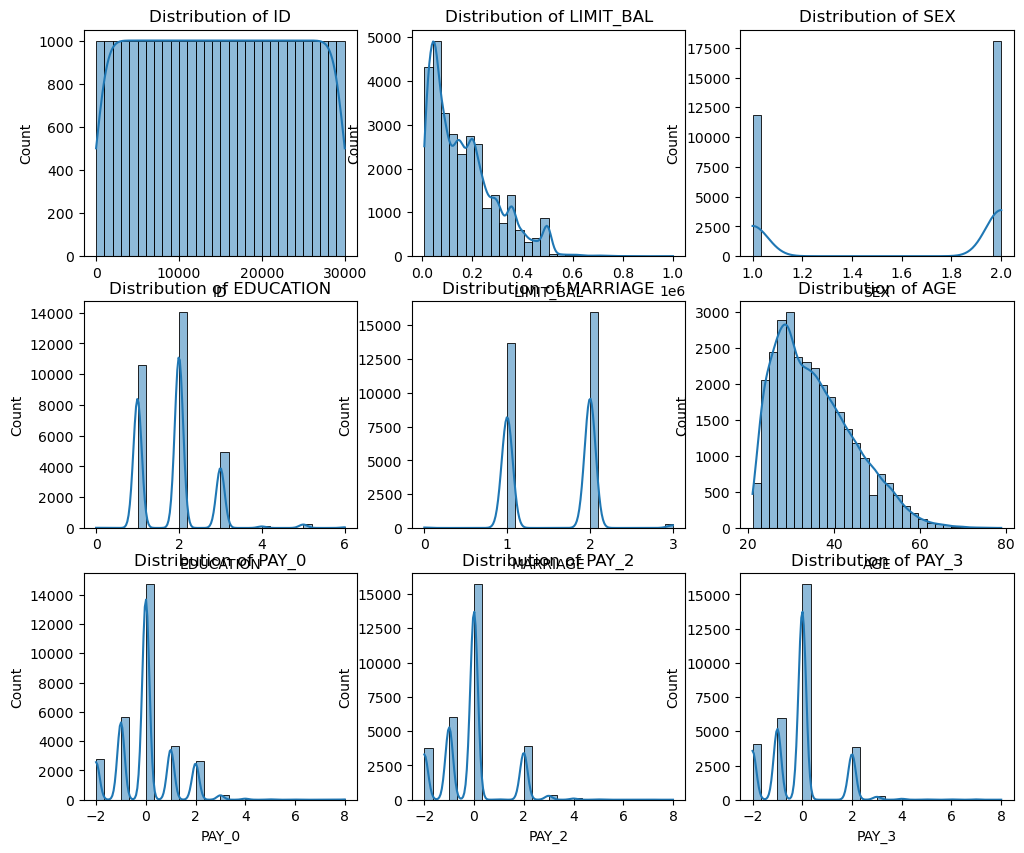
    plt.subplot(2, 3, i + 1)

    sns.countplot(x=data[feature], palette="Set2")

    plt.title(f'Distribution of {feature}')

plt.tight\_layout()

plt.show()



from sklearn.preprocessing import StandardScaler

# 提取特征与目标变量

X = data.iloc[:, :-1]

y = data["default"]

# 标准化特征

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

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

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42, stratify=y)

from sklearn.decomposition import PCA

# PCA降维至3维

pca = PCA(n\_components=3)

X\_train\_pca = pca.fit\_transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

print("降维后特征形状：", X\_train\_pca.shape)

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# 创建 3D 图形

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

# 绘制不同类别的点

categories = [0, 1]  # 假设目标变量有两类

colors = ['blue', 'orange']

labels = ['No Default', 'Default']

for category, color, label in zip(categories, colors, labels):

    indices = y\_train == category

    ax.scatter(

        X\_train\_pca[indices, 0], X\_train\_pca[indices, 1], X\_train\_pca[indices, 2],

        c=color, label=label, alpha=0.7, edgecolor='k'

    )

# 设置轴标签

ax.set\_xlabel('Principal Component 1')

ax.set\_ylabel('Principal Component 2')

