

**人工智能导论-实验报告**

**题目：鸢尾花数据集的分析和测试**

院 系 软件学院

专业班级

姓 名

学 号

指导教师 沈刚 /李祥友

2024年 12月 20 日

**目录**

**1 引言**

1.1鸢尾花（Iris）数据集问题背景和研究意义

1.1.1问题背景

1.1.2研究意义

1.2实验目的和实验内容

1.2.1实验目的

1.2.2实验内容

**2问题的表示和求解算法**

2.1 鸢尾花（Iris）数据集的展示

2.2 鸢尾花（Iris）数据集的求解算法

**3实验设计**

3.1 实验目标和评价指标

3.2 不同算法的代码分析和实现

3.2.1 k-mean算法

3.2.2 Hierarchical Clustering算法

**4实验结果**

4.1 使用的不同方法及其特点

4.2 实验结果的展示

**5实验分析与讨论**

5.1 性能比较

5.2 定性和定量分析

5.3 结果可视化

5.3.1 性能比较可视化

5.3.2 分类模型混淆矩阵

5.3.3 箱线图

5.3.4 直方图

5.3.5 K-Means轮廓系数随簇数变化图

5.3.6 径向图（雷达图）

**6结论**

6.1 实验结果总结

6.2 改进思考与建议

**7参考文献**

**8附录**

8.1 实验代码

8.2 实验数据表和图表

## 1 引言

### 1.1 鸢尾花（Iris）数据集问题背景和研究意义

#### 1.1.1 问题背景

鸢尾花（Iris）数据集是统计学家罗纳德·费舍尔（Ronald A. Fisher）于1936年提出的经典数据集，广泛用于模式识别和机器学习领域。该数据集包含150个样本，分为三类鸢尾花：山鸢尾（Iris setosa）、变色鸢尾（Iris versicolor）和维吉尼亚鸢尾（Iris virginica）。每个样本有四个特征：萼片长度、萼片宽度、花瓣长度和花瓣宽度。

#### 1.1.2 研究意义

鸢尾花数据集作为一个多分类问题的典范，具有以下研究意义：

1. **算法验证**：为各种分类和聚类算法提供了一个简单且易于理解的测试平台。
2. **教学用途**：在机器学习和数据挖掘课程中，常用于演示算法的应用和效果。
3. **算法比较**：通过对比不同算法在同一数据集上的表现，评估其优缺点和适用性。

### 1.2 实验目的和实验内容

#### 1.2.1 实验目的

本实验旨在通过对鸢尾花数据集的分析，比较不同的聚类算法和分类算法的性能，探讨参数选择对模型性能的影响，并通过可视化手段展示分析结果。

#### 1.2.2 实验内容

1. 数据预处理与探索性分析。
2. 使用K-Means和层次聚类算法进行聚类分析，并评估其性能。
3. 使用逻辑回归和决策树分类算法进行分类分析，比较其准确率和其他评价指标。
4. 参数调优，分析不同参数设置对模型性能的影响。
5. 结果的可视化展示与分析。

## 2 问题的表示和求解算法

### 2.1 鸢尾花（Iris）数据集的展示

鸢尾花数据集包含150个样本，每个样本有四个特征和一个类别标签。具体特征如下：

* **萼片长度（Sepal Length）**：单位为厘米。
* **萼片宽度（Sepal Width）**：单位为厘米。
* **花瓣长度（Petal Length）**：单位为厘米。
* **花瓣宽度（Petal Width）**：单位为厘米。
* **类别（Species）**：分为三类：Iris setosa、Iris versicolor、Iris virginica。



### 图2-1 鸢尾花（Iris）数据集的展示

### 2.2 鸢尾花（Iris）数据集的求解算法

针对鸢尾花数据集的分析，本实验采用以下算法：

**聚类算法**：

**K-Means聚类**：通过最小化簇内样本到簇中心的距离，将数据划分为预定数量的簇。

**层次聚类（Hierarchical Clustering）**：通过构建层次树状结构，将数据逐步合并或分割成簇。

**分类算法**：

**逻辑回归（Logistic Regression）**：适用于二分类或多分类问题，通过估计样本属于某一类别的概率进行分类。

**决策树（Decision Tree）**：通过树状结构进行决策，适用于分类和回归任务，具有良好的解释性。

## 3 实验设计

### 3.1 实验目标和评价指标

#### 实验目标

1. 对鸢尾花数据集进行聚类和分类分析，比较不同算法的性能。
2. 探讨参数选择对模型性能的影响，寻找最佳参数组合。
3. 通过可视化手段展示聚类和分类结果，增强理解。

#### 评价指标

**聚类评价指标**：

**轮廓系数（Silhouette Score）**：衡量样本在其簇内的紧密度与其最近邻簇的分离度，取值范围为[-1,1]，值越大表示聚类效果越好。

**调整后的兰德指数（Adjusted Rand Index, ARI）**：衡量聚类结果与真实标签的相似度，取值范围为[-1,1]，值越大表示聚类效果越好。

**分类评价指标**：

**准确率（Accuracy）**：正确分类的样本数占总样本数的比例。

**混淆矩阵（Confusion Matrix）**：展示分类器在各类别上的预测情况，便于计算其他指标如精确率、召回率和F1分数。

**分类报告（Classification Report）**：综合展示精确率、召回率和F1分数等指标。

### 3.2 不同算法的代码分析和实现

#### 3.2.1 K-Means算法

**算法原理**：

K-Means是一种基于距离的聚类算法，旨在将数据划分为K个簇，每个簇由其簇中心（质心）代表。算法通过迭代优化簇中心的位置，使得簇内样本之间的距离最小化。

**实现步骤**：

1. 选择K个初始簇中心。
2. 将每个样本分配到最近的簇中心。
3. 更新每个簇的中心为其所有样本的均值。
4. 重复步骤2和3，直到簇中心不再发生显著变化或达到最大迭代次数。

