

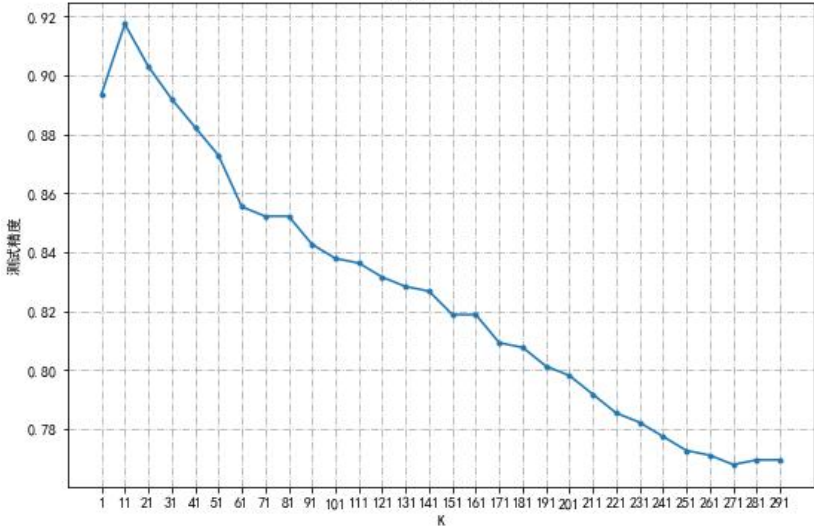
# 中北大学软件学院

## 实 验 报 告

专 业:	软件工程
方 向:	人工智能
课程名称:	机器学习实践
班 级:	22130418
学 号:	2213041835、2213041816
姓 名:	吕炳荣、刘晓峰
辅导教师:	程晓鼎

2024 年 3 月 制

成绩： \_\_\_\_\_

实验时间	2025 年 03 月 23 日 14 时至 18 时	学时数	4 学时																																																														
1. 实验名称																																																																	
空气质量等级的 K-近邻法预测																																																																	
2. 实验目的																																																																	
理解 K-近邻法原理，基于具体数据能对其做应用，掌握 K-近邻法的代码实现。体会 K-近邻法三要素（距离度量、K 值选择、分类决策规则）对算法性能的影响。																																																																	
3.实验内容																																																																	
分析空气质量监测数据，根据数据建立空气质量等级的 K-近邻法预测模型，分析不同距离度量方法、不同 K 取值以及不同分类决策规则下的算法准确率，通过测试误差确定最优参数 K。																																																																	
4. 实验原理或流程图																																																																	
距离度量方法：如曼哈顿距离、欧氏距离、切比雪夫距离等																																																																	
分类决策规则：如多数表决、倒数加权表决、高斯加权表决等																																																																	
通过测试误差确定最优参数 K 的方法：																																																																	
<div>加权K-近邻的测试精度(1-测试误差)变化折线图 (最优参数K=11)</div>  <table><tr><th>K</th><th>测试精度 (1-测试误差)</th></tr><tr><td>1</td><td>0.895</td></tr><tr><td>11</td><td>0.918</td></tr><tr><td>21</td><td>0.905</td></tr><tr><td>31</td><td>0.895</td></tr><tr><td>41</td><td>0.885</td></tr><tr><td>51</td><td>0.875</td></tr><tr><td>61</td><td>0.855</td></tr><tr><td>71</td><td>0.852</td></tr><tr><td>81</td><td>0.852</td></tr><tr><td>91</td><td>0.842</td></tr><tr><td>101</td><td>0.838</td></tr><tr><td>111</td><td>0.838</td></tr><tr><td>121</td><td>0.835</td></tr><tr><td>131</td><td>0.832</td></tr><tr><td>141</td><td>0.828</td></tr><tr><td>151</td><td>0.828</td></tr><tr><td>161</td><td>0.828</td></tr><tr><td>171</td><td>0.812</td></tr><tr><td>181</td><td>0.812</td></tr><tr><td>191</td><td>0.802</td></tr><tr><td>201</td><td>0.802</td></tr><tr><td>211</td><td>0.792</td></tr><tr><td>221</td><td>0.788</td></tr><tr><td>231</td><td>0.782</td></tr><tr><td>241</td><td>0.778</td></tr><tr><td>251</td><td>0.775</td></tr><tr><td>261</td><td>0.772</td></tr><tr><td>271</td><td>0.772</td></tr><tr><td>281</td><td>0.772</td></tr><tr><td>291</td><td>0.772</td></tr></table>				K	测试精度 (1-测试误差)	1	0.895	11	0.918	21	0.905	31	0.895	41	0.885	51	0.875	61	0.855	71	0.852	81	0.852	91	0.842	101	0.838	111	0.838	121	0.835	131	0.832	141	0.828	151	0.828	161	0.828	171	0.812	181	0.812	191	0.802	201	0.802	211	0.792	221	0.788	231	0.782	241	0.778	251	0.775	261	0.772	271	0.772	281	0.772	291	0.772
K	测试精度 (1-测试误差)																																																																
1	0.895																																																																
11	0.918																																																																
21	0.905																																																																
31	0.895																																																																
41	0.885																																																																
51	0.875																																																																
61	0.855																																																																
71	0.852																																																																
81	0.852																																																																
91	0.842																																																																
101	0.838																																																																
111	0.838																																																																
121	0.835																																																																
131	0.832																																																																
141	0.828																																																																
151	0.828																																																																
161	0.828																																																																
171	0.812																																																																
181	0.812																																																																
191	0.802																																																																
201	0.802																																																																
211	0.792																																																																
221	0.788																																																																
231	0.782																																																																
241	0.778																																																																
251	0.775																																																																
261	0.772																																																																
271	0.772																																																																
281	0.772																																																																
291	0.772																																																																

