



K近邻




同濟大學
TONGJI UNIVERSITY

王睿智

ruizhiwang@tongji.edu.cn

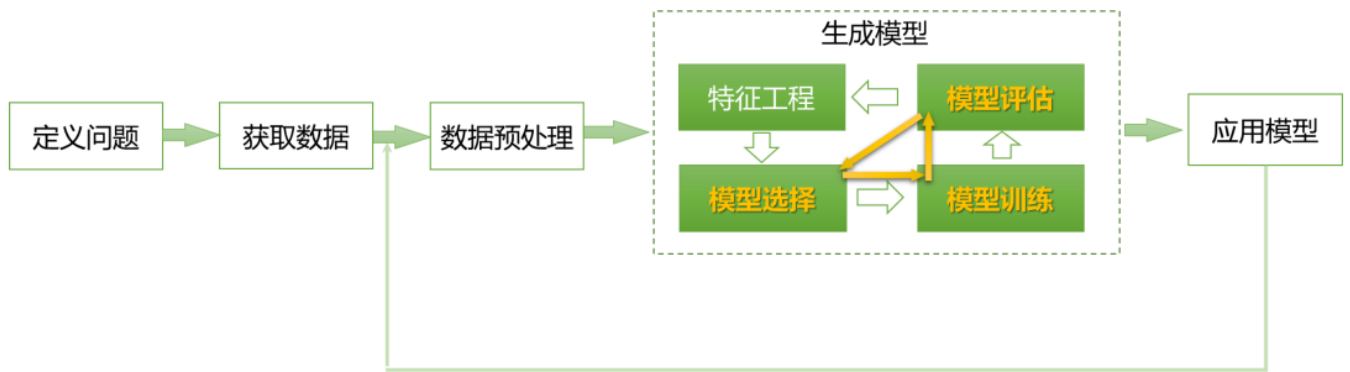
人工智能技术与应用

 第三章 监督学习-3.1 KNN.pdf

全篇大纲

- 引例：鸢尾花分类
 - 问题定义：使用鸢尾花特征识别种类。

- 数据获取：UCI iris 数据集。
- 数据集划分：训练集和测试集。
- 模型选择：K近邻法 (kNN)。
- **k近邻算法**
 - 基本思想：新数据点的类别由其k个最近邻的类别决定。
 - 影响因素：
 - k值：近邻个数，需人为指定，通常为奇数。
 - 距离计算：L1 (Manhattan) 距离、L2 (Euclidean) 距离等。
- **分类性能评估**
 - 模型评估：通过测试集评估模型泛化能力。
 - 超参数选择：通过模型验证选择最佳参数。
 - 模型验证方法：
 - 留出法：从训练集中留出验证集。
 - 交叉验证法：将训练集划分为多个子集进行验证。
 - 评估指标：
 - 准确率：正确预测样本比例。
 - 精确率：预测为正类的样本中正确预测的比例。
 - 召回率：真实为正类的样本中被正确预测出来的比例。
 - F1分数：精度和召回率的调和均值。
- **利用Scikit-learn**
 - 数据加载：使用 `load_iris()` 或 `fetch_openml()` 加载数据。
 - 数据集划分：使用 `train_test_split()` 划分数据集。
 - kNN 分类器：使用 `KNeighborsClassifier` 创建和训练 kNN 模型。
 - 交叉验证：使用 `cross_val_score()` 进行交叉验证。
 - 评估指标：使用 `metrics` 类中的方法评估模型性能。



监督学习，无监督学习与深度学习



监督学习 是一种机器学习方法，它从已标记的数据中学习，并预测新数据的标签。它主要分为两类任务：

- **分类**：预测数据的类别标签，例如鸢尾花分类、垃圾邮件识别。
- **回归**：预测数据的连续值，例如房价预测、温度预测。



无监督学习 是另一种机器学习方法，它从未标记的数据中学习，并发现数据中的模式和结构。它主要分为两类任务：

- **聚类**：将数据划分为不同的簇，例如客户细分、文本聚类。
- **降维**：将数据从高维空间映射到低维空间，例如主成分分析 (PCA)。

区别：

- **数据**：监督学习使用已标记的数据，无监督学习使用未标记的数据。
- **目标**：监督学习预测标签，无监督学习发现模式或结构。



深度学习 是一种特殊的监督学习方法，它使用多层神经网络来学习数据特征。它主要应用于图像识别、语音识别等复杂任务。

和普通监督学习区别：

- **模型复杂度**：深度学习模型比传统的监督学习模型更复杂，需要更多的数据和计算资源。
- **学习效果**：深度学习模型在复杂任务上通常比传统监督学习模型表现更好。
- **可解释性**：深度学习模型的可解释性较差，难以理解其内部工作机制。