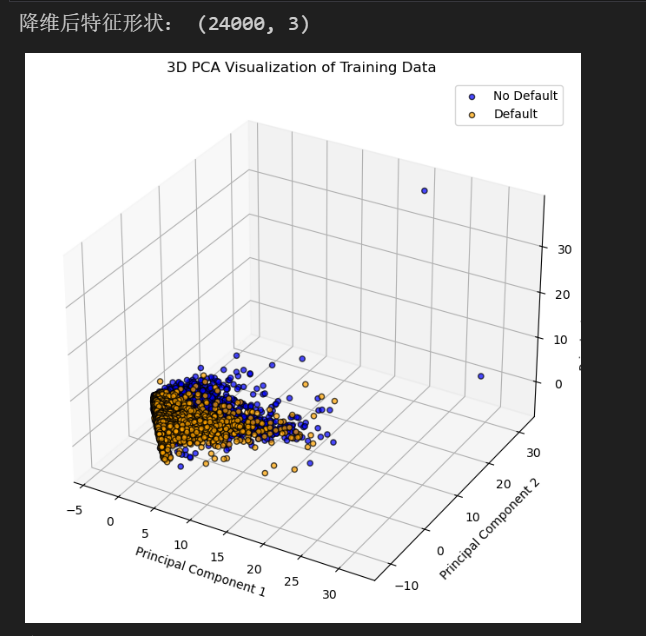
ax.set\_zlabel('Principal Component 3')

# 图例和标题

ax.legend()

ax.set\_title("3D PCA Visualization of Training Data")

plt.show()



# 记录开始时间

start\_time = time.time()

# 创建KNN模型并训练

knn = KNeighborsClassifier(n\_neighbors=5)

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

# 记录结束时间

end\_time = time.time()

# 计算训练时间

training\_time\_before = end\_time - start\_time

print(f"KNN训练时间（降维前）：{training\_time\_before:.4f}秒")

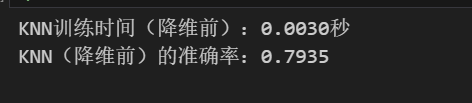
# 预测结果

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

# 计算准确率

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

print(f"KNN（降维前）的准确率：{accuracy\_knn\_before:.4f}")



# 记录开始时间

start\_time = time.time()

# 创建KNN模型并训练

knn\_pca = KNeighborsClassifier(n\_neighbors=5)

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

# 记录结束时间

end\_time = time.time()

# 计算训练时间

training\_time\_after = end\_time - start\_time

print(f"KNN训练时间（降维后）：{training\_time\_after:.4f}秒")

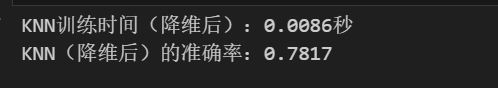
# 预测结果

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

# 计算准确率

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

print(f"KNN（降维后）的准确率：{accuracy\_knn\_pca:.4f}")



import time

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# 记录开始时间

start\_time\_nn\_before = time.time()

# 创建神经网络模型并训练（降维前）

mlp\_before = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=300)

mlp\_before.fit(X\_train, y\_train)

# 记录结束时间

end\_time\_nn\_before = time.time()

# 计算训练时间

training\_time\_nn\_before = end\_time\_nn\_before - start\_time\_nn\_before

print(f"神经网络训练时间（降维前）：{training\_time\_nn\_before:.4f}秒")

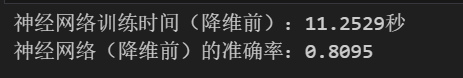
# 预测结果

y\_pred\_nn\_before = mlp\_before.predict(X\_test)

# 计算准确率

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

print(f"神经网络（降维前）的准确率：{accuracy\_nn\_before:.4f}")



# 记录开始时间

start\_time\_nn\_after = time.time()

# 创建神经网络模型并训练（降维后）

mlp\_after = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=300)

mlp\_after.fit(X\_train\_pca, y\_train)

# 记录结束时间

end\_time\_nn\_after = time.time()

# 计算训练时间

training\_time\_nn\_after = end\_time\_nn\_after - start\_time\_nn\_after

print(f"神经网络训练时间（降维后）：{training\_time\_nn\_after:.4f}秒")

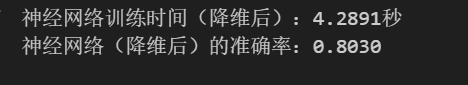
# 预测结果

y\_pred\_nn\_after = mlp\_after.predict(X\_test\_pca)

# 计算准确率

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

print(f"神经网络（降维后）的准确率：{accuracy\_nn\_after:.4f}")



import time

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# 记录开始时间

start\_time\_dt\_before = time.time()

# 创建决策树模型并训练（降维前）

dt\_before = DecisionTreeClassifier(criterion="entropy")  # 使用信息增益

dt\_before.fit(X\_train, y\_train)

# 记录结束时间

end\_time\_dt\_before = time.time()

# 计算训练时间

training\_time\_dt\_before = end\_time\_dt\_before - start\_time\_dt\_before

print(f"决策树训练时间（降维前）：{training\_time\_dt\_before:.4f}秒")