## 5. 实验过程或源代码

```
# -*- coding: utf-8 -*-
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,
GridSearchCV, cross_val_score, validation_curve
from sklearn.preprocessing import StandardScaler,
LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,
classification_report
import warnings

# --- 0. 全局设置 ---

warnings.filterwarnings('ignore') # 忽略一些不影响结果的警告

plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置中文字体
为黑体

plt.rcParams['axes.unicode_minus'] = False # 解决负号显示
问题

sns.set_style('whitegrid') # 设置 seaborn 风格

DATA_FILE = '北京市空气质量数据.xlsx' # 数据文件路径

RANDOM_STATE = 42 # 随机种子，保证结果可复现

TEST_SIZE = 0.3 # 测试集比例

CV_FOLDS = 5 # 交叉验证折数

MAX_K = 20 # 探索的最大 K 值
```

```
# --- 1. 数据加载与预处理 ---

print("--- 1. 数据加载与预处理 ---")

try:
    df = pd.read_excel(DATA_FILE)
    print(f"成功加载数据: {DATA_FILE}")

    print("数据前 5 行:\n", df.head())

    print("\n 数据信息:")
    df.info()
except FileNotFoundError:
    print(f"错误: 未找到数据文件 {DATA_FILE}")
    exit()

# 特征选择 (选择 'PM2.5' 到 'O3' 之间的所有列作为特征)
# 假设 'AQI', '质量等级' 在前两列, 污染物特征从第 3 列(索引 2)开始, 直到最后一列'O3'(索引 -1)

X = df.iloc[:, 3:-1].values # 选择 PM2.5, PM10, SO2, CO, NO2, O3 作为特征 (注意索引调整)

y_raw = df['质量等级'].values

print("\n 选择的特征 (X 前 5 行):\n", X[:5])

print("原始目标变量 (y_raw 前 5 个):", y_raw[:5])

# 目标变量标签编码
le = LabelEncoder()
y_encoded = le.fit_transform(y_raw)

print("\n 编码后的目标变量 (y_encoded 前 5 个):", y_encoded[:5])
```

```
print("目标变量类别:", le.classes_)

# 数据标准化 (对特征进行)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("\n 标准 化 后 的 特 征    (X_scaled 前 5 行 ):\n",
X_scaled[:5])

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_encoded,
    test_size=TEST_SIZE,
    random_state=RANDOM_STATE,
    stratify=y_encoded # 保持类别比例
)
print(f"\n 数据集划分为：训练集  {X_train.shape[0]} 条，测
试集  {X_test.shape[0]} 条")

# --- 2. 探索性数据分析 (EDA) 可视化 ---

print("\n--- 2. 生成 EDA 可视化图表 ---")
plt.figure(figsize=(18, 10))
plt.suptitle("空气质量数据探索性分析", fontsize=16, y=1.02)

# 质量等级分布
plt.subplot(2, 3, 1)
sns.countplot(x=df['质量等级'], order=df['质量等级'].value_counts().index, palette='viridis')
plt.title('质量等级分布')

plt.ylabel('样本数量')
```

```
# PM2.5 分布
plt.subplot(2, 3, 2)
sns.histplot(df['PM2.5'], kde=True, color='skyblue')
plt.title('PM2.5 浓度分布')
plt.xlabel('PM2.5 ( $\mu\text{g}/\text{m}^3$ )')
plt.ylabel('频数')

# 主要污染物箱线图
plt.subplot(2, 3, 3)
# 使用原始 X 数据进行可视化更有意义
sns.boxplot(data=df.iloc[:, 3:-1], palette='Set2')
plt.xticks(rotation=30)
plt.title('主要污染物浓度箱线图')
plt.ylabel('浓度')

# 特征相关性热力图
plt.subplot(2, 3, 4)
# 使用原始 X 数据进行可视化更有意义
corr_matrix = df.iloc[:, 3:-1].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',
            fmt=".2f", linewidths=.5)
plt.title('污染物相关性矩阵')

# O3 与 PM2.5 关系散点图
plt.subplot(2, 3, 5)
sns.scatterplot(data=df, x='PM2.5', y='O3', hue='质量等级',
                palette='magma', alpha=0.7)
plt.title('PM2.5 与 O3 关系 (按质量等级着色)')
plt.xlabel('PM2.5 ( $\mu\text{g}/\text{m}^3$ )')
plt.ylabel('O3 ( $\mu\text{g}/\text{m}^3$ )')
```

```

plt.tight_layout(rect=[0, 0, 1, 0.98]) # 调整布局防止标题重叠
plt.show()

# --- 3. K-近邻模型构建与参数选择 ---

print("\n--- 3. K-近邻模型构建与参数选择 ---")

# 定义高斯权重函数 (与原代码一致)
def gaussian_weight(distances):
    # 避免除以零或极小距离导致权重过大
    epsilon = 1e-6
    return np.exp(-0.5 * (distances**2) / np.mean(distances +
epsilon)**2) # 稍微调整以增加稳定性

# 定义参数网格
param_grid = {
    'n_neighbors': list(range(1, MAX_K + 1)),
    'metric': ['euclidean', 'manhattan', 'chebyshev'],
    # 'weights': ['uniform', 'distance'] # 暂时不包括自定义权重，GridSearchCV 不支持直接传入函数名
    'weights': ['uniform', 'distance']
}

# 如果需要测试自定义权重，可以在 GridSearch 后单独进行或修改 GridSearchCV 流程

# 使用 GridSearchCV 进行参数搜索 (寻找最佳 K、距离度量、基础权重)

print(f"\n 使用    GridSearchCV 寻找最佳参数组合 (K=1-{MAX_K}, 距离度量, 基础权重)...")

knn = KNeighborsClassifier()