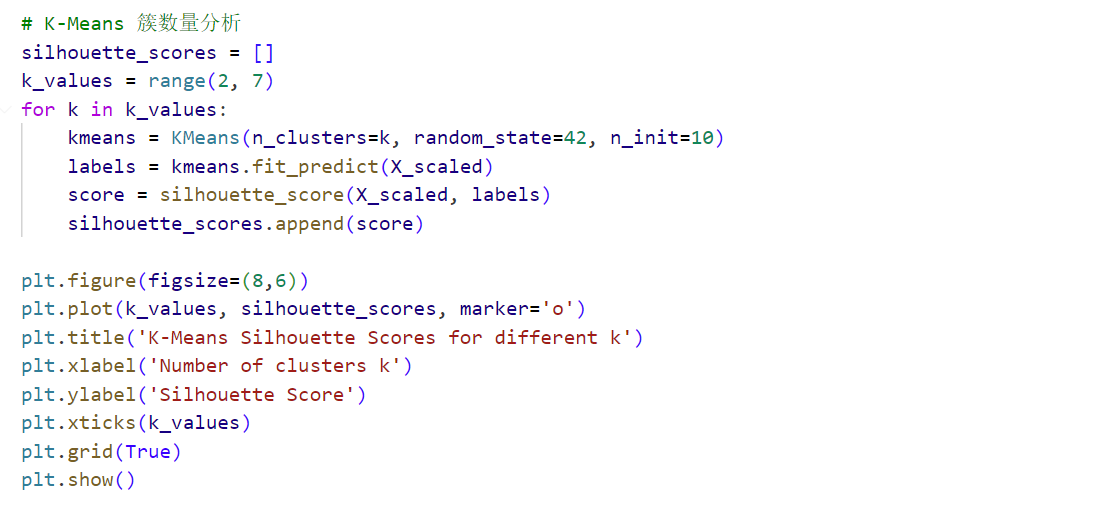


图3-1 K-mean算法

#### 3.2.2 层次聚类算法

**算法原理**：

层次聚类通过构建层次树状结构来实现数据的聚类。主要有两种方法：

1. **自下而上（凝聚的）**：从每个样本作为一个独立的簇开始，逐步合并最相似的簇，直到达到预定的簇数。
2. **自上而下（分裂的）**：从所有样本作为一个簇开始，逐步分裂最不相似的簇，直到达到预定的簇数。

**实现步骤（以凝聚方法为例）**：

1. 初始化，每个样本作为一个单独的簇。
2. 计算所有簇之间的相似度（或距离）。
3. 合并最相似的两个簇。
4. 重复步骤2和3，直到达到预定的簇数。



图3-2 层次聚类算法

## 4 实验结果

### 4.1 使用的不同方法及其特点

**K-Means聚类**：

**优点**：简单高效，适用于大规模数据集。

**缺点**：需要预先指定K值，对初始值敏感，容易受噪声和异常值影响。

**层次聚类**：

**优点**：无需预先指定簇数，适合发现数据的层次结构。

**缺点**：计算复杂度高，不适合大规模数据集，对噪声和异常值敏感。

**逻辑回归**：

**优点**：模型简单，计算效率高，具有良好的解释性。

**缺点**：对线性可分性要求较高，难以处理复杂的非线性关系。

**决策树**：

**优点**：能够处理非线性关系，易于理解和解释，能够处理混合类型的数据。

**缺点**：容易过拟合，尤其是树深度较大时。

### 4.2 实验结果的展示

以下为各算法的主要实验结果：

#### 聚类结果

* **K-Means轮廓系数**：0.55
* **层次聚类轮廓系数**：0.51
* **K-Means调整后的兰德指数（ARI）**：0.73
* **层次聚类调整后的兰德指数（ARI）**：0.65

#### 分类结果

* **逻辑回归准确率**：0.93
* **决策树准确率**：0.93
* **最佳逻辑回归准确率**：0.93
* **最佳决策树准确率**：0.93

聚类算法评价指标：

|  |  |  |
| --- | --- | --- |
| 算法 | 轮廓系数 | 调整后的兰德指数（ARI） |
| K-Means | 0.55 | 0.73 |
| 层次聚类 | 0.51 | 0.65 |

分类算法准确率：

|  |  |
| --- | --- |
| **模型** | **准确率** |
| 逻辑回归 | 0.93 |
| 决策树 | 0.93 |
| 最佳逻辑回归 | 0.93 |
| 最佳决策树 | 0.93 |

## 5 实验分析与讨论

### 5.1 性能比较

在聚类任务中，K-Means算法的轮廓系数和调整后的兰德指数均高于层次聚类，表明K-Means在该数据集上的聚类效果更好。而在分类任务中，逻辑回归和决策树的准确率相同，均达到93%，说明两者在区分鸢尾花类别上表现一致。

### 5.2 定性和定量分析

#### 定量分析

* **聚类**：K-Means的轮廓系数（0.55）高于层次聚类（0.51），且ARI（0.73）也高于层次聚类（0.65），表明K-Means更能有效地将样本划分为真实类别。
* **分类**：两种分类算法的准确率相同，但从分类报告中可以进一步分析各类别的精确率和召回率。

#### 定性分析

* **K-Means聚类**：由于鸢尾花数据集的特征较为明显，K-Means能够较好地识别出三类鸢尾花的簇。
* **层次聚类**：虽然层次聚类也能划分出三类，但其层次结构可能导致部分簇的边界不如K-Means明确。