Zero-shot learning (ZSL) 是一种特殊的监督学习方法，它能够在训练数据中从未出现过的类别上进行分类。它通常与**无监督学习方法**结合，例如：

- **多模态 Zero-shot Learning:** 利用不同模态的数据（例如文本、图像）进行 ZSL，例如使用文本描述来识别未见过的图像类别。
- **生成模型:** 使用生成模型（例如生成对抗网络）生成未见过的类别的样本，并将其用于训练 ZSL 模型。

数据读取与处理

首先，我们需要从 UCI 机器学习库中下载 iris 数据集。可以使用 Python 的 `pandas` 库来读取数据集并将其存储为 `DataFrame` 对象。

```
1 import pandas as pd
2
3 # 读取数据集
4 url = "https://archive.ics.uci.edu/ml/machine-learning-
   databases/iris/iris.data"
5 column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
   'species']
6 iris = pd.read_csv(url, header=None, names=column_names)
7
8 # 查看数据集信息
9 iris.head()
10
11 data = iris.values #返回numpy数组
12 num_test = 30 #30个样本作测试
13 np.random.seed(3)
14 np.random.shuffle(data) #打乱数据
```

由于鸢尾花种类是类别型数据，我们需要将其转换为数值型数据。可以使用 `pandas` 的 `LabelEncoder` 类来实现。

```
1 from sklearn.preprocessing import LabelEncoder
2
3 # 初始化 LabelEncoder
4 label_encoder = LabelEncoder()
5
6 # 将种类转换为数值型
7 iris['species_encoded'] = label_encoder.fit_transform(iris['species'])
```



```
8
9 # 删除原始种类列
10 iris.drop('species', axis=1, inplace=True)
```

接下来我们对数据集进行划分，以划分成训练集和测试集。使用 `sklearn.model_selection` 的 `train_test_split` 函数可以将数据集划分为训练集和测试集。默认情况下，它会使用随机采样。

```
1 from sklearn.model_selection import train_test_split
2
3 # 划分数据集，80% 为训练集，20% 为测试集
4 X_train, X_test, y_train, y_test =
    train_test_split(iris.drop('species_encoded', axis=1)
5 , iris['species_encoded'], test_size=0.2, random_state=42)
```

如果数据集中类别不平衡，可以使用分层采样来确保训练集和测试集中各类别的比例与原始数据集相同。

```
1 # 划分数据集，80% 为训练集，20% 为测试集，分层采样
2 X_train, X_test, y_train, y_test =
    train_test_split(iris.drop('species_encoded', axis=1)
3 , iris['species_encoded'], test_size=0.2, stratify=iris['species_encoded'],
    random_state=42)
```



随机采样:

- **过程:** 从整个数据集中随机抽取一定比例的样本作为测试集，其余样本作为训练集。
- **特点:**
 - 简单易行，计算成本低。
 - 可能导致训练集和测试集中各类别的比例与原始数据集不一致，特别是当数据集中类别不平衡时。
 - 适用于类别分布均匀的数据集。



分层采样:

- **过程:**
 - 将数据集按照类别划分成多个子集（层）。

- 从每个子集中随机抽取一定比例的样本作为测试集，其余样本作为训练集。
- 确保训练集和测试集中各类别的比例与原始数据集相同。
- **特点:**
 - 能够保证训练集和测试集中各类别的比例与原始数据集一致，从而更好地评估模型在所有类别上的泛化能力。
 - 计算成本比随机采样略高。
 - 适用于类别分布不均匀的数据集，例如医疗诊断数据集。

选择哪种方法:

- 如果数据集中类别分布均匀，可以使用随机采样。
- 如果数据集中类别分布不均匀，建议使用分层采样，以确保模型在所有类别上的性能都能得到有效评估。

注意:

- 无论使用哪种方法，都建议使用 `random_state` 参数设置随机数种子，以确保结果的可重复性。

KNN算法模型

K近邻（K-Nearest Neighbors, KNN）算法是一种基本的分类与回归方法。以下是K近邻算法的思想、原理、推导以及影响因素的详细描述：

基本思想

K近邻算法的核心思想是，对于一个未知类别的新数据点，通过在训练集中找到与它最邻近的k个数据点，根据这k个近邻的类别来判断新数据点的类别。简单来说，一个新数据点的类别由其周围最近的若干个点的多数类别所决定。

原理

1. **训练阶段：**KNN算法在训练阶段主要是存储所有的训练数据及其标签。
2. **预测阶段：**当需要预测一个新数据点的类别时，算法会在训练集中搜索与新数据点最邻近的k个数据点。

1. **距离计算**：对于一个新的数据点 x ，需要计算它与训练集中每个数据点的距离。常用的距离计算方法有：

- **L1距离 (Manhattan距离)**：各个维度上的绝对值之差的总和。

$$L1(x, y) = \sum_{i=1}^n |x_i - y_i|$$

- **L2距离 (Euclidean距离)**：各个维度上差的平方和的平方根。

$$L2(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

其中， x 和 y 是两个数据点， n 是数据点的维度。

2. **选择k值**：计算完所有距离后，选择距离最近的 k 个点。这通常是通过排序距离然后选择前 k 个来实现。

3. **投票机制**：在选出的 k 个近邻中，统计各个类别的出现频率，新数据点被分配到频率最高的类别。

影响因素

1. k值：

- **选择**：k值的选择对KNN算法的性能有很大影响。k值太小，模型对噪声敏感；k值太大，模型可能无法捕捉到数据的局部特征。
- **通常为奇数**：选择奇数可以避免在投票时出现平局的情况。

2. 距离计算：

- **距离度量**：不同的距离度量会影响近邻的选择，进而影响分类结果。
- **尺度问题**：不同特征的尺度可能不同，这可能导致某些特征在距离计算中占主导地位。因此，在实际应用中，通常需要对特征进行标准化处理。

3. **数据预处理**：在实际应用中，对数据进行适当的预处理（如标准化、归一化）是非常重要的，因为它可以确保每个特征在距离计算中都被公平对待。

评估指标

混淆矩阵(Confusion Matrix) 评估分类性能的更好方法

总体思路是统计A类别实例被分成B类别的次数。

独立地考察模型对某个类别（标签）分类能力。将被考察的类视为正类(Positive)，其他为负类(Negative)，混淆矩阵表示某个标签在验证/测试集中真实数量与预测数量的关系。

预测 \ 真实	预测为正类	预测为负类
真实是正类	TP	FN
真实是负类	FP	TN

TP：是真正类的样本数。
FP：是假正类的样本数。
TN：是真负类的样本数。
FN：是假负类的样本数。

- 1. TP（True Positive）：真正例，即实际为正类别且被预测为正类别的样本数。
- 2. FP（False Positive）：假正例，即实际为负类别但被预测为正类别的样本数。
- 3. FN（False Negative）：假负例，即实际为正类别但被预测为负类别的样本数。
- 4. TN（True Negative）：真负例，即实际为负类别且被预测为负类别的样本数。

应用场景：

- 多分类问题：对于多个类别的分类任务，混淆矩阵可以扩展以包括更多的行和列，分别表示不同的类别。
- 不平衡数据集：当数据集中各类别样本的数量不均衡时，混淆矩阵能够帮助识别哪些类别容易被误判或不敏感。

基于混淆矩阵可计算：准确率(accuracy)、精确率(precision)、召回率(recall)、F1分数(F1score)。

- 准确率： $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ 所有样本中被正确预测的比例。 整体准确率
- 精确率： $Precision = \frac{TP}{TP+FP}$ 预测为正类的样本中正确预测的比例，也称精度。
- 召回率： $Recall = \frac{TP}{TP+FN}$ 真实为正类的样本中被正确预测出来的比例，也称灵敏度。
- F1分数： $F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2TP}{2TP+FP+FN}$ 是精度和召回率的调和均值。