```

```

grid_search = GridSearchCV(knn, param_grid, cv=CV_FOLDS,
scoring='accuracy', n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

# 输出最佳参数

best_params_basic = grid_search.best_params_
best_score_basic = grid_search.best_score_

print(f"\nGridSearchCV 找到的最佳基础参数 :
{best_params_basic}")

print(f"对应的交叉验证准确率: {best_score_basic:.4f}")

# 考虑自定义高斯权重 (使用找到的最佳 K 和 metric)

best_k = best_params_basic.get('n_neighbors', 5) # 使用找到的
最佳 K, 若失败则用 5

best_metric = best_params_basic.get('metric', 'manhattan') # 使
用找到的最佳 metric

knn_gaussian = KNeighborsClassifier(n_neighbors=best_k,
metric=best_metric, weights=gaussian_weight)
scores_gaussian = cross_val_score(knn_gaussian, X_train,
y_train, cv=CV_FOLDS, scoring='accuracy', n_jobs=-1)
mean_score_gaussian = np.mean(scores_gaussian)

print(f"使用高斯权重的交叉验证准确率 (k={best_k},
metric={best_metric}): {mean_score_gaussian:.4f}")

# 确定最终最佳参数 (比较基础权重和高斯权重)

best_weights = best_params_basic['weights']
best_score = best_score_basic
if mean_score_gaussian > best_score_basic:
    best_weights = gaussian_weight
    best_score = mean_score_gaussian
    print("高斯权重表现更优。")
else:

