# 作业背景与目的

本次小组作业的核心目标是基于 **K 近邻（K-Nearest Neighbors, KNN）算法**，实现对经典机器学习数据集——**鸢尾花（Iris）数据集**的分类任务。通过从零开始手动实现 KNN 模型，深入理解其核心思想与算法流程，同时掌握数据预处理、模型评估与可视化的完整机器学习工程流程。

KNN 算法是一种**基于实例的监督学习算法**，不依赖参数化假设，也不需要显式训练模型，而是通过计算样本间的距离，利用“相似样本具有相似标签”的原则进行分类。该算法的核心思想简单但效果稳定，是入门机器学习的重要算法之一。

本作业的目的包括：

1. **理解算法原理**：掌握 KNN 的基本原理，包括距离度量、K 值选择及多数投票机制；
2. **掌握工程实现**：通过 Python 编程实现完整算法逻辑，理解从原理到代码的转化；
3. **熟悉数据分析流程**：学习如何进行数据加载、特征标准化、数据划分、可视化分析；
4. **进行模型评估与调优**：学会利用准确率、精确率、召回率、F1 值和混淆矩阵对模型进行定量评估；
5. **强化团队协作**：通过分工合作完成数据处理、算法实现、可视化与分析撰写等任务，培养协同开发与结果复现能力。

本次实验不仅是一次算法实现训练，更是一次将理论与实践相结合的工程思维训练过程。

# 数据集分析

## 1. 数据来源与基本信息

本实验使用 sklearn.datasets 提供的内置 **Iris 数据集**。该数据集包含 150 条样本数据，共 4 个特征：

* 花萼长度（sepal length, cm）
* 花萼宽度（sepal width, cm）
* 花瓣长度（petal length, cm）
* 花瓣宽度（petal width, cm）

每条样本被标注为以下三种鸢尾花之一：

* Setosa（山鸢尾）
* Versicolor（变色鸢尾）
* Virginica（维吉尼亚鸢尾）

每类各 50 条样本，类别分布均衡，无缺失值或异常值，数据质量良好。

## 2. 数据探索与统计

在数据探索阶段，通过 Pandas 和 NumPy 输出了特征的均值、标准差、最小值与最大值。可见不同特征间数值尺度不同，例如花瓣长度普遍较大、花萼宽度较小。因此，在模型训练前需要进行**特征标准化**，使所有特征服从相同尺度。

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1数据集预处理运行截图

展示了数据加载、缺失值检查、数据标准化与划分结果。训练集与测试集按 7:3 比例划分，设置随机种子保证实验可复现。标准化后特征均值接近 0、标准差接近 1。

A group of graphs with different colored bars

AI-generated content may be incorrect.

Figure 2特征分布的直方图

展示了四个特征在三种鸢尾花类别中的分布。可以观察到：

* Setosa 类在花瓣长度和宽度上显著小于其他两类；
* Versicolor 与 Virginica 在花瓣特征上有部分重叠；
* 花萼特征（sepal）区分类别的能力较弱。

A screenshot of a screen shot of a screen shot of a screen shot of a screen shot of a screen shot of a screen shot of a screen shot of a screen shot of a screen shot

AI-generated content may be incorrect.

Figure 3特征相关度热力图

显示花瓣长度与花瓣宽度的相关系数约为 0.96，强正相关。这表明花瓣特征在分类任务中贡献较大。

A chart with different colored dots

AI-generated content may be incorrect.

Figure 4花瓣长宽散点图

展示了两个最具区分性的特征（花瓣长度与宽度）的样本分布。可以明显看出：

* Setosa 样本形成独立分群；
* Versicolor 与 Virginica 存在部分重叠；  
  这意味着对这两类样本的区分更依赖于精确的距离计算与合适的 K 值。

# KNN 算法实现细节

A screenshot of a computer

AI-generated content may be incorrect.

