实训 1：使用 sklearn 处理竞标行为数据集

步骤 1：导入必要的库和数据集

源程序：

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 读取数据集
data = pd.read_csv('shill_bidding.csv')

# 查看数据集的前几行
display(data.head())
```

过程性结果：

步骤1：导入必要的库和数据集

[3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 读取数据集
data = pd.read_csv('shill_bidding.csv')

# 查看数据集的前几行
display(data.head())
```

	Record_ID	Auction_ID	Bidder_ID	Bidder_Tendency	Bidding_Ratio	Successive_Outbidding	Last_Bidding	Auction_Bids	Starting_Price_Average	Early_Bidding	Winning_Ratio	Auction_Duration	Class
0	1	732	_***i	0.200000	0.400000	0.0	0.000028	0.0	0.993593	0.000028	0.666667	5	0
1	2	732	g***r	0.024390	0.200000	0.0	0.013123	0.0	0.993593	0.013123	0.944444	5	0
2	3	732	t***p	0.142857	0.200000	0.0	0.003042	0.0	0.993593	0.003042	1.000000	5	0
3	4	732	7***n	0.100000	0.200000	0.0	0.097477	0.0	0.993593	0.097477	1.000000	5	0
4	5	900	z***z	0.051282	0.222222	0.0	0.001318	0.0	0.000000	0.001242	0.500000	7	0

步骤 2：数据预处理和特征标准化

源程序：

```
# 提取特征和标签
X = data.drop(columns=['Class', 'Record_ID', 'Auction_ID', 'Bidder_ID'])
y = data['Class']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

标准化特征

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

display(X_train_scaled)

过程性结果:

步骤2：数据预处理和特征标准化

[5]:
```
# 提取特征和标签
X = data.drop(columns=['Class', 'Record_ID', 'Auction_ID', 'Bidder_ID'])
y = data['Class']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 标准化特征
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

display(X_train_scaled)
```

```
array([[ 1.71097905,  2.20429041,  3.18828508, ...,  1.21569663,
         0.7867721 ,  0.95691672],
       [ 1.78718435, -0.26571534,  1.40903901, ...,  0.00742892,
        -0.84744794, -1.47066529],
       [-0.34070222, -0.34432524, -0.37020705, ...,  0.03188474,
        -0.84744794, -1.47066529],
       ...,
       [-0.64087431, -0.34432524, -0.37020705, ...,  0.60665508,
        -0.84744794,  0.95691672],
       [-0.09889693,  0.10487418, -0.37020705, ..., -0.89745647,
         0.7867721 , -0.66147128],
       [-0.12404468, -0.2871544 , -0.37020705, ...,  1.25328314,
        -0.84744794,  0.95691672]])
```

步骤 3：PCA 降维

源程序:

设定 PCA 降维后的特征数

pca = PCA(n_components=0.999)

X_train_pca = pca.fit_transform(X_train_scaled)

X_test_pca = pca.transform(X_test_scaled)

print(f'PCA 后的特征数: {X_train_pca.shape[1]}')

过程性结果：

步骤3：PCA降维

[6]:
```
# 设定PCA降维后的特征数
pca = PCA(n_components=0.999)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print(f'PCA后的特征数: {X_train_pca.shape[1]}')
```

```
PCA后的特征数: 9
```

结论：

通过 PCA 降维，我们成功将特征数量从原来的 11 个减少到 9 个主成分，同时保留了 99.9%的信息量。这表明 PCA 在减少特征维度的同时，能够保留大部分数据的变异性，达到降维的效果。

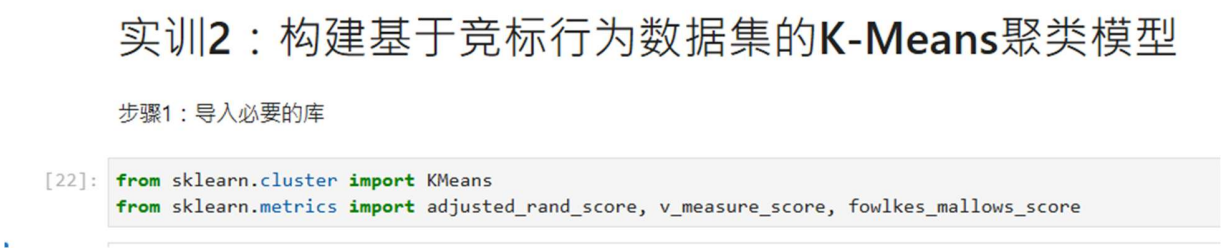
实训 2：构建基于竞标行为数据集的 K-Means 聚类模型

步骤 1：导入必要的库

源程序：

```
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score, v_measure_score, fowlkes_mallows_score
```

过程性结果：



（当然是导入成功了这能有什么结果呀~）

步骤 2：构建 K-Means 模型、并按要求的方法给出评分

源程序：

```
# 选择特征：竞标者倾向、竞标比率、连续竞标
features = ['Bidder_Tendency', 'Bidding_Ratio', 'Successive_Outbidding']
X_selected = data[features]