```



```

        print(f"{best_weights.capitalize() if
isinstance(best_weights, str) else '基础'}权重表现更优或相当。
")

final_best_params = {'n_neighbors': best_k, 'metric':
best_metric, 'weights': best_weights}
print(f"\n 最终确定的最佳参数组合: { {k: (v if isinstance(v,
(str, int)) else 'gaussian') for k, v in
final_best_params.items()} }") # 打印时显示函数名

print(f"对应的最佳交叉验证准确率: {best_score:.4f}")

# --- 4. 分析 K 值对模型性能的影响 ---

print("\n--- 4. 分析 K 值对模型性能的影响 ---")

# 使用 validation_curve 分析 K 值影响 (使用找到的最佳
metric 和 weights)
train_scores, valid_scores = validation_curve(
    KNeighborsClassifier(metric=best_metric,
weights=best_weights),
    X_train, y_train, # 在训练集上进行分析更标准
    param_name="n_neighbors",
    param_range=list(range(1, MAX_K + 1)),
    cv=CV_FOLDS,
    scoring="accuracy",
    n_jobs=-1
)

# 计算平均得分和误差
train_mean = np.mean(train_scores, axis=1)
valid_mean = np.mean(valid_scores, axis=1)
train_error = 1 - train_mean
valid_error = 1 - valid_mean

```

```

# 绘制 K 值与误差率曲线

plt.figure(figsize=(10, 6))
plt.plot(list(range(1, MAX_K + 1)), train_error, 'o-', color='r',
label='训练误差率')

plt.plot(list(range(1, MAX_K + 1)), valid_error, 'o-', color='g',
label='交叉验证误差率')

plt.xlabel('K 值 (n_neighbors)')

plt.ylabel('误差率')

plt.title(f'K 值对 KNN 模型性能的影响
(metric={best_metric}, weights={"gaussian" if
callable(best_weights) else best_weights}'))
plt.xticks(list(range(1, MAX_K + 1)))
plt.legend()
plt.grid(True)

# 标记最佳 K 值点

plt.axvline(x=best_k, color='blue', linestyle='--', label=f'最佳
K = {best_k}')

plt.legend() # 更新图例以包含最佳 K 线

plt.show()

optimal_k_from_curve = np.argmin(valid_error) + 1 # 从曲线
中找到误差最低的 K

print(f" 从 验 证 曲 线 看 ， 最 优 K 值 约 为 :
{optimal_k_from_curve} (交叉验证误差最低点)")

print(f"GridSearch 找到的最优 K 值为: {best_k}")

# --- 5. 分析不同距离度量和分类决策规则的影响 ---

```

```

print("\n--- 5. 分析不同参数的影响 (参数消融实验) ---")

# 辅助绘图函数
def plot_comparison(scores_dict, title, xlabel, ylabel, palette):
    plt.figure(figsize=(8, 5))
    keys = list(scores_dict.keys())
    values = list(scores_dict.values())
    sns.barplot(x=keys, y=values, palette=palette)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

    # 在柱状图上显示数值
    for i, v in enumerate(values):
        plt.text(i, v + 0.005, f'{v:.3f}', ha='center',
va='bottom')

    plt.ylim(min(values) * 0.95, max(values) * 1.05) # 调整 y
轴范围

    plt.show()

# a) 不同距离度量对比 (使用最佳 K 和 'distance' 权重)
metrics_to_compare = ['euclidean', 'manhattan', 'chebyshev']
metric_scores = {}

print(f"\n 比较不同距离度量 (固定 k={best_k},
weights='distance'):")
for m in metrics_to_compare:
    knn_metric_test =
KNeighborsClassifier(n_neighbors=best_k, metric=m,
weights='distance') # 使用 distance 权重对比

    scores = cross_val_score(knn_metric_test, X_train,
y_train, cv=CV_FOLDS, scoring='accuracy', n_jobs=-1)
    metric_scores[m] = np.mean(scores)

    print(f"- {m}: 平均准确率 = {metric_scores[m]:.4f}")

plot_comparison(metric_scores, '不同距离度量对准确率的影响