KNN 算法的思想是：对于一个待分类样本，计算其与训练集中所有样本的距离，取距离最近的 K 个样本，通过\*\*多数投票（Majority Voting）\*\*确定类别。

## 1. 距离度量

本实验在 MyKNN 类中实现了两种距离计算方式：

* **欧氏距离 (Euclidean Distance)**

适用于连续型特征数据。

* 曼哈顿距离 (Manhattan Distance)

对异常值更鲁棒，计算开销较低。

用户可在初始化 MyKNN(k=5, distance\_type='euclidean') 时指定距离类型。

## 2. 核心逻辑实现

算法流程如下：

1. 存储训练数据（lazy learner，不进行显式训练）；
2. 对每个测试样本，计算与所有训练样本的距离；
3. 取前 K 个最近样本；
4. 统计这些邻居中标签出现次数，进行投票；
5. 若票数相同，则优先选择距离最近的样本类别。

整个逻辑封装在 MyKNN 类中，核心方法包括：

* fit(X\_train, y\_train)：保存训练集；
* predict(X\_test)：逐样本预测；
* calculate\_distance()：计算距离；
* predict\_single()：对单一样本进行预测。

1. class MyKNN:
2. """
3. Custom implementation of K-Nearest Neighbors classifier.
5. This implementation supports multiple distance metrics and follows
6. the standard sklearn-style API with fit() and predict() methods.
7. KNN is a lazy learning algorithm that stores training data and
8. performs classification based on majority voting of k nearest neighbors.
10. Parameters
11. ----------
12. k : int, default=5
13. Number of nearest neighbors to consider for classification.
14. distance\_type : {'euclidean', 'manhattan'}, default='euclidean'
15. Distance metric to use for computing distances between samples.
17. Attributes
18. ----------
19. X\_train : np.ndarray of shape (n\_samples, n\_features)
20. Training feature matrix stored during fit().
21. y\_train : np.ndarray of shape (n\_samples,)
22. Training labels stored during fit().
24. Examples
25. --------
26. >>> knn = MyKNN(k=5, distance\_type="euclidean")
27. >>> knn.fit(X\_train, y\_train)
28. >>> predictions = knn.predict(X\_test)
29. """
31. def \_\_init\_\_(self, k: int = 5, distance\_type: str = "euclidean") -> None:
32. """
33. Initialize the KNN classifier.
35. Parameters
36. ----------
37. k : int, default=5
38. Number of nearest neighbors to consider.
39. distance\_type : {'euclidean', 'manhattan'}, default='euclidean'
40. Distance metric to use.
41. """
42. self.k: int = k
43. self.distance\_type: str = distance\_type
44. self.X\_train: np.ndarray | None = None
45. self.y\_train: np.ndarray | None = None
47. def fit(self, X\_train: np.ndarray, y\_train: np.ndarray) -> "MyKNN":
48. """
49. Fit the KNN model by storing the training data.
51. KNN is a lazy learner - no actual training occurs. The method
52. simply stores the training data for later use during prediction.
54. Parameters
55. ----------
56. X\_train : np.ndarray of shape (n\_samples, n\_features)
57. Training feature matrix.
58. y\_train : np.ndarray of shape (n\_samples,)
59. Training target labels.
61. Returns
62. -------
63. self : MyKNN
64. Returns self for method chaining.
65. """
66. self.X\_train = X\_train
67. self.y\_train = y\_train
68. return self
70. def calculate\_distance(self, x1: np.ndarray, x2: np.ndarray) -> float:
71. """
72. Calculate distance between two samples based on the selected metric.
74. Parameters
75. ----------
76. x1 : np.ndarray of shape (n\_features,)
77. First sample (feature vector).
78. x2 : np.ndarray of shape (n\_features,)
