山东大学 软件 学院

机器学习 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： 202200201095 | 姓名： 杨伟康 | | 班级： 22网安 |
| 实验题目：KNN算法在鸢尾花数据集上的实现 | | | |
| 实验学时：4 | | 实验日期： 11.12 | |
| **1.实验目的：**   * 深入理解和掌握K近邻（K-Nearest Neighbors, KNN）算法的原理、构建过程以及分类机制，提升综合素养； * 利用KNN分类器实现鸢尾花分类，不仅加深对KNN算法的理解，同时对植物学和相关领域的研究提供有力的技术支持和实际应用价值。 | | | |
| **2.引言：**  **背景和动机**  在机器学习和数据科学领域，KNN作为一种简单但有效的分类和回归方法，因其直观易懂、无需复杂的模型训练等特点，受到了广泛的关注和应用。KNN算法通过计算样本之间的距离，选择距离目标样本最近的K个邻居，根据这些邻居的类别进行投票，实现对目标样本的分类。鸢尾花数据集作为机器学习中的经典数据集，因其数据简单、类别明确，常被用作算法演示和性能测试的基准。本文旨在通过实现KNN分类器在鸢尾花数据集上的应用，验证其分类效果，并探讨其在实际应用中的优势和潜力。  **问题描述**  本文要解决的核心问题是：利用KNN算法对鸢尾花进行分类。在Python的PyCharm环境中，利用sklearn库中的iris数据集，该数据集包含150个样本，每个样本具有花萼长度、花萼宽度、花瓣长度、花瓣宽度四个特征属性。目标是基于这四个特征值，将鸢尾花正确地分类为Iris-Setosa、Iris-Versicolour和Iris-Virginica三类。  **文献综述**  KNN算法是一种基于实例的学习方法，或者说是懒惰学习算法，它不会从训练数据中学习判别函数，而是将训练数据存储在内存中，直到需要分类时才进行计算。KNN算法具有简单直观、易于实现、无需复杂的模型训练等优点。在鸢尾花数据集上，KNN算法已被广泛研究和应用，并取得了良好的分类效果。与其他分类算法相比，KNN算法在分类准确率和计算效率上均表现出一定的优势，尤其是在处理小规模数据集时。 | | | |
| **3.实验方法：**  数据集：  Iris鸢尾花数据集（Iris Dataset），包含150个样本，分为三类：Setosa、Versicolor、Virginica。每个样本具有4个特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度。  数据集划分：  将数据集分为训练集（70%）和测试集（30%），或使用交叉验证进行评估。  模型选择：  本实验选择的模型是KNN分类器。KNN分类器通过计算样本之间的距离，选择距离目标样本最近的K个邻居，根据这些邻居的类别进行投票，实现对目标样本的分类。在本实验中，我们将使用Python的scikit-learn库来实现KNN分类器，并利用其提供的参数调整功能来优化模型的性能。  评估指标  为了评估模型的性能，我们将使用准确率（accuracy）、精确率（precision）、召回率（recall）和F1分数（F1-score）等评估指标。准确率是正确分类的样本数与总样本数的比例；精确率表示被正确分类为正例的样本数与所有被预测为正例的样本数之比；召回率表示被正确分类为正例的样本数与所有实际为正例的样本数之比；F1分数是精确率和召回率的调和平均数，用于综合评估模型的性能。在鸢尾花数据集上，这些评估指标能够全面反映模型的分类效果和性能。 | | | |
| **4.实验过程报告：**  **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 from sklearn.preprocessing import StandardScaler from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report, confusion\_matrix  # 加载数据集 iris = load\_iris() X = iris.data y = iris.target  # 数据集划分 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=45)  # 特征缩放 scaler = StandardScaler() X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)  # 自定义KNN算法 class CustomKNN:  def \_\_init\_\_(self, k=1):  self.k = k   def fit(self, X, y):  self.X\_train = X  self.y\_train = y   def predict(self, X):  predictions = [self.