```

```

', '距离度量', '平均交叉验证准确率', 'rocket')

# b) 不同分类决策规则对比 (使用最佳 K 和最佳 metric)

weights_to_compare = {'uniform': '多数表决', 'distance': '距离
加权', 'gaussian_weight': '高斯加权'}

weight_scores = {}

print(f"\n 比较不同分类决策规则 (固定 k={best_k},
metric={best_metric}):")
for w_func, w_name in weights_to_compare.items():
    knn_weight_test =
KNeighborsClassifier(n_neighbors=best_k, metric=best_metric,
weights=w_func)
    scores = cross_val_score(knn_weight_test, X_train,
y_train, cv=CV_FOLDS, scoring='accuracy', n_jobs=-1)
    weight_scores[w_name] = np.mean(scores)

    print(f"- {w_name}: 平均准确率 =
{weight_scores[w_name]:.4f}")

plot_comparison(weight_scores, '不同分类决策规则对准确率的
影响', '分类决策规则', '平均交叉验证准确率', 'mako')

# --- 6. 参数组合热力图 (来自 GridSearchCV 结果) ---

print("\n--- 6. 可视化参数组合性能 (GridSearchCV 热力图)
---")
try:
    results_df = pd.DataFrame(grid_search.cv_results_)
    # 选择只包含 uniform 和 distance 权重的行进行透视
    results_subset = results_df[results_df['param_weights'] !=
gaussian_weight] # 排除自定义函数行

    pivot_table =
results_subset.pivot_table(index='param_n_neighbors',

```

```

columns=['param_metric', 'param_weights'], # 多列组合

values='mean_test_score')

    # 为了简化显示，可以分别绘制不同权重的热力图
    for weight_type in ['uniform', 'distance']:
        pivot_subset =
results_subset[results_subset['param_weights'] ==
weight_type].pivot_table(
            index='param_n_neighbors',
            columns='param_metric',
            values='mean_test_score'
        )
        plt.figure(figsize=(10, 7))
        sns.heatmap(pivot_subset, annot=True, fmt=".3f",
cmap="viridis", linewidths=.5)

        plt.title(f'KNN 准确率热力图 (权重 :
{weight_type.capitalize()}')

        plt.xlabel('距离度量 (Metric)')

        plt.ylabel('K 值 (n_neighbors)')

        plt.show()

except Exception as e:

    print(f"绘制热力图时出错: {e}. 可能 GridSearchCV 结
果格式不兼容或无数据。")

# --- 7. 最终模型评估 ---

print("\n--- 7. 在测试集上评估最终模型 ---")

# 使用找到的最佳参数创建最终模型

final_knn = KNeighborsClassifier(**final_best_params) # 使用

```

```
** 解包字典传入参数

# 训练最终模型 (使用所有训练数据)
final_knn.fit(X_train, y_train)
print("\n 最终模型已使用最佳参数在整个训练集上训练完成。")

# 在测试集上进行预测
y_pred = final_knn.predict(X_test)

# 评估模型性能
test_accuracy = accuracy_score(y_test, y_pred)
print(f"\n 最终模型在测试集上的准确率: {test_accuracy:.4f}")
print("\n 测试集分类报告:")

# 使用 le.inverse_transform 将编码后的 y_test 和 y_pred
转回原始标签显示

# 但 classification_report 本身可以接受 target_names 参数
print(classification_report(y_test, y_pred,
target_names=le.classes_))

print("\n--- 实验结束 ---")
```

--- 1. 数据加载与预处理 ---

成功加载数据：北京市空气质量数据.xlsx

数据前5行：

	日期	AQI	质量等级	PM2.5	PM10	S02	CO	N02	O3
0	2014-01-01	81	良	45	111	28	1.5	62	52
1	2014-01-02	145	轻度污染	111	168	69	3.4	93	14
2	2014-01-03	74	良	47	98	29	1.3	52	56
3	2014-01-04	149	轻度污染	114	147	40	2.8	75	14
4	2014-01-05	119	轻度污染	91	117	36	2.3	67	44

数据信息：

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 2155 entries, 0 to 2154

Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
0	日期	2155 non-null	datetime64[ns]
1	AQI	2155 non-null	int64
2	质量等级	2155 non-null	object
3	PM2.5	2155 non-null	int64
4	PM10	2155 non-null	int64
5	S02	2155 non-null	int64
6	CO	2155 non-null	float64
7	N02	2155 non-null	int64
8	O3	2155 non-null	int64

```
7 N02 2155 non-null int64
8 O3 2155 non-null int64
dtypes: datetime64[ns](1), float64(1), int64(6), object(1)
memory usage: 151.6+ KB
```