79. Second sample (feature vector).
81. Returns
82. -------
83. distance : float
84. Computed distance value between x1 and x2.
86. Raises
87. ------
88. ValueError
89. If distance\_type is not 'euclidean' or 'manhattan'.
91. Notes
92. -----
93. Euclidean distance: d(x1, x2) = sqrt(sum((x1\_i - x2\_i)^2))
94. Manhattan distance: d(x1, x2) = sum(|x1\_i - x2\_i|)
95. """
96. if self.distance\_type == "euclidean":
97. # Euclidean distance: sqrt(sum((x1\_i - x2\_i)^2))
98. return float(np.sqrt(np.sum((x1 - x2) \*\* 2)))
99. elif self.distance\_type == "manhattan":
100. # Manhattan distance: sum(|x1\_i - x2\_i|)
101. return float(np.sum(np.abs(x1 - x2)))
102. else:
103. raise ValueError(f"Unknown distance type: {self.distance\_type}")
105. def predict\_single(self, x: np.ndarray) -> int:
106. """
107. Predict the class label for a single test sample.
109. Parameters
110. ----------
111. x : np.ndarray of shape (n\_features,)
112. Single test sample (feature vector).
113. np.ndarray
114. Returns
115. -------
116. label : int
117. Predicted class label.
119. Notes
120. -----
121. The prediction process involves:
122. 1. Calculate distances from x to all training samples
123. 2. Sort by distance and select k nearest neighbors
124. 3. Perform majority voting on neighbor labels
125. 4. If tie occurs, return label of the closest neighbor
126. """
127. if self.X\_train is None or self.y\_train is None:
128. raise RuntimeError("You must call fit before predict!")
130. # Calculate distances from test sample to all training samples
131. distances: list[tuple[float, int]] = []
132. for i, train\_sample in enumerate(self.X\_train):
133. dist = self.calculate\_distance(x, train\_sample)
134. distances.append((dist, self.y\_train[i]))
136. # Sort by distance and select k nearest neighbors
137. distances.sort(key=lambda x: x[0])
138. k\_nearest: list[tuple[float, int]] = distances[: self.k]
140. # Extract labels of k nearest neighbors
141. k\_nearest\_labels: list[int] = [label for \_, label in k\_nearest]
143. # Majority voting: select the most common label
144. label\_counts = Counter(k\_nearest\_labels)
145. most\_common: list[tuple[int, int]] = label\_counts.most\_common()
147. # Handle tie-breaking: if multiple labels have same count, choose closest
148. if len(most\_common) > 1 and most\_common[0][1] == most\_common[1][1]:
149. return k\_nearest\_labels[0]  # Return label of closest neighbor
151. return most\_common[0][0]
153. def predict(self, X\_test: np.ndarray) -> np.ndarray:
154. """
155. Predict class labels for all test samples.
157. Parameters
158. ----------
159. X\_test : np.ndarray of shape (n\_test\_samples, n\_features)
160. Test feature matrix.
162. Returns
163. -------
164. predictions : np.ndarray of shape (n\_test\_samples,)
165. Array of predicted labels for each test sample.
166. """
167. predictions: list[int] = []
168. for x in X\_test:
169. pred = self.predict\_single(x)
170. predictions.append(pred)
171. return np.array(predictions)