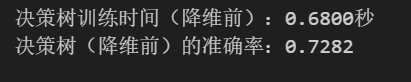
# 预测结果

y\_pred\_dt\_before = dt\_before.predict(X\_test)

# 计算准确率

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

print(f"决策树（降维前）的准确率：{accuracy\_dt\_before:.4f}")



# 记录开始时间

start\_time\_dt\_after = time.time()

# 创建决策树模型并训练（降维后）

dt\_after = DecisionTreeClassifier(criterion="entropy")  # 使用信息增益

dt\_after.fit(X\_train\_pca, y\_train)

# 记录结束时间

end\_time\_dt\_after = time.time()

# 计算训练时间

training\_time\_dt\_after = end\_time\_dt\_after - start\_time\_dt\_after

print(f"决策树训练时间（降维后）：{training\_time\_dt\_after:.4f}秒")

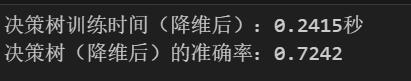
# 预测结果

y\_pred\_dt\_after = dt\_after.predict(X\_test\_pca)

# 计算准确率

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

print(f"决策树（降维后）的准确率：{accuracy\_dt\_after:.4f}")



# 计算准确率与训练时间的比率

accuracy\_time\_ratio = [

    accuracy\_knn\_before / training\_time\_before,

    accuracy\_knn\_pca / training\_time\_after,

    accuracy\_nn\_before / training\_time\_nn\_before,

    accuracy\_nn\_after / training\_time\_nn\_after,

    accuracy\_dt\_before / training\_time\_dt\_before,

    accuracy\_dt\_after / training\_time\_dt\_after

]

# 结果存储：KNN、神经网络和决策树模型对比，包括比率

results = {

    "Model": [

        "KNN (Full Features)", "KNN (PCA 3D)",

        "Neural Network (Full Features)", "Neural Network (PCA 3D)",

        "Decision Tree (Full Features)", "Decision Tree (PCA 3D)"

    ],

    "Accuracy": [

        accuracy\_knn\_before, accuracy\_knn\_pca,

        accuracy\_nn\_before, accuracy\_nn\_after,

        accuracy\_dt\_before, accuracy\_dt\_after

    ],

    "Training Time (seconds)": [

        training\_time\_before, training\_time\_after,

        training\_time\_nn\_before, training\_time\_nn\_after,

        training\_time\_dt\_before, training\_time\_dt\_after

    ],

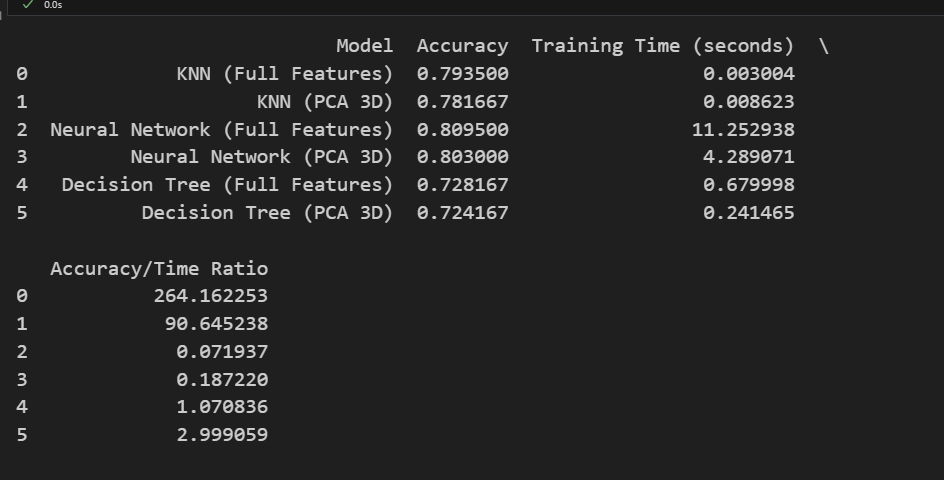
    "Accuracy/Time Ratio": accuracy\_time\_ratio

}

# 输出对比结果

df\_results = pd.DataFrame(results)

print(df\_results)



import numpy as np

import matplotlib.pyplot as plt

# 模型名称

models = [

    "KNN (Full Features)", "KNN (PCA 3D)",

    "Neural Network (Full Features)", "Neural Network (PCA 3D)",

    "Decision Tree (Full Features)", "Decision Tree (PCA 3D)"

]

# 准确率和训练时间

accuracies = [

    accuracy\_knn\_before, accuracy\_knn\_pca,

    accuracy\_nn\_before, accuracy\_nn\_after,

    accuracy\_dt\_before, accuracy\_dt\_after

]

training\_times = [

    training\_time\_before, training\_time\_after,

    training\_time\_nn\_before, training\_time\_nn\_after,

    training\_time\_dt\_before, training\_time\_dt\_after

]