选择的特征 (X 前5行):

```
[[ 45. 111. 28. 1.5 62. ]
 [111. 168. 69. 3.4 93. ]
 [ 47. 98. 29. 1.3 52. ]
 [114. 147. 40. 2.8 75. ]
 [ 91. 117. 36. 2.3 67. ]]
```

原始目标变量 (y\_raw 前5个): ['良' '轻度污染' '良' '轻度污染' '轻度污染']

编码后的目标变量 (y\_encoded 前5个): [4 5 4 5 5]

目标变量类别: ['严重污染' '中度污染' '优' '无' '良' '轻度污染' '重度污染']

标准化后的特征 (X\_scaled 前5行):

```
[[-0.32434822  0.30751958  1.29814083  0.54735734  0.72985305]
 [ 0.78230743  1.15235245  4.2960559  2.80226691  2.10192174]
 [-0.2908132  0.1148384  1.37126071  0.30999844  0.28725025]
 [ 0.83260996  0.84109823  2.17557938  2.0901902  1.30523669]
 [ 0.44695723  0.39644935  1.88309986  1.49679295  0.95115445]]
```

数据集划分为：训练集 1508 条，测试集 647 条

--- 2. 生成 EDA 可视化图表 ---

--- 3. K-近邻模型构建与参数选择 ---

使用 GridSearchCV 寻找最佳参数组合 (K=1-20, 距离度量, 基础权重)...

Fitting 5 folds for each of 120 candidates, totalling 600 fits

GridSearchCV 找到的最佳基础参数: {'metric': 'manhattan', 'n\_neighbors': 20, 'weights': 'distance'}  
对应的交叉验证准确率: 0.7334

使用高斯权重交叉验证准确率 (k=20, metric=manhattan): 0.7341

高斯权重表现更优。

最终确定的最佳参数组合: {'n\_neighbors': 20, 'metric': 'manhattan', 'weights': 'gaussian'}

对应的最佳交叉验证准确率: 0.7341

--- 4. 分析 K 值对模型性能的影响 ---

从验证曲线看, 最优 K 值约为: 19 (交叉验证误差最低点)

GridSearch 找到的最优 K 值为: 20

--- 5. 分析不同参数的影响 (参数消融实验) ---

比较不同距离度量 (固定 k=20, weights='distance'):

- euclidean: 平均准确率 = 0.7221
- manhattan: 平均准确率 = 0.7334
- chebyshev: 平均准确率 = 0.6910

比较不同分类决策规则 (固定 k=20, metric=manhattan):