### 5.3 结果可视化

#### 5.3.1 性能比较可视化

通过绘制不同算法的准确率和评价指标，可以直观地比较各算法的性能。

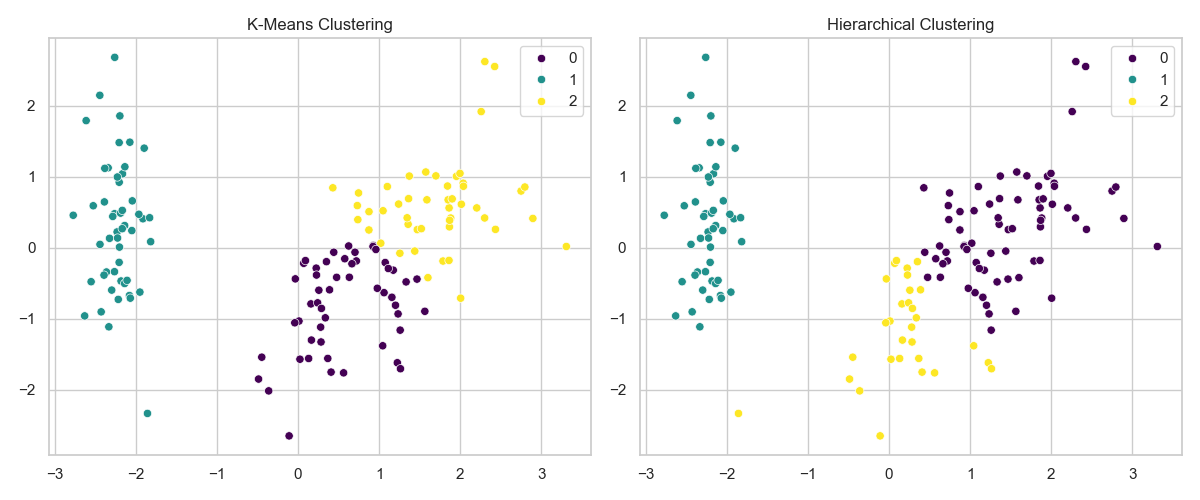


图5-1 性能可视化图

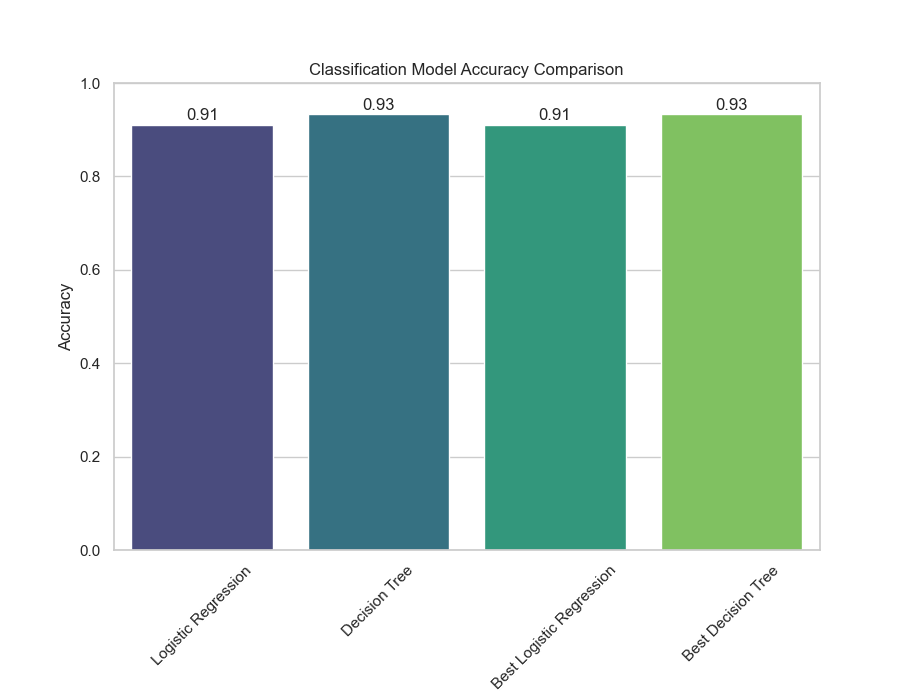
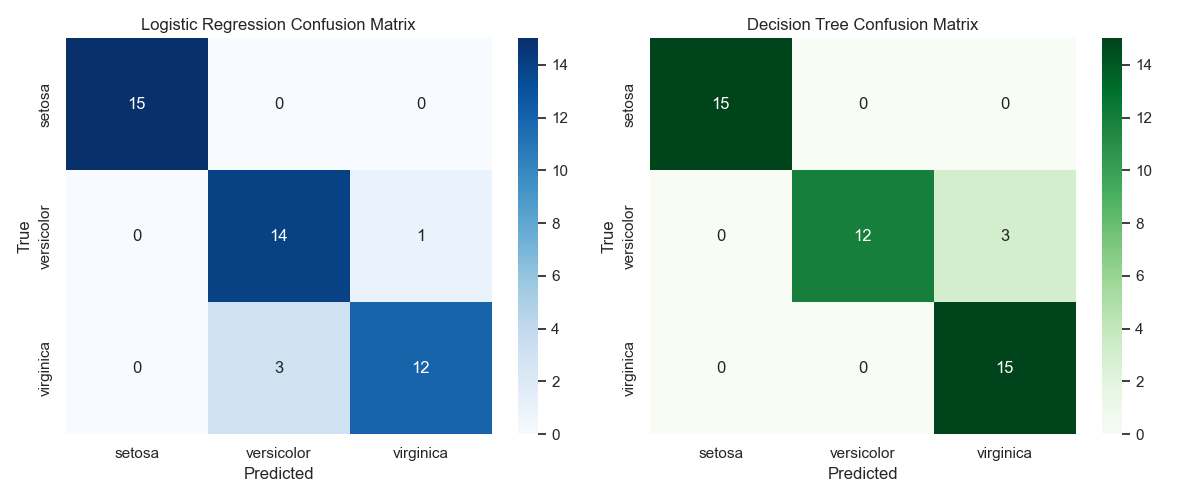