# 模型评估结果

A screenshot of a computer

AI-generated content may be incorrect.

使用测试集（30% 数据）评估模型性能。主要指标包括：

| **指标** | **含义** | **结果（K=9, 欧氏距离）** |
| --- | --- | --- |
| Accuracy | 准确率 | 0.9556 |
| Precision | 各类别精确率 | Setosa=1.00, Versicolor=0.8824, Virginica=1.00 |
| Recall | 各类别召回率 | Setosa=1.00, Versicolor=1.00, Virginica=0.8667 |
| F1-score | 各类别F1-分数 | Setosa=1.00, Versicolor=0.9375, Virginica=0.9286 |

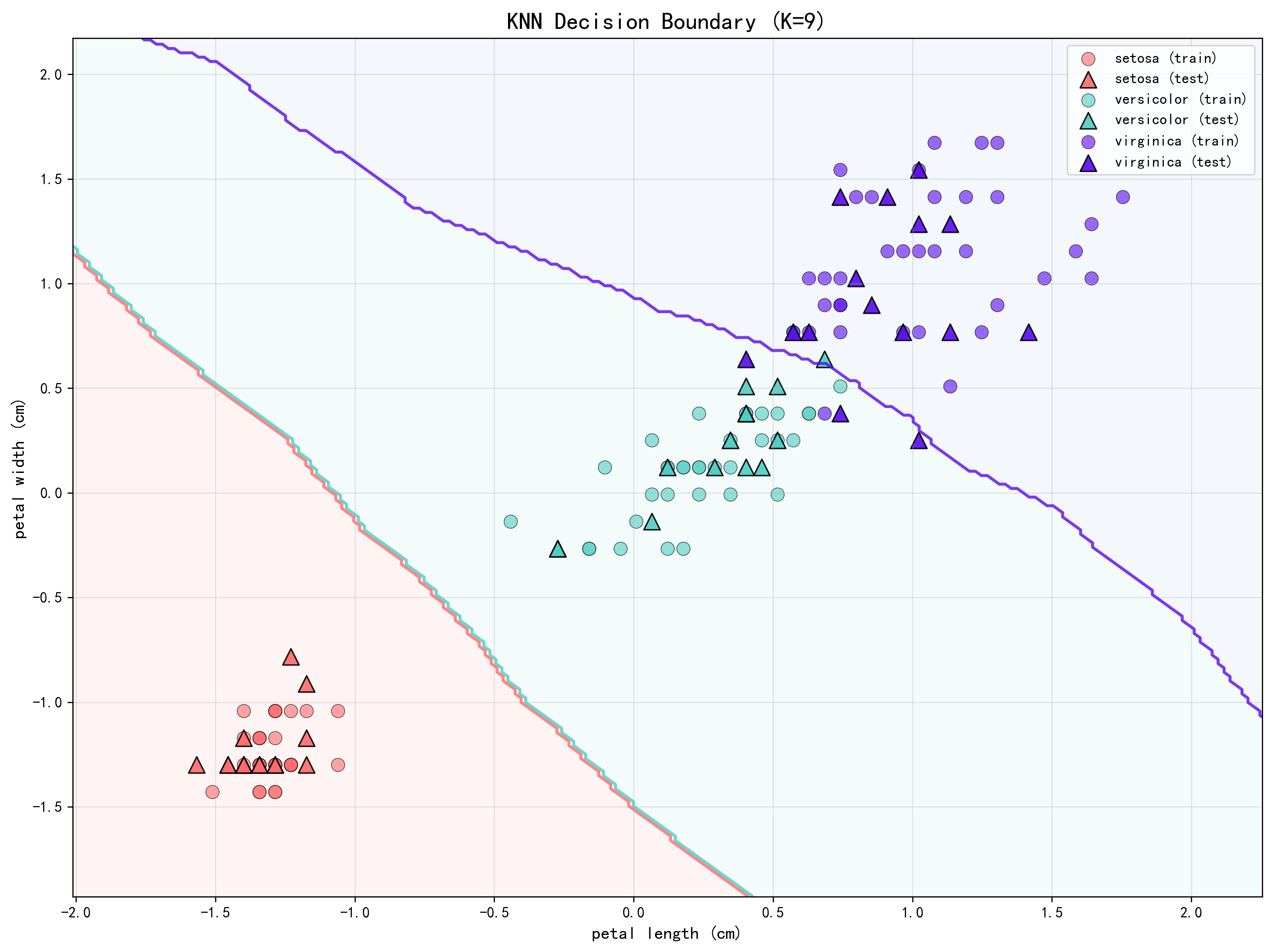


Figure 5 KNN决策边界

该图展示了当 K = 9 时模型在花瓣长度与花瓣宽度特征空间中的分类决策边界。

* 三个颜色区域分别对应三种鸢尾花类别的预测区域：红色（setosa）、青色（versicolor）、紫色（virginica）。
* 圆点表示训练集样本，三角形表示测试集样本。

从图中可以观察到：

* **Setosa** 样本在左下方形成独立聚类，分类边界清晰；
* **Versicolor** 与 **Virginica** 之间的边界较复杂，存在部分重叠区域；
* 决策边界平滑且符合数据分布特征，说明 K=9 时模型具有较好的泛化性，没有明显过拟合。

因此，本实验最终采用 K=9 作为最优参数，兼顾准确率与模型稳定性。

A blue squares with white text

AI-generated content may be incorrect.