关注被考察类别

- ✓ 一个好的分类器，要具有良好的精度和召回率。
- ✓ F1score是精度和召回率的合成指标，是比较两个或多个分类器的直接简单方法。
- ✓ 只有当召回率和精度都很高时，分类器才能得到较高的F1分数。
- ✓ 有时我们更关心精度，另一些情况则更注重召回率，要权衡。
- 准确率（Accuracy）：所有正确分类的比例。
- 召回率（Recall） / 灵敏度（Sensitivity）：真正例占有所有实际为正类别的比例。
- 精确度（Precision）：真正例占有所有预测为正类别的比例。

- F1分数：召回率和精确度的调和平均数。



近邻分类法训练快还是预测快？

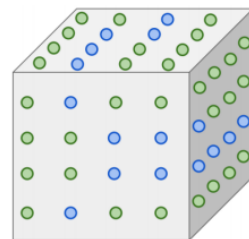
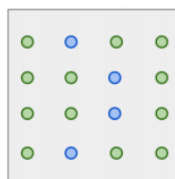
❑ 预测效率低

- 训练时，只需记住样本，快；
- 预测时，需计算待测样本与每个训练样本的距离，若有N个训练样本，就需N次距离计算，慢。

我们希望分类器在**预测时快**；训练时为获得好模型，慢些可以！如图像分类等高维数据分类问题，一般不用kNN。

❑ 对高维、稀疏数据集预测效果差

- 维数灾难



在机器学习和数据分析领域，“维数灾难”通常指的是随着数据特征（即维度）的增加，算法的性能会下降的现象。具体来说，当数据的维度增加时，会出现以下几个问题：

1. **样本稀疏性**：在高维空间中，即使是非常大量的数据点也会显得非常稀疏。这意味着数据点之间的距离可能会变得非常大，导致难以找到有效的分类或回归边界。
2. **计算复杂度**：高维空间的计算量通常会指数级增长。例如，对于k近邻(kNN)算法，其时间复杂度为 $O(Nd)$ ，其中N是样本的数量，d是特征的维度。因此，随着维度的增加，计算成本会迅速上升。
3. **过拟合风险**：在高维空间中，模型更容易捕捉到噪声而不是真正的数据模式，这增加了过拟合的风险。过拟合是指模型在训练数据上表现很好，但在未见过的测试数据上的性能却很差。
4. **参数数量**：模型的参数数量与维度直接相关。高维意味着更多的参数需要估计，这不仅增加了计算的复杂性，还可能导致模型不稳定。
5. **可解释性问题**：高维模型往往更难理解和解释，因为它们可能包含了大量相互作用的变量。

为了应对维数灾难，研究者们提出了多种降维技术，如主成分分析(PCA)、线性判别分析(LDA)、t-SNE等，以及正则化方法来限制模型的复杂度。这些技术的目的是减少数据的维度，同时保留足够的信息以进行有效的学习。