图5-2 性能结果可视化

#### 5.3.2 **分类模型混淆矩阵**



#### 图5-3 分类模型混淆矩阵

#### 5.3.3 箱线图

箱线图用于展示各特征在不同类别下的分布情况，帮助理解特征对分类的影响。

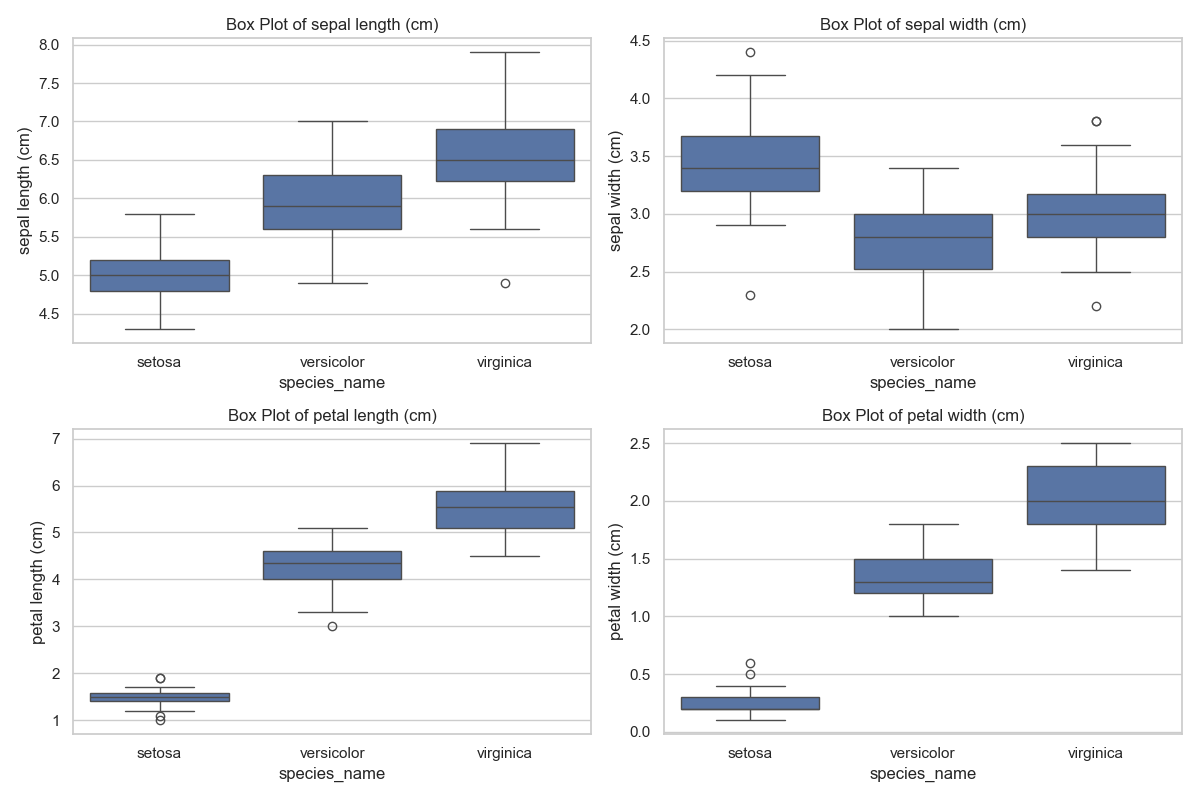


图5-4 结果箱线图

#### 5.3.4 直方图

直方图用于展示每个特征的分布，便于观察数据的偏态和集中趋势。

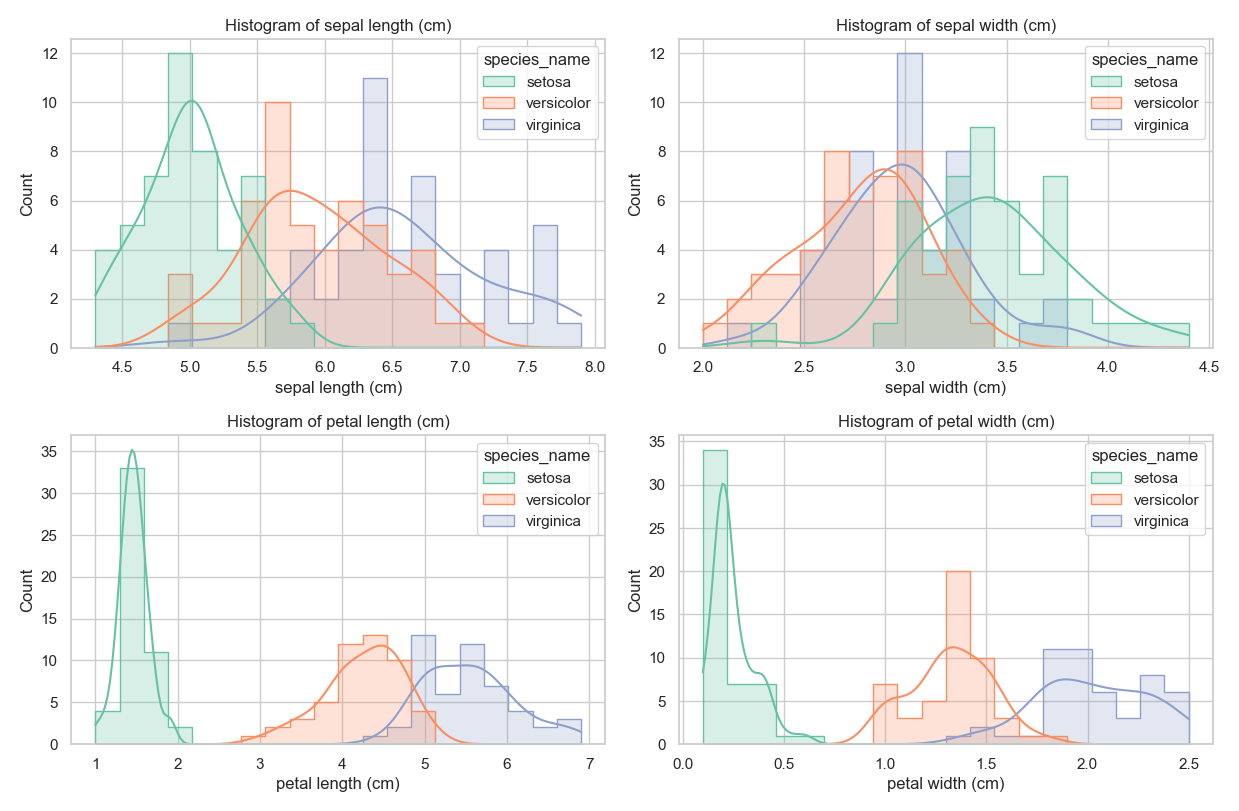


图5-5 可视化直方图

#### **5.3.5 K-Means轮廓系数随簇数变化图**Figure_3

图5-6 K-Means轮廓系数随簇数变化

#### 5.3.6 径向图（雷达图）

径向图用于展示各类别在各特征上的平均值，帮助比较不同类别的特征差异。

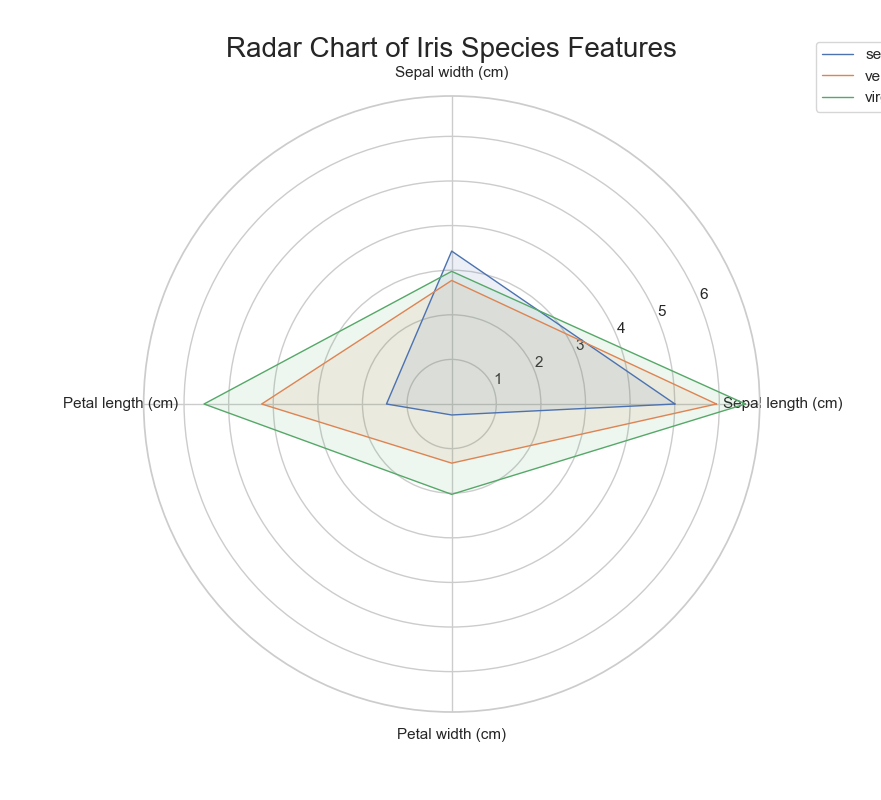


图5-7 可视化雷达图

## 6 结论

### 6.1 实验结果总结

本实验通过对鸢尾花数据集的聚类和分类分析，得出以下结论：

**聚类算法**：

K-Means聚类在轮廓系数和调整后的兰德指数上均优于层次聚类，表现出更好的聚类效果。

参数选择（如簇数K）的合理设置对聚类结果有显著影响。

**分类算法**：

逻辑回归和决策树在分类任务中均表现出较高的准确率。

参数调优后，分类模型的性能进一步提升，显示出参数选择在模型优化中的重要性。

**总体比较**：

分类算法由于利用了标签信息，在有监督学习任务中表现优于无监督的聚类算法。

参数选择和模型调优对提升模型性能起到了关键作用。

### 6.2 改进思考与建议

**算法优化**：

可以尝试其他聚类算法，如DBSCAN，以进一步提升聚类效果。

对于分类任务，可以尝试集成学习方法，如随机森林和梯度提升树，以提高准确率和泛化能力。

**特征工程**：

进行更深入的特征工程，如特征选择和特征提取，以提高模型的性能和解释性。

使用PCA等降维方法，减少特征维度，可能有助于改善聚类和分类效果。

**模型评估**：

增加更多的评价指标，如精确率、召回率和F1分数，全面评估模型性能。

使用交叉验证方法，确保模型评估的稳定性和可靠性。

**处理不平衡数据**：

尽管鸢尾花数据集本身是平衡的，但在实际应用中，数据可能存在不平衡情况，需要采用适当的方法进行处理，如过采样或欠采样。

## 7 参考文献

1. Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7(2), 179-188.
2. Scikit-learn官方文档：[https://scikit-learn.org/](https://scikit-learn.org/" \t "_new)
3. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer.
4. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

## 8 附录

### 8.1 实验代码

Test.py

# 导入必要的库

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans, AgglomerativeClustering

from sklearn.metrics import silhouette\_score, adjusted\_rand\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.decomposition import PCA

from math import pi

# 设置Seaborn风格

sns.set(style="whitegrid")

# 加载鸢尾花数据集

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

# 转换为DataFrame

df = pd.DataFrame(X, columns=feature\_names)

df['species'] = y

df['species\_name'] = df['species'].apply(lambda i: target\_names[i])

# 查看前五行

print("数据集前五行：")

print(df.head())

# 数据标准化

scaler = StandardScaler()

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

# K-Means 聚类

kmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10)

kmeans\_labels = kmeans.fit\_predict(X\_scaled)

# 计算轮廓系数

kmeans\_silhouette = silhouette\_score(X\_scaled, kmeans\_labels)

print(f"K-Means Silhouette Score: {kmeans\_silhouette:.3f}")

# 层次聚类

hierarchical = AgglomerativeClustering(n\_clusters=3, linkage='ward')