- 多数表决: 平均准确率 = 0.7201
- 距离加权: 平均准确率 = 0.7334
- 高斯加权: 平均准确率 = 0.7341

--- 6. 可视化参数组合性能 (GridSearchCV 热力图) ---

--- 7. 在测试集上评估最终模型 ---

最终模型已使用最佳参数在整个训练集上训练完成。

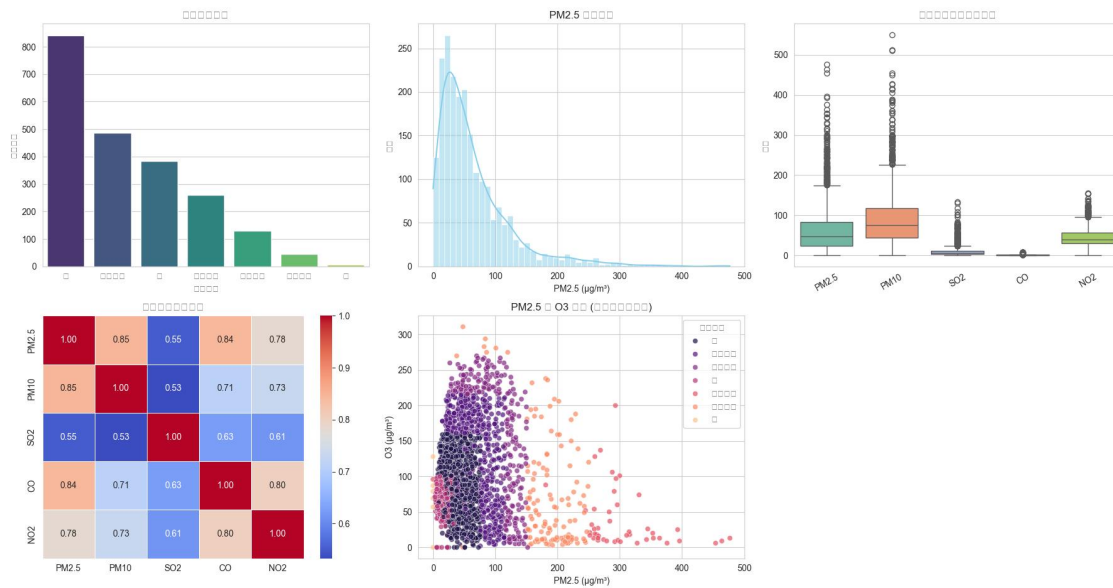
最终模型在测试集上的准确率: 0.7295

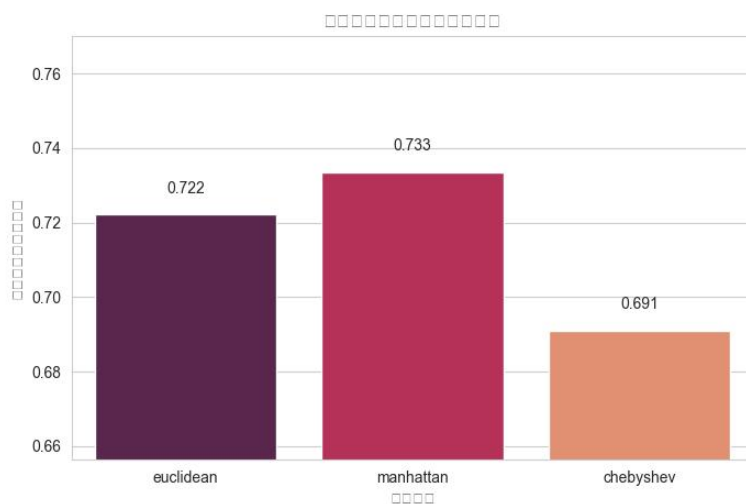
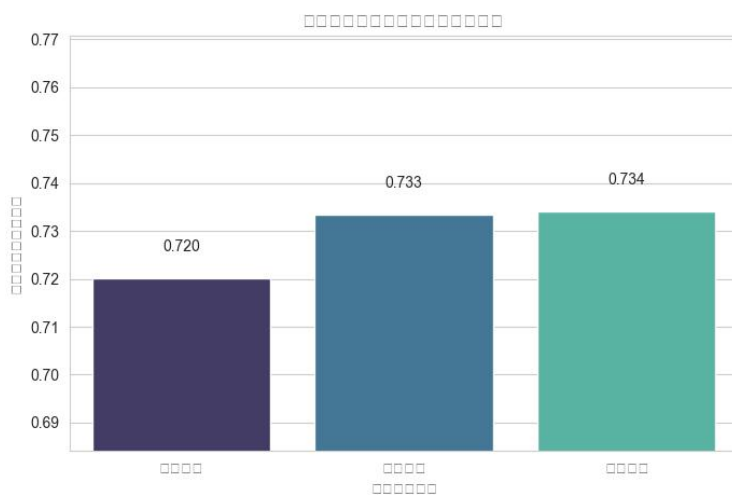
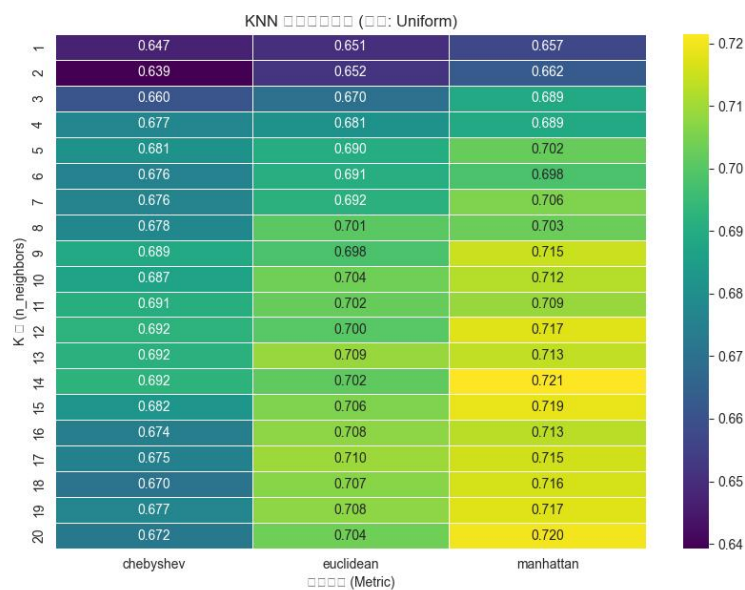