实操代码

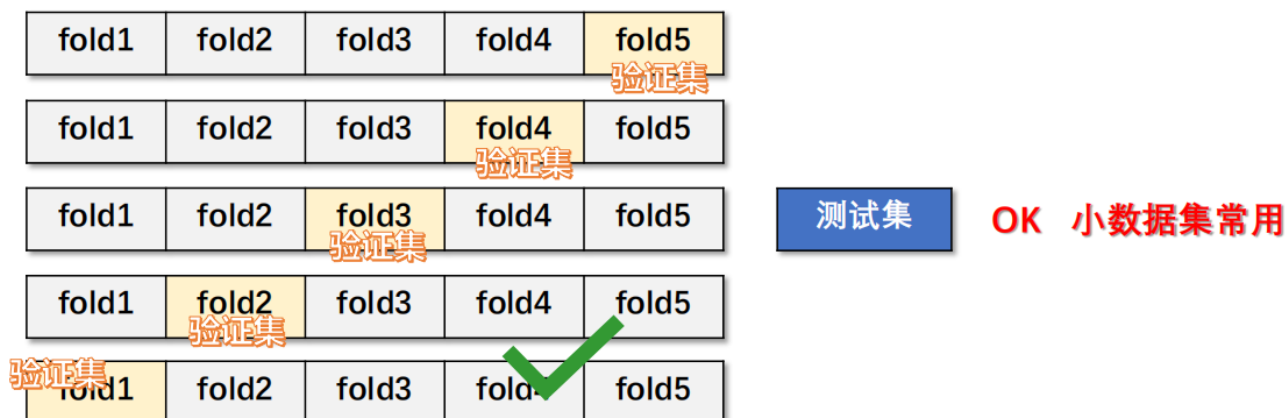
```
1 from sklearn import datasets
2 from sklearn.model_selection import train_test_split, cross_val_score
3 from sklearn.neighbors import KNeighborsClassifier
4 from sklearn.metrics import classification_report, confusion_matrix
```

```
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 # 加载鸢尾花数据集
9 iris = datasets.load_iris()
10 X = iris.data
11 y = iris.target
12
13 # 划分数据集为训练集和测试集
14 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
15
16 # 初始化k值的范围和准确率列表
17 k_range = list(range(1, 40))
18 accuracies = []
19
20 # 对每个k值进行交叉验证
21 for k in k_range:
22     knn = KNeighborsClassifier(n_neighbors=k)
23     scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
24     accuracies.append(scores.mean())
25
26 # 找到准确率最高的k值
27 optimal_k = k_range[np.argmax(accuracies)]
```

交叉验证是一种常用的机器学习技术，主要用于评估算法的性能。它可以估计不同k值下模型的泛化能力，避免过拟合问题。它的基本思想是将原始数据集分割为两个部分：训练集和验证集。然后，使用训练好的模型对验证集进行测试，计算模型的表现。具体来说，交叉验证的过程如下：

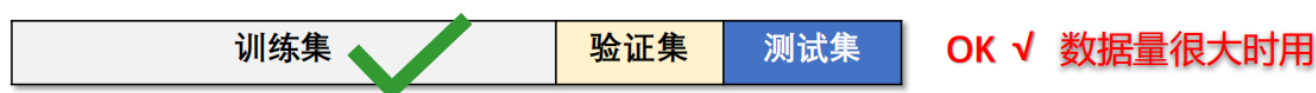
1. 将原始数据集分为训练集和验证集：通常按照一定比例分配样本量。
2. 使用训练集训练模型，并用验证集进行测试。
3. 计算模型在验证集上的表现，并与训练集进行比较。通过比较模型在训练集和验证集上的表现，可以得到模型的效果。

交叉验证法 (Cross-Validation) : **测试集**不动的情况下，将训练数据拆分为多个子集（多折），尝试每折作为验证数据。每个模型都在其余数据上进行训练，在每个**验证集**上评估一次，并平均评估结果。通过对模型的所有评估求平均值，可更准确地衡量模型的性能。**缺点**是训练时间是验证集个数的倍数。



留出法: **测试集**不动的情况下，从**训练集**中留出一部分，以评估各候选模型并选择最佳模型（或最优超参数），新留出集称为**验证集**。

具体做法：首先在简化的训练集（即完整训练集减去验证集）上训练具有不同超参数的多个模型，选择在验证集上表现最佳的模型。在此验证之后，再用完整的训练集（包括验证集）训练最佳模型。最后，在测试集上评估这个模型，以获得泛化误差的估计值。



缺点: 在数据量小的情况下，若留出的验证集太小，则模型验证阶段的评估就将不精确，可能得到次优模型；若留出的验证集太大，则剩余训练集将比完整的训练集小得多，也不好。

留出法 (Hold-out method) :

- **数据划分:** 将数据集划分为两个不相交的部分，通常是较大的部分作为训练集（例如70%），较小的部分作为测试集（例如30%）。
- **模型训练和评估:** 使用训练集来训练模型，然后在测试集上评估模型的性能。
- **评估次数:** 通常只进行一次评估，因此评估结果可能依赖于特定的数据划分方式，具有较大的方差。
- **优点:** 简单、快速。