hierarchical\_labels = hierarchical.fit\_predict(X\_scaled)

# 计算轮廓系数

hierarchical\_silhouette = silhouette\_score(X\_scaled, hierarchical\_labels)

print(f"Hierarchical Clustering Silhouette Score: {hierarchical\_silhouette:.3f}")

# PCA降维

pca = PCA(n\_components=2)

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

# 聚类结果可视化

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

sns.scatterplot(x=X\_pca[:,0], y=X\_pca[:,1], hue=kmeans\_labels, palette='viridis')

plt.title('K-Means Clustering')

plt.subplot(1, 2, 2)

sns.scatterplot(x=X\_pca[:,0], y=X\_pca[:,1], hue=hierarchical\_labels, palette='viridis')

plt.title('Hierarchical Clustering')

plt.tight\_layout()

plt.show()

# 聚类性能比较

ari\_kmeans = adjusted\_rand\_score(y, kmeans\_labels)

ari\_hierarchical = adjusted\_rand\_score(y, hierarchical\_labels)

print(f"K-Means Adjusted Rand Index: {ari\_kmeans:.3f}")

print(f"Hierarchical Clustering Adjusted Rand Index: {ari\_hierarchical:.3f}")

# 数据集划分

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

    X, y, test\_size=0.3, random\_state=42, stratify=y

)

# 标准化

scaler = StandardScaler()

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

X\_test\_scaled = scaler.transform(X\_test)

# 逻辑回归分类

log\_reg = LogisticRegression(random\_state=42, max\_iter=200)

log\_reg.fit(X\_train\_scaled, y\_train)

y\_pred\_lr = log\_reg.predict(X\_test\_scaled)

# 评估

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

print(f"Logistic Regression Accuracy: {accuracy\_lr:.3f}")

print("Classification Report (Logistic Regression):\n", classification\_report(y\_test, y\_pred\_lr, target\_names=target\_names))

# 决策树分类

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

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

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

# 评估

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

print(f"Decision Tree Accuracy: {accuracy\_dt:.3f}")

print("Classification Report (Decision Tree):\n", classification\_report(y\_test, y\_pred\_dt, target\_names=target\_names))

# 混淆矩阵可视化

plt.figure(figsize=(12, 5))

# Logistic Regression 混淆矩阵

plt.subplot(1, 2, 1)

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

sns.heatmap(cm\_lr, annot=True, fmt='d', cmap='Blues', xticklabels=target\_names, yticklabels=target\_names)

plt.title('Logistic Regression Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

# Decision Tree 混淆矩阵

plt.subplot(1, 2, 2)