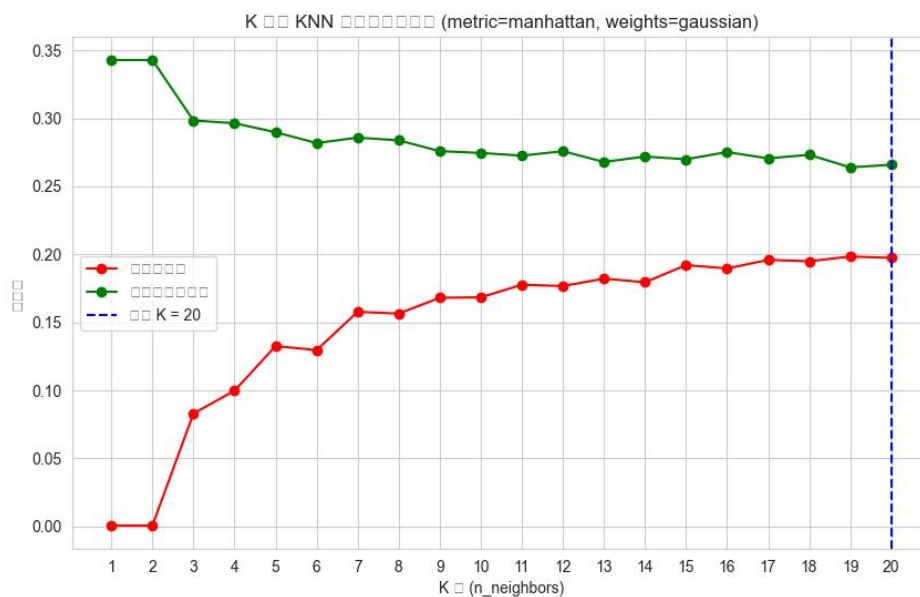
### 测试集分类报告：

	precision	recall	f1-score	support
严重污染	0.91	0.71	0.80	14
中度污染	0.63	0.49	0.55	78
优	0.79	0.90	0.84	115
无	0.00	0.00	0.00	2
良	0.74	0.85	0.79	253
轻度污染	0.65	0.56	0.60	146
重度污染	0.86	0.64	0.74	39
accuracy			0.73	647
macro avg	0.65	0.59	0.62	647
weighted avg	0.72	0.73	0.72	647

--- 实验结束 ---







## 6. 实验心得

通过本次实验，我不仅掌握了 K-NN 算法的基本原理和 Python 实现（基于 scikit-learn 库），更深刻体会到：

**理论与实践结合的重要性：**理解 K-NN 的三个核心要素（K、距离、权重）如何在实际代码中体现，并通过调整它们观察对模型性能的直接影响，远比单纯阅读理论更具启发性。

**数据预处理的关键性：**对于 K-NN 这类基于距离的算法，特征标准化是不可或缺的一步，直接关系到模型的有效性。

**模型调优的必要性与方法：**没有所谓的“万能”参数设置，必须根据具体数据集进行调

优。GridSearch 和 Validation Curve 是进行系统化参数选择和理解模型行为的有力工具。

评估指标的全面性：单一的准确率可能无法完全反映模型在不平衡数据集上的表现，结合分类报告中的精确率、召回率、F1 分数能提供更全面的视角。

算法的局限性：K-NN 计算复杂度较高（尤其是在大数据集上预测时，需要计算与所有训练样本的距离），且对数据维度敏感（高维稀疏数据下距离度量可能失效），存储需求也较大（需要保存整个训练集）。

总的来说，这次实验是一次宝贵的机器学习实践经历，它锻炼了我的数据分析、模型构建、参数调优和结果解读能力，并为后续学习更复杂的算法打下了基础。