Figure 6 混淆矩阵

该图展示了模型在测试集上的分类结果。  
矩阵的横轴为预测类别，纵轴为真实类别。数值表示被分入各类别的样本数量：

| **实际→预测** | **setosa** | **versicolor** | **virginica** |
| --- | --- | --- | --- |
| **setosa** | 15 | 0 | 0 |
| **versicolor** | 0 | 15 | 0 |
| **virginica** | 0 | 2 | 13 |

# 超参数调优分析

A screenshot of a computer

AI-generated content may be incorrect.

A graph with red lines

AI-generated content may be incorrect.

Figure 7 K 值 - 准确率 折线图

该图展示了 K 值变化对模型准确率的影响。  
横轴为 K 值（从 1 到 15），纵轴为模型在验证集上的准确率。  
红色星标标出了最优参数点 K = 9，对应的准确率最高，约为 0.956。

从图中趋势可见：

* 当 K 过小（例如 1、3、5）时，模型对噪声敏感，出现过拟合现象；
* 随着 K 增大，准确率提升并趋于稳定；
* 当 K 大于 11 后，模型略有欠拟合，准确率轻微下降。

综合分析，K=9 时模型取得最佳平衡，能有效区分三类鸢尾花样本。

# 问题与优化建议

欧式距离在这个数据集上面更有效果

A screenshot of a computer

AI-generated content may be incorrect.

从实验输出结果（见上图）可观察到：

* 使用 **曼哈顿距离（Manhattan）** 时，最佳准确率为 **93.33%**，最优 K 值为 **1**；
* 使用 **欧氏距离（Euclidean）** 时，最佳准确率达到 **95.56%**，最优 K 值为 **9**。

这说明在鸢尾花数据集上，**欧氏距离的效果明显优于曼哈顿距离**。  
主要原因是该数据集的四个特征（花萼长、花萼宽、花瓣长、花瓣宽）都是**连续数值特征**，且经过标准化后服从近似高斯分布。此时，欧氏距离能够更准确地反映样本在多维空间中的真实相似性。  
相比之下，曼哈顿距离在这种光滑分布的数据上会放大单维差异，对模型的整体分类边界不够敏感。

因此，在类似连续型特征数据的分类任务中，建议优先采用 **欧氏距离**，并结合中等规模的 K 值（如 K=9）来平衡模型的过拟合与欠拟合风险。

 **距离计算复杂度较高**  
手动实现的 KNN 在预测阶段需计算每个测试样本与所有训练样本的距离，时间复杂度为 O(n×m)。若在更大数据集上使用，需采用 KD-Tree 或 Ball Tree 等结构优化。

 **K 值固定不适应所有类别**  
可考虑自适应 K 值或加权 KNN（距离越近权重越大），提高模型鲁棒性。

 **特征选择优化**  
实验发现花瓣特征贡献较大，若加入 PCA 或特征选择机制，可进一步减少噪声影响。

 **多模态扩展**  
当前模型仅处理结构化数值特征，未来可考虑结合图像或文本嵌入特征，以验证模型在多维特征空间下的表现。

# 小组总结

通过本次鸢尾花 KNN 分类实验，我们小组从数据加载、标准化、算法编程、模型评估到结果可视化，完整体验了一个机器学习项目的全流程。

每位成员在分工中发挥所长：

* 数据工程师负责数据探索与标准化；
* 算法工程师独立实现了 MyKNN 类；
* 可视化工程师设计了特征分布、相关热力图、决策边界图；
* 评估工程师负责模型性能计算与调优分析。

实验结果验证了理论学习的正确性：  
KNN 虽然简单，但在标准化数据上表现稳定，能有效区分线性可分的数据类别。通过本项目，我们不仅深入理解了 KNN 的原理，更掌握了机器学习任务从理论推导到工程实现的完整闭环。  
这为后续学习更复杂模型（如 SVM、决策树、神经网络）奠定了坚实基础。