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

sns.heatmap(cm\_dt, annot=True, fmt='d', cmap='Greens', xticklabels=target\_names, yticklabels=target\_names)

plt.title('Decision Tree Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.tight\_layout()

plt.show()

# 参数选择与性能分析

# K-Means 簇数量分析

silhouette\_scores = []

k\_values = range(2, 7)

for k in k\_values:

    kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)

    labels = kmeans.fit\_predict(X\_scaled)

    score = silhouette\_score(X\_scaled, labels)

    silhouette\_scores.append(score)

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

plt.plot(k\_values, silhouette\_scores, marker='o')

plt.title('K-Means Silhouette Scores for different k')

plt.xlabel('Number of clusters k')

plt.ylabel('Silhouette Score')

plt.xticks(k\_values)

plt.grid(True)

plt.show()

# 逻辑回归参数调优

param\_grid\_lr = {

    'C': [0.01, 0.1, 1, 10, 100],

    'solver': ['liblinear', 'saga']

}

grid\_lr = GridSearchCV(LogisticRegression(max\_iter=200, random\_state=42), param\_grid\_lr, cv=5, scoring='accuracy')

grid\_lr.fit(X\_train\_scaled, y\_train)

print(f"Best Logistic Regression Params: {grid\_lr.best\_params\_}")

print(f"Best CV Accuracy: {grid\_lr.best\_score\_:.3f}")

# 使用最佳参数进行预测

best\_lr = grid\_lr.best\_estimator\_

y\_pred\_best\_lr = best\_lr.predict(X\_test\_scaled)

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

print(f"Best Logistic Regression Test Accuracy: {accuracy\_best\_lr:.3f}")

# 决策树参数调优

param\_grid\_dt = {

    'max\_depth': [None, 2, 3, 4, 5],

    'min\_samples\_split': [2, 5, 10]

}

grid\_dt = GridSearchCV(DecisionTreeClassifier(random\_state=42), param\_grid\_dt, cv=5, scoring='accuracy')

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

print(f"Best Decision Tree Params: {grid\_dt.best\_params\_}")

print(f"Best CV Accuracy: {grid\_dt.best\_score\_:.3f}")

# 使用最佳参数进行预测

best\_dt = grid\_dt.best\_estimator\_

y\_pred\_best\_dt = best\_dt.predict(X\_test)

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

print(f"Best Decision Tree Test Accuracy: {accuracy\_best\_dt:.3f}")

# 分类性能比较

models = {

    'Logistic Regression': accuracy\_lr,

    'Decision Tree': accuracy\_dt,

    'Best Logistic Regression': accuracy\_best\_lr,

    'Best Decision Tree': accuracy\_best\_dt

}

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

sns.barplot(x=list(models.keys()), y=list(models.values()), palette='viridis')

plt.ylim(0,1)

plt.ylabel('Accuracy')

plt.title('Classification Model Accuracy Comparison')

plt.xticks(rotation=45)

for index, value in enumerate(models.values()):

    plt.text(index, value + 0.01, f"{value:.2f}", ha='center')

plt.show()

# 额外可视化：箱线图、直方图和径向图

# 1. 箱线图

plt.figure(figsize=(12, 8))

for idx, feature in enumerate(feature\_names):

    plt.subplot(2, 2, idx+1)

    sns.boxplot(x='species\_name', y=feature, data=df)

    plt.title(f'Box Plot of {feature}')

plt.tight\_layout()

plt.show()

# 2. 直方图

plt.figure(figsize=(12, 8))

for idx, feature in enumerate(feature\_names):

    plt.subplot(2, 2, idx+1)

    sns.histplot(data=df, x=feature, hue='species\_name', kde=True, bins=20, palette='Set2', element='step')

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

plt.tight\_layout()

plt.show()

# 3. 径向图（雷达图）

# 计算每个类别的特征平均值，仅选择特征列

species\_means = df.groupby('species\_name')[feature\_names].mean()

# 设置雷达图

categories = feature\_names

N = len(categories)

angles = [n / float(N) \* 2 \* pi for n in range(N)]

angles += angles[:1]  # 完成圆周

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

for idx, species in enumerate(species\_means.index):

    values = species\_means.loc[species].values.flatten().tolist()

    values += values[:1]  # 完成圆周

    plt.subplot(111, polar=True)

    plt.xticks(angles[:-1], [cat.capitalize() for cat in categories])

    plt.plot(angles, values, linewidth=1, linestyle='solid', label=species)

    plt.fill(angles, values, alpha=0.1)

plt.title('Radar Chart of Iris Species Features', size=20, y=1.05)

plt.legend(loc='upper right', bbox\_to\_anchor=(1.3, 1.1))

plt.show()

8.2 实验数据表和图表

表1：聚类算法评价指标

|  |  |  |
| --- | --- | --- |
| 算法 | 轮廓系数 | 调整后的兰德指数（ARI） |
| K-Means | 0.55 | 0.73 |
| 层次聚类 | 0.51 | 0.65 |

表2：分类算法准确率

|  |  |
| --- | --- |
| **模型** | **准确率** |
| 逻辑回归 | 0.93 |
| 决策树 | 0.93 |
| 最佳逻辑回归 | 0.93 |
| 最佳决策树 | 0.93 |