- **缺点：**如果数据集较小，或者数据划分不够随机，则评估结果可能不具有代表性；模型训练过程中未能充分利用所有数据。

交叉验证 (Cross-validation) :

- **数据划分：**将数据集划分为k个大小相似的互斥子集。最常见的是k=10，即10折交叉验证。
- **模型训练和评估：**每次将k-1个子集合并作为训练集，剩下的一个子集作为验证集。这个过程重复k次，每次使用不同的子集作为验证集，最终得到k个模型性能评估结果。
- **评估次数：**进行k次评估，然后计算这些评估结果的平均值，从而减少方差。
- **优点：**更好地利用了数据，减少了评估结果的方差，通常能够提供更可靠的模型性能估计。
- **缺点：**计算成本较高，尤其是对于大型数据集和复杂模型。

总结：

- 留出法简单快速，但可能不够稳定和准确。
- 交叉验证更复杂，计算成本更高，但通常能提供更可靠的模型性能估计。

```
1 # 使用最优k值重新训练分类器
2 knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)
3 knn_optimal.fit(X_train, y_train)
4
5 # 测试最优分类器的准确率
6 test_accuracy = knn_optimal.score(X_test, y_test)
7
8 # 预测新数据点(5, 2.9, 1.0, 0.2)的类别
9 new_data = np.array([[5, 2.9, 1.0, 0.2]])
10 predicted_class = knn_optimal.predict(new_data)
11
12 # 画出k值与分类准确率之间关系的线形图
13 plt.figure(figsize=(10, 6))
14 plt.plot(k_range, accuracies, marker='o')
15 plt.title('k vs. Classification Accuracy')
16 plt.xlabel('Number of Neighbors (k)')
17 plt.ylabel('Accuracy')
18 plt.show()
19
20 # 输出分类器在测试集上的性能评估报告和混淆矩阵
21 y_pred = knn_optimal.predict(X_test)
22 report = classification_report(y_test, y_pred)
```

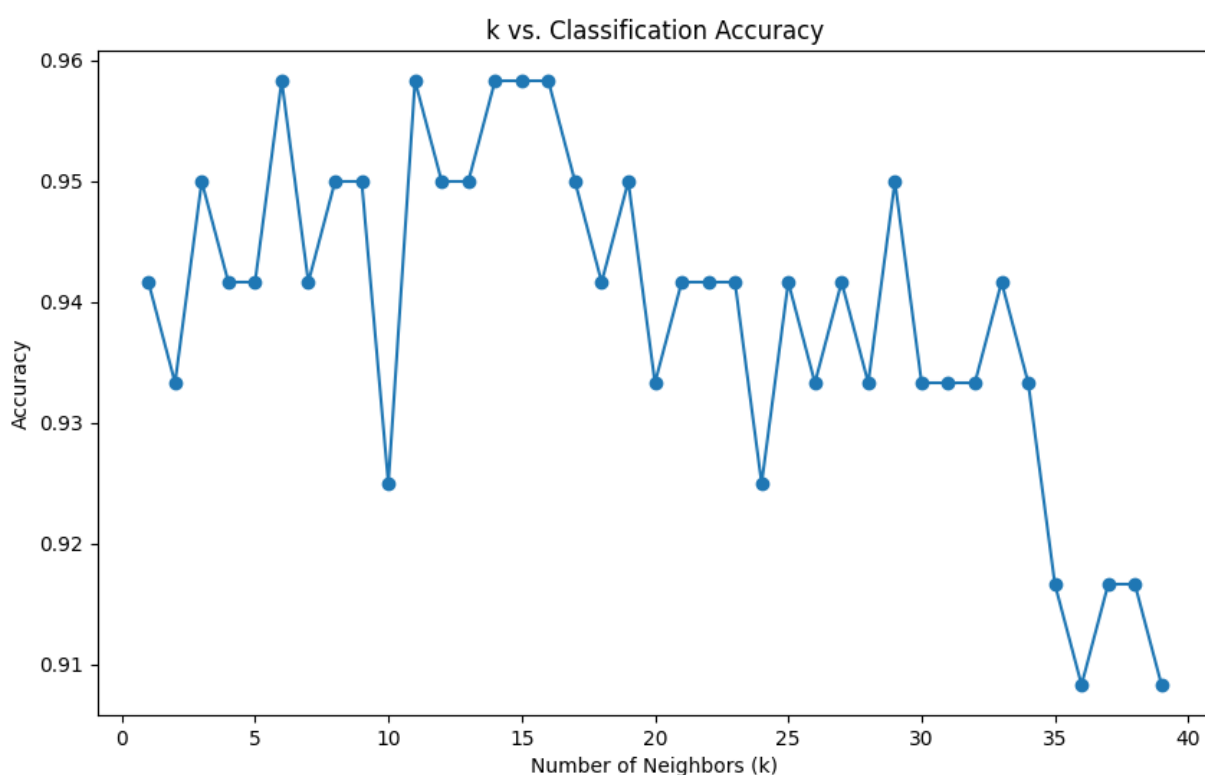
```

23 conf_matrix = confusion_matrix(y_test, y_pred)
24
25 print(optimal_k, test_accuracy, predicted_class, report, conf_matrix)