# 对训练时间取自然对数

log\_training\_times = np.log(training\_times)

# 创建散点图（训练时间为横轴，准确率为纵轴）

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

plt.scatter(log\_training\_times, accuracies, color='blue', s=100)

# 为每个点添加标签

for i, model in enumerate(models):

    plt.text(log\_training\_times[i] + 0.02, accuracies[i], model, fontsize=9, ha='left', va='bottom')

# 设置图形标题和轴标签

plt.title('Log(Training Time) vs Accuracy for Different Models', fontsize=14)

plt.xlabel('Log(Training Time) (seconds)', fontsize=12)

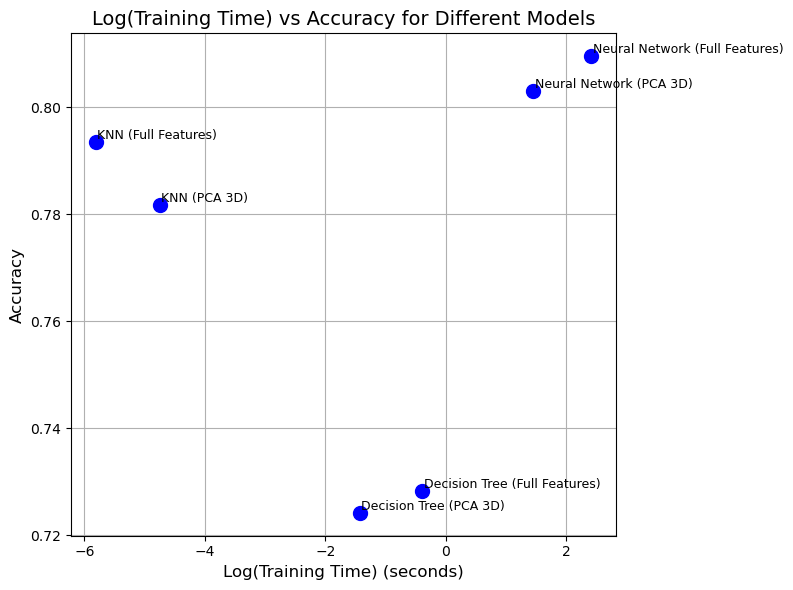
plt.ylabel('Accuracy', fontsize=12)

# 显示图形

plt.grid(True)

plt.tight\_layout()

plt.show()

****

1. **实验总结**

根据上面的数据，我们可以对三种模型（KNN、神经网络和决策树）在完整特征和降维后特征上的表现进行分析。首先，从准确率来看，神经网络在完整特征上的表现最好，达到了 **0.8095**，这表明神经网络能够更好地捕捉数据中的复杂模式。然而，训练时间相对较长，为 **11.2529 秒**，这是由于神经网络的计算复杂性较高，特别是在特征维度较大的情况下，训练过程需要进行多次迭代和反向传播。

KNN在使用完整特征时的准确率为 **0.7935**，相比神经网络略低，但训练时间非常短，仅为 **0.003004 秒**，表明KNN在训练时的计算开销较小。通过PCA降维后，KNN的准确率略有下降，降至 **0.7817**，训练时间却增加至 **0.008623 秒**。这种情况表明，尽管降维减少了特征空间的复杂度，PCA降维过程本身仍然需要一定的计算时间，导致训练时间有所增加，而准确率的轻微下降可能是由于降维过程中丢失了部分有用的信息。

决策树模型在完整特征下的准确率为 **0.7282**，在三种模型中表现最差，但训练时间较长，达到了 **0.679998 秒**，这可能是由于决策树算法在训练时需要根据不同特征计算最佳分割点，导致计算复杂度较高。降维后，决策树的准确率略有下降，降至 **0.7242**，但训练时间大幅减少，降至 **0.2415 秒**。这种变化表明，降维减少了特征空间的复杂度，从而使得决策树的训练更加高效，尽管在准确性上有一定牺牲。

总体而言，神经网络模型在准确性上表现最好，但代价是较长的训练时间。KNN模型虽然准确率略低，但在使用完整特征时具有非常短的训练时间，适合在对时间要求较高的场合使用。决策树则在准确率上处于劣势，但通过降维显著减少了训练时间，适合在计算资源有限的情况下使用。降维对所有模型的训练时间都有所优化，但会在一定程度上牺牲准确性，因此在实际应用中需要根据具体需求做出平衡选择。如果对准确率的要求较高，神经网络会是最佳选择；如果计算时间是优先考虑的因素，KNN和决策树则是更合适的选择。