


- 0. 背景知识
  - 1.导入包
  - 2. 导入数据集
  - 3. 数据预处理
    - 3.1 检测缺失值
    - 3.2 生成自变量和因变量
    - 3.3 查看样本是否均衡
    - 3.4 把数据拆分成训练集和测试集
    - 3.5 特征缩放
  - 4. 使用不同的参数 构建KNN模型
    - 4.1 模型1
      - 4.1.1 构建KNN模型并训练
      - 4.1.2 预测测试集
      - 4.1.3 生成混淆矩阵
      - 4.1.4 可视化
      - 4.1.5 评估模型性能
    - 4.2 模型2
- 5 实验结论
  - 实验结论
    - 1. KNN可以进行非线性分类
    - 2. 不同超参数对KNN模型性能的影响

# 0. 背景知识

询问chatgpt相关知识:

ChatGPT 4o

介绍一下KNN



KNN (K-Nearest Neighbors, K最近邻) 是一种简单且常用的监督学习算法，主要用于分类和回归问题。KNN的基本思想是通过比较样本之间的距离来进行分类或预测。以下是对KNN模型的详细介绍：

**核心思想**

KNN算法的核心思想是基于相似性进行分类或预测。给定一个新的样本，KNN通过计算该样本与训练集中所有样本的距离，找到最近的K个样本，然后根据这些邻居的标签进行分类或预测。

**算法步骤**

- 选择参数K：**  
确定要使用的邻居数量K，这是KNN算法中唯一的超参数。
- 计算距离：**  
对于给定的待分类样本（或待预测样本），计算它与训练集中所有样本的距离。常用的距离度量方法包括欧氏距离、曼哈顿距离和闵可夫斯基距离。欧氏距离的公式如下：
$$d(x,y)=\sqrt{\sum_{i=1}^n(x_i-y_i)^2}$$

3. **寻找最近的K个邻居：**  
根据计算出的距离，找到离待分类样本最近的K个训练样本。

# 1. 导入包

```
# 导入包
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

## 2. 导入数据集

```
dataset = pd.read_csv('/home/dataset/Social_Network_Ads.csv')
dataset
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

## 3. 数据预处理

### 3.1 检测缺失值

```
# 检测缺失值
blank = dataset.isnull().sum()
blank
```

```
Out[3]: User ID          0
        Gender          0
        Age            0
        EstimatedSalary  0
        Purchased       0
        dtype: int64
```

## 3.2 生成自变量和因变量

```
# 生成自变量和因变量
x = dataset.iloc[:, [2, 3]].values
x[:5, :]
```

```
Out[4]: array([[ 19, 19000],
               [ 35, 20000],
               [ 26, 43000],
               [ 27, 57000],
               [ 19, 76000]])
```

```
y = dataset.iloc[:, 4].values
y[:5]
```

```
Out[5]: array([0, 0, 0, 0, 0])
```

## 3.3 查看样本是否均衡

```
# 查看样本是否均衡
sample_0 = sum(dataset['Purchased']==0)
sample_1 = sum(dataset['Purchased']==1)
print('不买车的样本占总样本的%.2f' %(sample_0/(sample_0 + sample_1)))
```

不买车的样本占总样本的0.64

## 3.4 把数据拆分成训练集和测试集

```
# 将数据拆分成训练集和测试集
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
                                                    random_state = 0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(300, 2)
(100, 2)
(300,)
(100,)
```

## 3.5 特征缩放

```
# 特征缩放
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

## 4. 使用不同的参数 构建KNN模型

### 4.1 模型1

#### 4.1.1 构建KNN模型并训练

```
# 使用不同的参数构建KNN模型
# 模型1: 构建KNN模型并训练模型 (n_neighbors = 5, weights='uniform', metric =
'minkowski', p = 2)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, weights='uniform', metric =
'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

