基于 K-means 算法的鸢尾花数据集聚类分析

数据挖掘实验报告

2024 年 12 月 18 日

摘要

本实验基于经典的鸢尾花（Iris）数据集，实现并分析了 K-means 聚类算法。通过对算法的实现、参数优化和结果可视化，深入研究了 K-means 算法在实际数据集上的应用效果。实验结果表明 K-means 算法能够有效地对鸢尾花数据进行聚类分析，并取得了良好的聚类效果。

目录

1 引言

1.1 研究背景

聚类分析是数据挖掘和机器学习中的重要任务之一，其目的是将相似的数据对象划分到同一个簇中，而将不相似的对象划分到不同簇中。K-means 算法作为最经典的聚类算法之一，因其简单、高效的特点而被广泛应用。

1.2 研究目的

本实验旨在：

- 实现 K-means 聚类算法
- 分析算法在鸢尾花数据集上的表现
- 通过可视化手段理解聚类结果
- 评估聚类效果并进行参数优化

2 理论基础

2.1 K-means 算法原理

K-means 算法的基本思想是通过迭代方式寻找 K 个簇的一种划分方案，使得聚类结果对应的代价函数最小。其主要步骤如下：

1. 随机选择 K 个点作为初始聚类中心
2. 计算每个数据点到各个聚类中心的距离，将其划分到最近的聚类中心所对应的簇
3. 重新计算每个簇的中心点（计算簇内所有点的均值）
4. 重复步骤 2 和 3，直到聚类中心不再发生变化或达到最大迭代次数

2.2 评估指标

本实验使用以下指标评估聚类效果：

- SSE（簇内误差平方和）：评估簇内的紧密度
- 轮廓系数：评估簇的分离度和紧密度
- 聚类准确率：与真实标签比较的正确率

3 实验设计

3.1 数据集说明

鸢尾花数据集包含 150 个样本，每个样本有 4 个特征：

- 萼片长度（Sepal Length）
- 萼片宽度（Sepal Width）
- 花瓣长度（Petal Length）
- 花瓣宽度（Petal Width）

数据集包含三个品种的鸢尾花：Setosa、Versicolor 和 Virginica，每类 50 个样本。

3.2 实验环境

- 编程语言：Python 3.6+
- 主要库：NumPy, Pandas, Matplotlib, Scikit-learn
- 开发环境：Visual Studio Code

3.3 实验流程

1. 数据预处理
    - 数据加载和清洗
    - 特征标准化
    - 数据可视化分析
2. 算法实现
    - 实现 K-means 核心算法
    - 实现评估指标计算
    - 实现结果可视化
3. 参数优化
    - 使用肘部法则确定最优 K 值
    - 分析不同 K 值对聚类效果的影响

4 实验结果与分析

4.1 数据分布分析

通过特征分布图和散点矩阵，我们可以观察到：

- 不同特征的分布特征
- 特征之间的相关性
- 数据的可分性

4.2 聚类结果分析

4.2.1 最优 K 值选择

通过肘部曲线和轮廓系数分析，我们发现：

- K=3 时，SSE 下降趋势变缓
- K=3 时，轮廓系数达到较高值
- 这与数据集中实际的三个品种数量相符

4.2.2 聚类效果评估

在 K=3 的情况下：

- 聚类准确率达到较高水平
- 簇内紧密度良好
- 簇间分离度明显

4.3 算法性能分析

- 收敛速度快，一般在 10 次迭代内收敛
- 计算复杂度适中，适合中等规模数据集
- 对初始中心点的选择较为敏感

5 结论与展望

5.1 主要结论

- K-means 算法在鸢尾花数据集上表现　好
- 算法能够有效识别数据集中的自然聚类
- 通过参数优化可以进一步提升聚类效果

5.2 改进方向

- 优化初始中心点的选择策略
- 考虑使用其他距离度量方式
- 结合其他算法进行集成学习

6 参考文献

1. MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations.
2. Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems.
3. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