```

输出：

1. **最优k值**：通过交叉验证，我们发现最优的k值为11。
2. **测试集上的分类准确率**：使用最优k值（11）重新训练的分类器在测试集上的准确率为100%。
3. **新数据点(5, 2.9, 1.0, 0.2)的预测类别**：该数据点被预测为类别0。



1		precision	recall	f1-score	support
2	0	1.00	1.00	1.00	10
3	1	1.00	1.00	1.00	9
4	2	1.00	1.00	1.00	11
5					
6	accuracy			1.00	30
7	macro avg	1.00	1.00	1.00	30
8	weighted avg	1.00	1.00	1.00	30

- precision（精确率）：模型预测为正的样本中实际为正的比率。
- recall（召回率）：实际为正的样本中被模型预测为正的比率。
- f1-score：精确率和召回率的调和平均数。

- support: 每个类别的样本数量。

输出混淆矩阵:

```
1 [[10  0  0]
2  [ 0  9  0]
3  [ 0  0 11]]
```

- 混淆矩阵显示了实际类别与模型预测类别之间的关系。对角线上的元素表示正确预测的数量。

根据以上结果, 最优模型 (k=11) 在测试集上表现出色, 达到了完美的分类准确率。这个模型可以用来预测新的数据点, 正如我们已经用数据点(5, 2.9, 1.0, 0.2)所展示的那样。

重要的代码

1. 数据读取:

```
1 import pandas as pd
2 url = "https://archive.ics.uci.edu/ml/machine-learning-
   databases/iris/iris.data"
3 column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
   'species']
4 iris = pd.read_csv(url, header=None, names=column_names)
```

2. 数据预处理:

```
1 from sklearn.preprocessing import LabelEncoder
2 label_encoder = LabelEncoder()
3 iris['species_encoded'] = label_encoder.fit_transform(iris['species'])
4 iris.drop('species', axis=1, inplace=True)
```

3. 数据集划分:

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test =
   train_test_split(iris.drop('species_encoded', axis=1),
   iris['species_encoded'], test_size=0.2, random_state=42)
```

4. KNN模型训练与评估:

```
1 from sklearn.neighbors import KNeighborsClassifier
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import classification_report, confusion_matrix
4 knn = KNeighborsClassifier(n_neighbors=k)
5 knn.fit(X_train, y_train)
6 test_accuracy = knn.score(X_test, y_test)
```

5. 交叉验证:

```
1 cross_val_scores = cross_val_score(knn, X_train, y_train, cv=10,
    scoring='accuracy')
```

6. 绘制k值与准确率关系图:

```
1 plt.plot(k_range, accuracies, marker='o')
2 plt.title('k vs. Classification Accuracy')
3 plt.xlabel('Number of Neighbors (k)')
4 plt.ylabel('Accuracy')
5 plt.show()
```

7. 评估指标:

```
1 from sklearn.metrics import classification_report, confusion_matrix
2 y_pred = knn.predict(X_test)
3 report = classification_report(y_test, y_pred)
4 conf_matrix = confusion_matrix(y_test, y_pred)
```