#### 4.1.2 预测测试集

```
# 预测测试集
y_pred = classifier.predict(X_test)
y_pred[:5]
```

```
array([0, 0, 0, 0, 0])
```

#### 4.1.3 生成混淆矩阵

```
# 生成混淆矩阵
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[64  4]
 [ 3 29]]
```

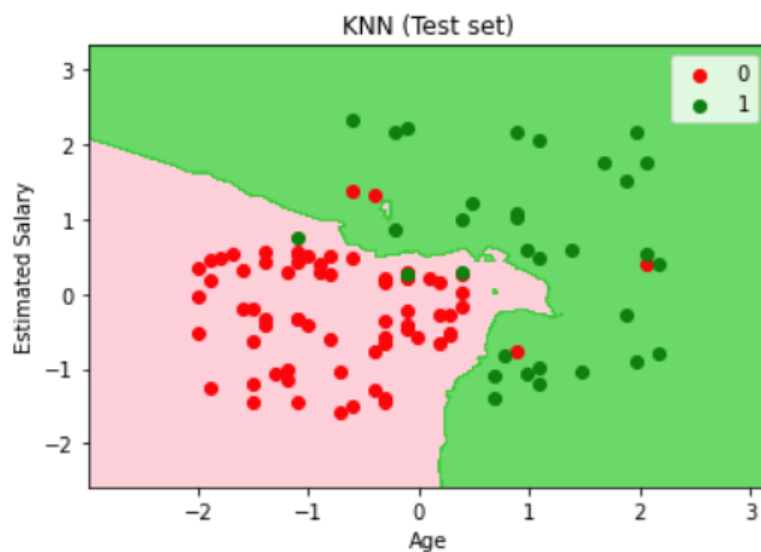
#### 4.1.4 可视化

```
# 可视化测试集的预测结果
from matplotlib.colors import ListedColormap
plt.figure()
X_set, y_set = X_test, y_test
x1, x2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
```

```

        np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('pink', 'limegreen')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('KNN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

```



### 4.1.5 评估模型性能

调用函数库的办法：

```

# 评估模型性能
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))

```

0.93

使用混淆矩阵的办法：

```

(cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])

```

## 4.2 模型2

```

# 模型2：构建KNN模型并训练模型 (n_neighbors = 3, weights='distance', metric =
'minkowski', p = 1)

```

```

classifier = KNeighborsClassifier(n_neighbors = 3, weights='distance', metric =
'minkowski', p = 1)
classifier.fit(X_train, y_train)
# 预测测试集
y_pred = classifier.predict(X_test)
y_pred[:5]
# 生成混淆矩阵
cm = confusion_matrix(y_test, y_pred)
print(cm)
# 可视化测试集的预测结果
plt.figure()
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.75, cmap = ListedColormap(('pink', 'limegreen'))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('KNN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
# 评估模型性能
print(accuracy_score(y_test, y_pred))

```

## 5 实验结论

### 实验结论

#### 1. KNN可以进行非线性分类

在本实验中，我们验证了KNN算法在处理非线性分类任务中的有效性。通过实验，我们发现KNN可以根据样本之间的相似性，在复杂的非线性数据分布中准确地进行分类。这是因为KNN不依赖于数据的线性可分性，而是利用距离度量和邻近样本的标签信息进行分类，从而能够适应各种形状和模式的决策边界。

实验结果表明，在使用KNN进行非线性分类时，即使数据的决策边界非常复杂，KNN也能够通过适当选择邻居数量K，找到合理的分类边界，获得较高的分类精度。具体而言，当K值适中时，KNN能够平滑决策边界，减少噪声影响，提高分类准确性。

#### 2. 不同超参数对KNN模型性能的影响

KNN模型的性能在很大程度上依赖于超参数的选择，尤其是邻居数量K和距离度量方式。通过实验分析，我们总结了以下几点结论：

- 邻居数量K的影响：

- **K值过小**：当K值较小时，模型容易受到噪声和异常值的影响，导致过拟合。这是因为少数邻居可能会包含异常样本，从而使分类结果偏向这些异常点。
- **K值过大**：当K值较大时，模型的决策边界过于平滑，可能导致欠拟合。由于考虑的邻居样本过多，一些重要的局部模式可能被忽略，从而降低分类精度。
- **最佳K值**：实验表明，存在一个最佳的K值范围，使得模型能够在平滑决策边界和保持局部模式之间取得平衡，从而实现最优的分类性能。通常通过交叉验证来确定最佳K值。
- **距离度量方式的影响**：
  - **欧氏距离**：常用于连续变量，适用于大多数情况下的KNN分类任务。实验表明，欧氏距离在处理空间均匀分布的数据时表现良好。
  - **曼哈顿距离**：在某些高维数据或城市街区网格状数据中表现较好。实验发现，曼哈顿距离在某些特定应用场景中优于欧氏距离。
  - **闵可夫斯基距离**：通过调节参数 $p$ ，能够在欧氏距离和曼哈顿距离之间切换，为KNN模型提供了更多灵活性。实验结果显示，不同的 $p$ 值会显著影响分类性能，需根据具体数据特点进行选择。
- **权重策略的影响**：
  - **统一权重**：所有邻居样本具有相同的权重，适用于数据分布均匀的情况。
  - **距离加权**：根据邻居样本与待分类样本的距离赋予不同权重，距离越近权重越大，适用于考虑邻近样本更具代表性的数据分布。实验表明，距离加权策略在处理非均匀分布数据时能有效提高分类精度。

通过一系列实验，我们确认了KNN模型在非线性分类任务中的有效性，同时详细分析了不同超参数对模型性能的影响。最佳的K值、合适的距离度量方式和权重策略能够显著提升KNN模型的性能。未来的研究可以进一步探索KNN在不同应用场景中的优化方法，为更复杂的分类任务提供更强大的解决方案。