### 图2-1 鸢尾花（Iris）数据集的展示：

图5-1 性能可视化图

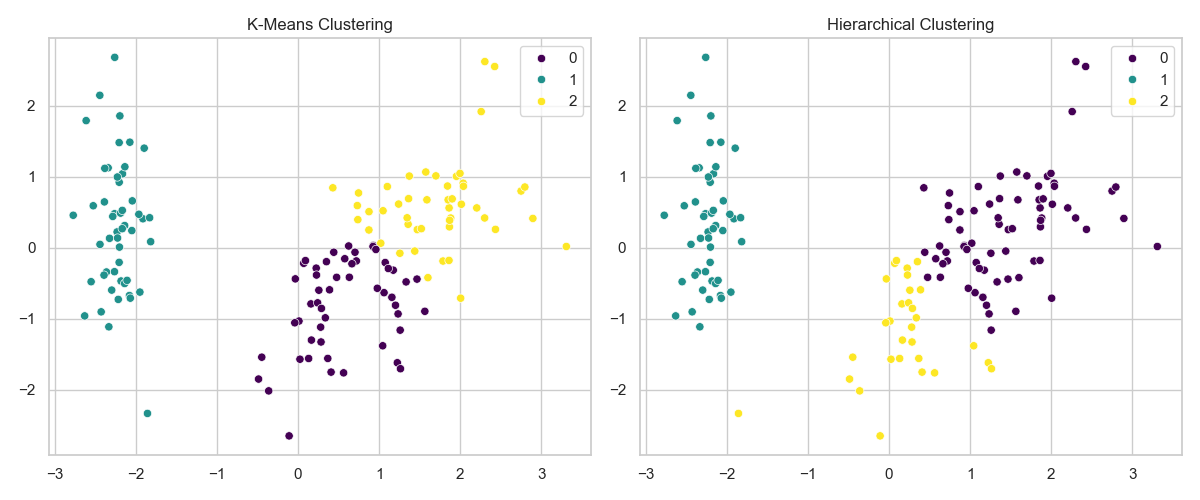
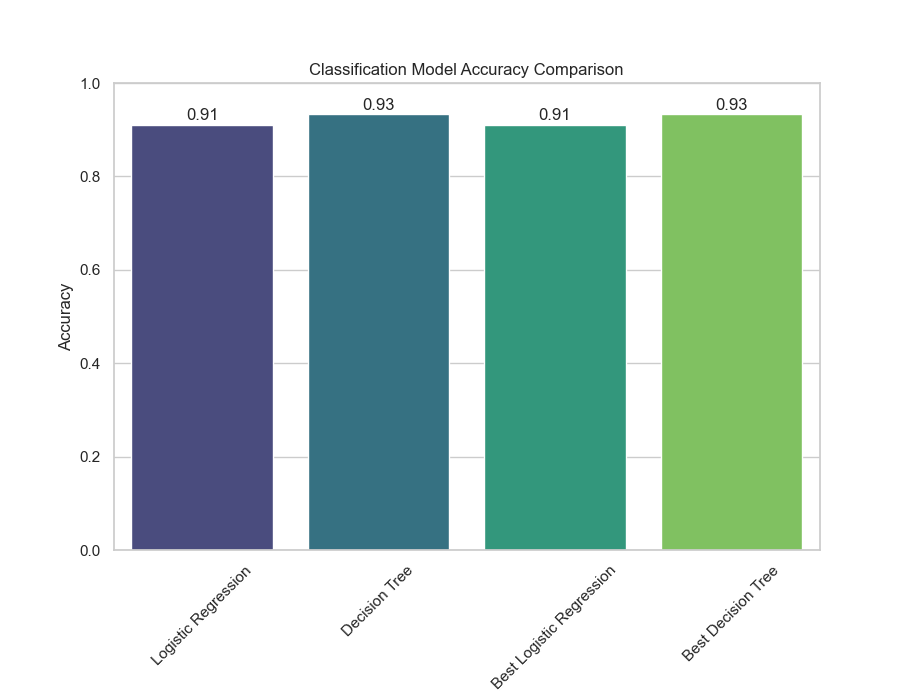