A 代码实现

A.1 K-means 核心实现

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
from data_visualization import load_data

class KMeans:
    def __init__(self, n_clusters=3, max_iters=300):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        self.centroids = None
        self.labels = None

    def fit(self, X):
        # 随机初始化聚类中心
        idx = np.random.choice(len(X), self.n_clusters, replace=False)
        self.centroids = X[idx]

        for _ in range(self.max_iters):
            old_centroids = self.centroids.copy()

            # 计算每个样本到各个聚类中心的距离
            distances = np.sqrt(((X - self.centroids[:, np.newaxis])**2).sum(axis=2))
            self.labels = np.argmin(distances, axis=0)

            # 更新聚类中心
            for k in range(self.n_clusters):
                if sum(self.labels == k) > 0:
                    self.centroids[k] = X[self.labels == k].mean(axis=0)

            # 检查是否收敛
            if np.all(old_centroids == self.centroids):
                break

```

```
34          return self
35
36      def predict(self, X):
37          distances = np.sqrt(((X - self.centroids[:, np.newaxis])**2).
               sum(axis=2))
38          return np.argmin(distances, axis=0)
39
40  def calculate_sse(X, kmeans):
41      """计算SSE（簇内误差平方和）"""
42      sse = 0
43      for k in range(kmeans.n_clusters):
44          cluster_points = X[kmeans.labels == k]
45          centroid = kmeans.centroids[k]
46          sse += np.sum((cluster_points - centroid) ** 2)
47      return sse
48
49  def plot_elbow_curve(X):
50      """绘制肘部曲线"""
51      sse_values = []
52      silhouette_values = []
53      k_range = range(2, 8)
54
55      for k in k_range:
56          kmeans = KMeans(n_clusters=k)
57          kmeans.fit(X)
58          sse = calculate_sse(X, kmeans)
59          silhouette = silhouette_score(X, kmeans.labels)
60          sse_values.append(sse)
61          silhouette_values.append(silhouette)
62
63      # 绘制SSE曲线
64      plt.figure(figsize=(12, 5))
65      plt.subplot(1, 2, 1)
66      plt.plot(k_range, sse_values, 'bo-')
67      plt.xlabel('聚类数量 (k)')
68      plt.ylabel('SSE')
69      plt.title('肘部曲线')
70
71      # 绘制轮廓系数曲线
72      plt.subplot(1, 2, 2)
```

```
    plt.plot(k_range, silhouette_values, 'ro-')
    plt.xlabel('聚类数量 (k)')
    plt.ylabel('轮廓系数')
    plt.title('轮廓系数曲线')

    plt.tight_layout()
    plt.savefig('kmeans_evaluation.png')
    plt.close()

def plot_clusters(X, kmeans, title='K-means聚类结果'):
    """绘制聚类结果"""
    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels, cmap='
        viridis')
    plt.scatter(kmeans.centroids[:, 0], kmeans.centroids[:, 1],
                c='red', marker='x', s=200, linewidths=3, label='聚类
                    中心')
    plt.colorbar(scatter)
    plt.title(title)
    plt.xlabel('Sepal Length')
    plt.ylabel('Petal Length')
    plt.legend()
    plt.savefig('kmeans_clusters.png')
    plt.close()

if __name__ == '__main__':
    # 加载数据
    X_scaled, y, _ = load_data()