# K-Means 聚类
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_selected)
labels = kmeans.labels_

# 评估聚类结果
ari = adjusted_rand_score(data['Class'], labels)
v_measure = v_measure_score(data['Class'], labels)
fmi = fowlkes_mallows_score(data['Class'], labels)
```

print(f'ARI: {ari}, V-Measure: {v_measure}, FMI: {fmi}')

过程性结果：

步骤2：构建K-Means模型、并按要求的方法给出评分

[11]:
```
# 选择特征：竞标者倾向、竞标比率、连续竞标
features = ['Bidder_Tendency', 'Bidding_Ratio', 'Successive_Outbidding']
X_selected = data[features]

# K-Means聚类
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_selected)
labels = kmeans.labels_

# 评估聚类结果
ari = adjusted_rand_score(data['Class'], labels)
v_measure = v_measure_score(data['Class'], labels)
fmi = fowlkes_mallows_score(data['Class'], labels)

print(f'ARI: {ari}, V-Measure: {v_measure}, FMI: {fmi}')
```

```
C:\Users\ezra\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The def
ly to suppress the warning
  warnings.warn(
ARI: 0.8413594373186039, V-Measure: 0.7368999600504711, FMI: 0.9667623498850926
```

结论：

K-Means 聚类模型在对竞标行为进行分类时表现出较好的聚类效果。具体评分如下：

调整兰德指数（ARI）：0.841

V-Measure：0.737

FMI：0.967

K-Means 模型显然能较好地将正常竞标行为和异常竞标行为区分开来，这样还是比较合适的。但准确率并不是最高。

实训 3：构建基于竞标行为数据集的支持向量机分类模型

步骤 1：导入必要的库

源程序:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

过程性结果:

导入成功了嗯嗯。

步骤 2：构建支持向量机模型

源程序:

```
# 标准化特征
X_scaled = scaler.fit_transform(X)
# 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)
# 构建 SVM 模型
svm = SVC()
svm.fit(X_train, y_train)
# 预测和评估
y_pred = svm.predict(X_test)
report = classification_report(y_test, y_pred)
print(report)
```

过程性结果：

实训3：构建基于竞标行为数据集的支持向量机分类模型

步骤1：导入必要的库

[13]:
```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

步骤2：构建支持向量机模型

[15]:
```
# 标准化特征
X_scaled = scaler.fit_transform(X)

# 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)

# 构建SVM模型
svm = SVC()
svm.fit(X_train, y_train)

# 预测和评估
y_pred = svm.predict(X_test)
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      1133
           1       0.88      0.95      0.91       132

    accuracy                           0.98      1265
   macro avg       0.94      0.97      0.95      1265
weighted avg       0.98      0.98      0.98      1265
```

结论：

支持向量机分类模型在分类正常竞标行为和异常竞标行为时表现出较高的准确率。具体分类报告如下：

总体准确率：98%

异常行为（Class 1）的精确率：88%，召回率：95%，F1 分数：91%

正常行为（Class 0）的精确率：99%，召回率：98%，F1 分数：99%

这些结果表明 SVM 模型能够较好地区分正常和异常竞标行为，尤其是对异常行为的召回率较高。

实训 4：构建基于竞标行为数据集的回归模型

步骤 1：导入必要的库

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

过程性结果：

当然是导入成功了呀。

步骤 2：构建线性回归模型并评估 MSE、R^2 特征

```
# 选择目标变量和特征
y = data['Class']
X = data.drop(columns=['Class', 'Record_ID', 'Auction_ID', 'Bidder_ID'])
# 标准化特征
X_scaled = scaler.fit_transform(X)
# 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)
# 构建线性回归模型
lr = LinearRegression()
lr.fit(X_train, y_train)
# 预测和评估
y_pred = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'MSE: {mse}, R^2: {r2}')
```

过程性结果：

实训4：构建基于竞标行为数据集的回归模型

步骤1：导入必要的库

[18]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

步骤2：构建线性回归模型并评估 MSE、R^2 特征

[21]:
```
# 选择目标变量和特征
y = data['Class']
X = data.drop(columns=['Class', 'Record_ID', 'Auction_ID', 'Bidder_ID'])

# 标准化特征
X_scaled = scaler.fit_transform(X)

# 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)

# 构建线性回归模型
lr = LinearRegression()
lr.fit(X_train, y_train)

# 预测和评估
y_pred = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MSE: {mse}, R^2: {r2}')
```

```
MSE: 0.0205196171580357, R^2: 0.780443416735419
```

结论：

线性回归模型用于分类问题的效果不太理想。具体评估指标如下：

均方误差（MSE）：0.021

决定系数（R^2）：0.780

R^2值表明模型能够解释大部分数据的变异性，但 MSE 较高表明模型在分类问题上的误差很大。

因此，线性回归模型不太适合用于竞标行为的分类问题，更加复杂的非线性模型应该能更好地捕捉数据特征。（比如之前的聚类或 SVM 模型。）