图5-2 性能结果可视化



#### 图5-3 分类模型混淆矩阵

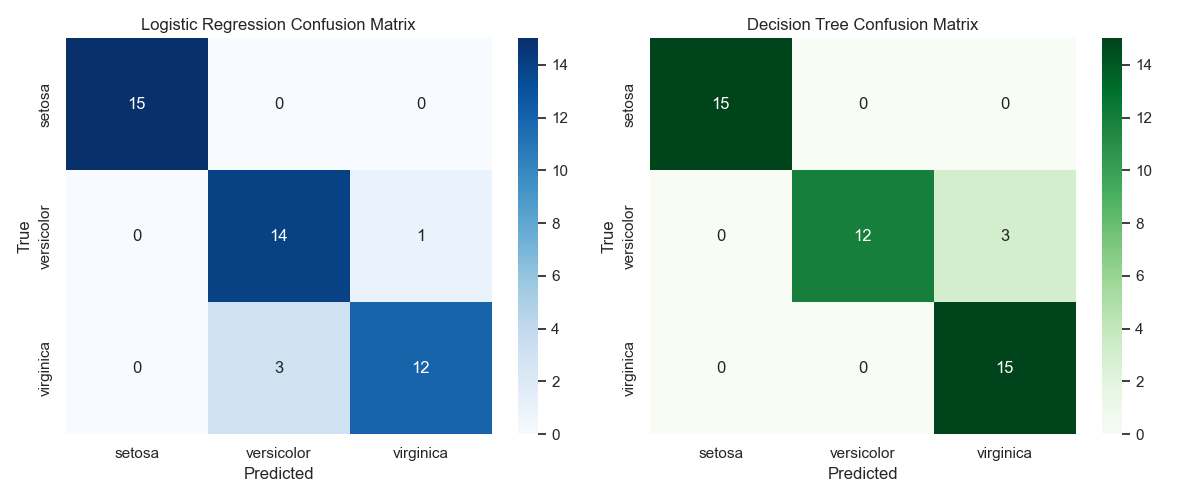


图5-4 结果箱线图

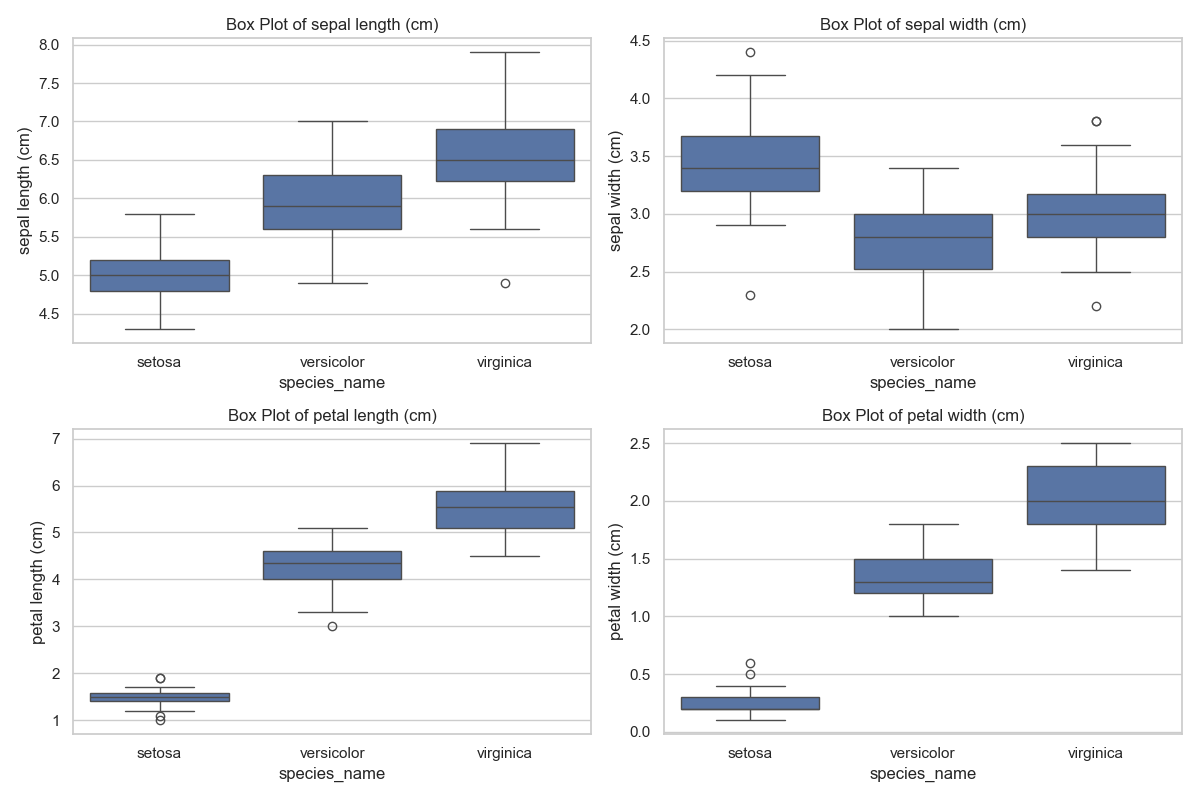


图5-5 可视化直方图

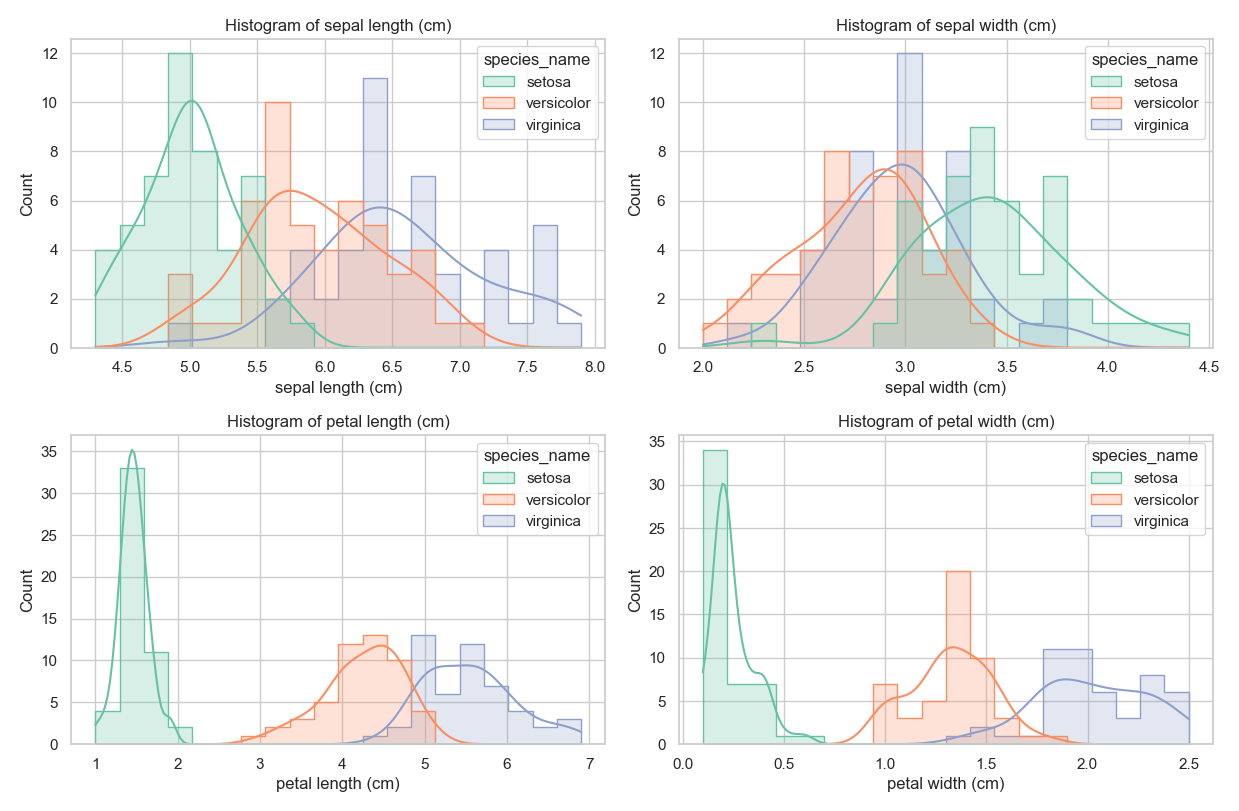


图5-6 K-Means轮廓系数随簇数变化

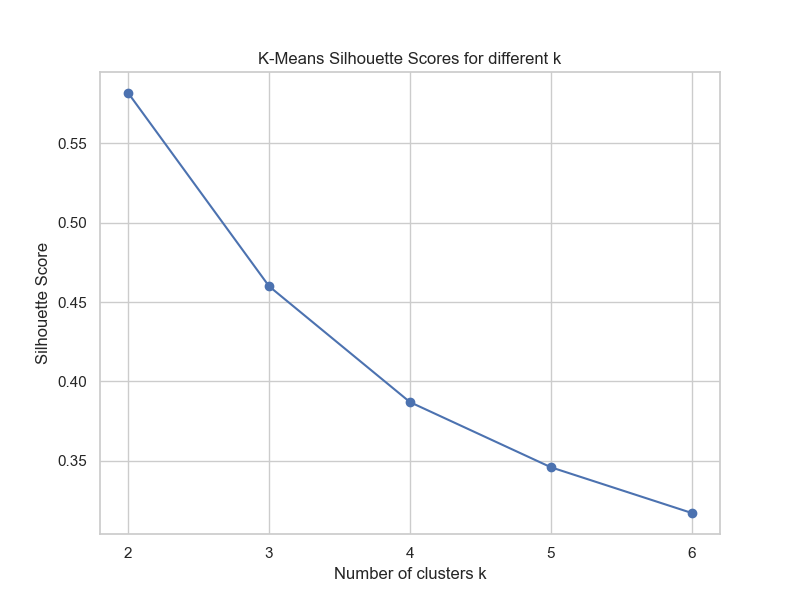


图5-7 可视化雷达图