    # 绘制肘部曲线和轮廓系数
    plot_elbow_curve(X_scaled)

    # 使用最优k值进行聚类
    kmeans = KMeans(n_clusters=3)
    kmeans.fit(X_scaled)

    # 绘制聚类结果
    plot_clusters(X_scaled, kmeans)

    # 计算聚类准确率
```

```
    correct = sum(kmeans.labels == y)
    accuracy = correct / len(y) * 100
    print(f"聚类准确率: {accuracy:.2f}%")
```

Listing 1: K-means 算法实现

A.2 数据可视化实现

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

def load_data():
    """加载和预处理数据"""
    try:
        # 加载数据，跳过第一行，使用第二行作为列名
        data = pd.read_csv('iris_train_shuzhi.csv', skiprows=[0])
        print("原始数据形状:", data.shape)
        print("原始数据前几行:\n", data.head())

        # 删除第一列（索引列）
        data = data.iloc[:, 1:]

        # 检查是否有缺失值
        if data.isnull().any().any():
            print("警告: 数据中存在缺失值，将进行处理")
            data = data.dropna()

        # 分离特征和标签
        X = data.iloc[:, :-1].astype(float)  # 特征
        y = data.iloc[:, -1].astype(int)     # 标签

        print("\n数据统计信息:")
        print(X.describe())

        return X.values, y.values, X.values
    except Exception as e:
        print(f"数据加载错误: {str(e)}")
```

```
33          raise
34
35  def plot_features_distribution(X, y):
36      """绘制特征分布图"""
37      try:
38          features = ['Sepal Length', 'Sepal Width', 'Petal Length', '
                Petal Width']
39
40          plt.figure(figsize=(15, 10))
41          for i, feature in enumerate(features):
42              plt.subplot(2, 2, i+1)
43              for class_label in np.unique(y):
44                  mask = y == class_label
45                  plt.hist(X[mask, i],
46                           bins=20,
47                           alpha=0.5,
48                           label=f'Class {class_label}',
49                           density=True)
50              plt.title(f'{feature} Distribution')
51              plt.xlabel(feature)
52              plt.ylabel('Density')
53              plt.legend()
54
55          plt.tight_layout()
56          plt.savefig('feature_distribution.png')
57          print("特征分布图数据范围: ", np.ptp(X, axis=0))
58          plt.close()
59      except Exception as e:
60          print(f"特征分布图绘制错误: {str(e)}")
61          plt.close()
62
63  def plot_scatter_matrix(X, y):
64      """绘制特征散点矩阵"""
65      try:
66          plt.figure(figsize=(10, 8))
67
68          # 使用不同的颜色和标记绘制散点图
69          markers = ['o', 's', '^']  # 不同的标记样式
70          for i, class_label in enumerate(np.unique(y)):
71              mask = y == class_label
```

```
72              plt.scatter(X[mask, 0],
73                          X[mask, 2],
74                          alpha=0.6,
75                          marker=markers[i],
76                          s=50,
77                          label=f'Class {class_label}')
78
79          plt.xlabel('Sepal Length')
80          plt.ylabel('Petal Length')
81          plt.title('Sepal Length vs Petal Length')
82          plt.legend()
83          plt.grid(True, alpha=0.3)
84
85          plt.tight_layout()
86          plt.savefig('scatter_matrix.png')
87          print("散点图数据范围: ",
88                "\nSepal Length:", np.ptp(X[:, 0]),
89                "\nPetal Length:", np.ptp(X[:, 2]))
90          plt.close()
91      except Exception as e:
92          print(f"散点矩阵图绘制错误: {str(e)}")
93          plt.close()
94
95  def print_data_summary(X, y):
96      """打印数据摘要信息"""
97      try:
98          print("\n数据集摘要信息: ")
99          print(f"样本数量: {len(X)}")
100         print(f"特征数量: {X.shape[1]}")
101         print(f"类别数量: {len(np.unique(y))}")
102         print("\n各类别样本数量: ")
103         unique_labels, counts = np.unique(y, return_counts=True)
104         for label, count in zip(unique_labels, counts):
105             print(f"类别 {label}: {count}")
106
107         print("\n特征值范围: ")
108         for i, feature in enumerate(['Sepal Length', 'Sepal Width', '
                Petal Length', 'Petal Width']):
109             print(f"{feature}: [{X[:, i].min():.2f}, {X[:, i].max()
                :.2f}]")
```

```
    except Exception as e:
        print(f"数据摘要信息生成错误: {str(e)}")

if __name__ == '__main__':
    try:
        # 设置中文显示
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.rcParams['axes.unicode_minus'] = False

        # 加载数据
        X_scaled, y, X_original = load_data()

        # 打印数据摘要
        print_data_summary(X_original, y)

        # 绘制特征分布图
        plot_features_distribution(X_original, y)
        print("特征分布图已保存为 'feature_distribution.png'")

        # 绘制散点矩阵
        plot_scatter_matrix(X_original, y)
        print("散点矩阵图已保存为 'scatter_matrix.png'")

    except Exception as e:
        print(f"程序执行出错: {str(e)}")
```

Listing 2: 数据可视化实现