\_predict(x) for x in X]  return np.array(predictions)   def \_predict(self, x):  # 计算与所有训练样本的距离  distances = [np.linalg.norm(x - x\_train) for x\_train in self.X\_train]  # 获取距离最近的k个训练样本的索引  k\_indices = np.argsort(distances)[:self.k]  # 获取这些k个训练样本的类别  k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]  # 返回出现次数最多的类别  most\_common = np.bincount(k\_nearest\_labels).argmax()  return most\_common  # 模型选择 k = 3 # 选择K值 knn = CustomKNN(k=k)  # 模型训练 knn.fit(X\_train, y\_train)  # 模型预测 y\_pred = knn.predict(X\_test)  # 评估指标 accuracy = accuracy\_score(y\_test, y\_pred) precision = precision\_score(y\_test, y\_pred, average='weighted') recall = recall\_score(y\_test, y\_pred, average='weighted') f1 = f1\_score(y\_test, y\_pred, average='weighted')  # 输出结果 print(f"Accuracy: {accuracy}") print(f"Precision: {precision}") print(f"Recall: {recall}") print(f"F1 Score: {f1}") print("\nClassification Report:\n", classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))  # 可视化 - 混淆矩阵 conf\_matrix = confusion\_matrix(y\_test, y\_pred) plt.figure(figsize=(8, 6)) sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target\_names, yticklabels=iris.target\_names) plt.xlabel('Predicted') plt.ylabel('True') plt.title('Confusion Matrix') plt.show()  # 可视化 - 分类指标条形图 metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score'] values = [accuracy, precision, recall, f1] plt.figure(figsize=(10, 6)) sns.barplot(x=metrics, y=values, palette='viridis') plt.xlabel('Metrics') plt.ylabel('Value') plt.title('Classification Metrics') plt.show()**  选择不同的随机种子进行验证，当random\_state=45        替换为另外一个种子random\_state=42则会出现完美预测        最后我们尝试把k改成其他值，比如1查看参考的邻居少，是否出现了欠拟合的情况  random\_state=45 K=1结果如下        可以看到相比于k=3，k=1精确度，准确率和召回度等都有略微下降  由于数据量较小，结果下降不明显，在数据量较大时易出现欠拟合的现象 | | | |
| **5. 实验结果与分析**  通过实验，我发现KNN算法在鸢尾花数据集上表现出了良好的分类性能。当K值选择恰当时（如K=3或K=5），算法的准确率可以达到较高水平（接近或超过95%）。然而，K值的选择对算法的性能有较大影响。当K值过小时（如K=1），算法可能会受到噪声数据的影响，导致过拟合；而当K值过大时（如K=9），算法可能会忽略掉一些重要的局部信息，导致欠拟合。  此外，我还发现KNN算法的计算复杂度较高，特别是在处理大规模数据集时。因为对于每个新的实例，都需要计算它与训练集中所有实例的距离，这会导致算法的运行时间随着训练集规模的增加而显著增加。  **6. 结论**  在本次实验中，我采用了K近邻（K-Nearest Neighbors, KNN）算法对鸢尾花数据集进行了分类，并得出了一系列有意义的实验结果。KNN算法是一种基于实例的学习方法，或者说是懒惰学习，它并不从训练数据中学习一个判别函数，而是简单地将新的实例与训练集中的K个最近邻实例进行比较，并根据这些近邻的类别来进行分类。  通过本次实验，我对KNN算法有了更深入的理解。KNN算法虽然简单直观，但在实际应用中却非常有效。它不需要对数据进行复杂的预处理，也不需要构建复杂的模型结构，只需要根据距离度量方法找到最近的K个邻居即可进行分类。这种方法的优点是易于理解和实现，同时也具有很好的灵活性和适应性。  然而，KNN算法也存在一些局限性。首先，它对于大规模数据集的计算复杂度较高，这限制了它在某些应用场景中的使用。其次，K值的选择对算法的性能有较大影响，而如何选择合适的K值并没有一个固定的标准，这需要根据具体的数据集和任务来进行调整。最后，KNN算法对于数据的分布和特征的选择也比较敏感，如果数据存在不平衡或特征之间存在冗余信息，可能会影响算法的分类效果。 | | | |

1. 当次实验结束后一周内按班组织上交实验报告。

2. 实验报告文件命名为：“学号+姓名”，格式为WORD文档。