模式识别与机器学习 – 实验报告 1

姓名: 年骏杰 学号: 24307140012 专业：AICS 学院：计算机学院

本实验聚焦于课程第三讲决策树与最近邻问题的内容，包含以下三部分：

- 分类评测指标 (20%)
- 决策树 (40%)
- 最近邻问题 (40%)

占课程总分：20% | 提交截至日期：**10/13 23:59**

分类评测指标 – 20%

1. **指标定义 – 5%**

   用你的话简单解释下列每个指标的含义以及使用场景

	含义	使用场景
Accuracy	预测正确的样本数占总样本数的比例	适用于整体评价模型好坏的场景，尤其是类别均衡时
MSE	均方误差，预测值与真实值差的平方的平均值	常用于回归问题的误差衡量
Precision	预测为正类的样本中，真正为正类的比例	适用于关注预测结果中“正确性”的场景，如垃圾邮件检测
Recall	真实为正类的样本中，被正确预测为正类的比例	适用于关注漏判的场景，如疾病筛查
F1	Precision 和 Recall 的调和平均	适用于需要在 Precision 与 Recall 之间取得平衡的场景，如类别不均衡的数据集

2. **代码实现（见Part1_ReadMe.md) – 10%**

   请在报告中贴出你实现的五个核心函数的代码并简单说明实现的逻辑：

   accuracy_score()

```
# ================ TODO (students) ================
accuracy = np.sum(y_true == y_pred) / y_true.shape[0]
return accuracy
# =================================================
```

   y_true：真实标签
   y_pred：模型预测结果
   np.sum(y_true == y_pred)：统计预测正确的样本数量
   y_true.shape[0]：样本总数
   预测正确的样本数 ÷ 总样本数 → 准确率

   mean_squared_error()

```
# ================ TODO (students) ================
error = np.mean((y_true - y_pred) ** 2)
return error
# =================================================
```

   参数同上
   (y_true - y_pred) ** 2：平方误差
   np.mean：取平均值
   计算预测值和真实值的平方差的均值

precision_score()

```
    # ================ TODO (students) ================
    precision = tp / (tp + fp + EPS)
    return precision
    # =================================================
```

tp：真正例，预测为正且实际为正的数量
fp：假正例，预测为正但实际为负的数量
EPS：极小数，防止分母为 0
在所有预测为正的样本中，有多少是真正的正样本

recall_score()

```
    # ================ TODO (students) ================
    recall = tp / (tp + fn + EPS)
    return recall
    # =================================================
```

tp：真正例
fn：假负例，实际为正但预测为负
EPS：防止分母为 0
在所有实际为正的样本中，有多少被模型正确识别

f1_score()

```
# ================ TODO (students) ================
p = precision_score(y_true, y_pred)
r = recall_score(y_true, y_pred)
f1 = 2 * (p * r) / (p + r + EPS)
return f1
# =================================================
```

p, r：分别是精确率和召回率。
F1 是精确率和召回率的调和平均数。

3. **测试与结果 – 5%**
完成评测指标的函数后，运行 test.py，把输出log的截图粘贴在下面一行：

```
============================================================
实验: Part 1 - 分类评测指标
姓名: 年骏杰    学号: 24307140012
时间: 2025-10-07 12:15:32
============================================================
[BASIC] accuracy / MSE
 - accuracy expected 0.75 -> got 0.75 : PASS
 - mse      expected 0.6667 -> got 0.6667 : PASS
[PRF] precision / recall / f1 (binary)
 - precision expected 0.50 -> got 0.50 : PASS
 - recall    expected 0.50 -> got 0.50 : PASS
 - f1        expected 0.50 -> got 0.50 : PASS
```

删掉上面的示例图片，改成你自己的结果，记得在test.py中填上你的名字和学号
使用以下新的输入测试，你可以再test.py中更改，或自己计算

Y_true	Y_predict
[0, 1, 0, 1, 1, 0, 1, 0, 0]	[1, 1, 1, 1, 1, 0, 0, 0, 0]

Confusion Matrix

TP	FP	FN	TN
3	2	1	3

指标计算

Accuracy	MSE	Precision	Recall	F1
0.667	0.333	0.600	0.750	0.667

决策树 – 40%

1. 简要解释下决策树，以及其优缺点 – 2%

决策树概述	优缺点
决策树是一种基于树形结构的分类与回归方法。它通过对数据特征进行逐步划分，将样本空间划分成若干个子空间，最终在叶子节点给出预测结果	**优点：** 结构清晰、易于可视化，可解释性强； 训练速度较快，对数据的预处理要求较低； **缺点：** 容易过拟合，泛化能力不足； 对噪声和数据的轻微变化敏感；

2. **代码实现（见Part2_ReadMe.md） – 15%**

请在报告中贴出你实现的四个核心函数的代码并简单说明实现的逻辑：

criterion.py

- __info_gain(...)

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
def entropy(labels, total_count):
    ent = 0.0
    for count in labels.values():
        p = count / total_count
        ent -= p * math.log2(p)
    return ent
total_count = len(y)
left_count = len(l_y)
right_count = len(r_y)
before = entropy(all_labels, total_count)
after = (left_count / total_count) * entropy(left_labels, left_count) + (right_count / total_count) * entropy(right_labels, right_count)
info_gain = before - after
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

y: 父节点的标签列表；l_y、r_y 是按某个划分条件切分后的左右子集标签列表

all_labels、left_labels、right_labels: 分别是父节点、左/右子节点的标签计数字典

left_count、right_count: 左/右子集的样本数

entropy根据某个节点内各类别的计数字典labels以及该节点样本总数total_count计算熵

before: 划分前父节点的熵

after: 划分后两个子节点熵的加权平均，权重是子集样本占比

info_gain = before - after

- __info_gain_ratio(...)

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
def split_info(left_count, right_count, total_count):
    si = 0.0
    for count in [left_count, right_count]:
        if count == 0:
            continue
        p = count / total_count
        si -= p * math.log2(p)
    return si
total_count = len(y)
left_count = len(l_y)
right_count = len(r_y)
si = split_info(left_count, right_count, total_count)
if si == 0:
    return 0
info_gain /= si
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

left_count、right_count、total_count: 左/右/父样本数。
split_info = -∑ (|S_i|/|S|) log2(|S_i|/|S|)：只与切分后的份数及其大小比例有关
增益率是info_gain / split_info
当 si==0（例如所有样本都进同一侧）时直接返回 0，避免除零

- __gini_index(...)

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
def gini(labels, total_count):
    gini_index = 1.0
    for count in labels.values():
        p = count / total_count
        gini_index -= p * p
    return gini_index
total_count = len(y)
left_count = len(l_y)
right_count = len(r_y)
before = gini(all_labels, total_count)
after = (left_count / total_count) * gini(left_labels, left_count) + (right_count / total_count) * gini(right_labels, right_count)
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

输入参数同上
gini = 1 - ∑ p^2
与熵类似，这里也计算分裂后的加权基尼，并比较 before - after

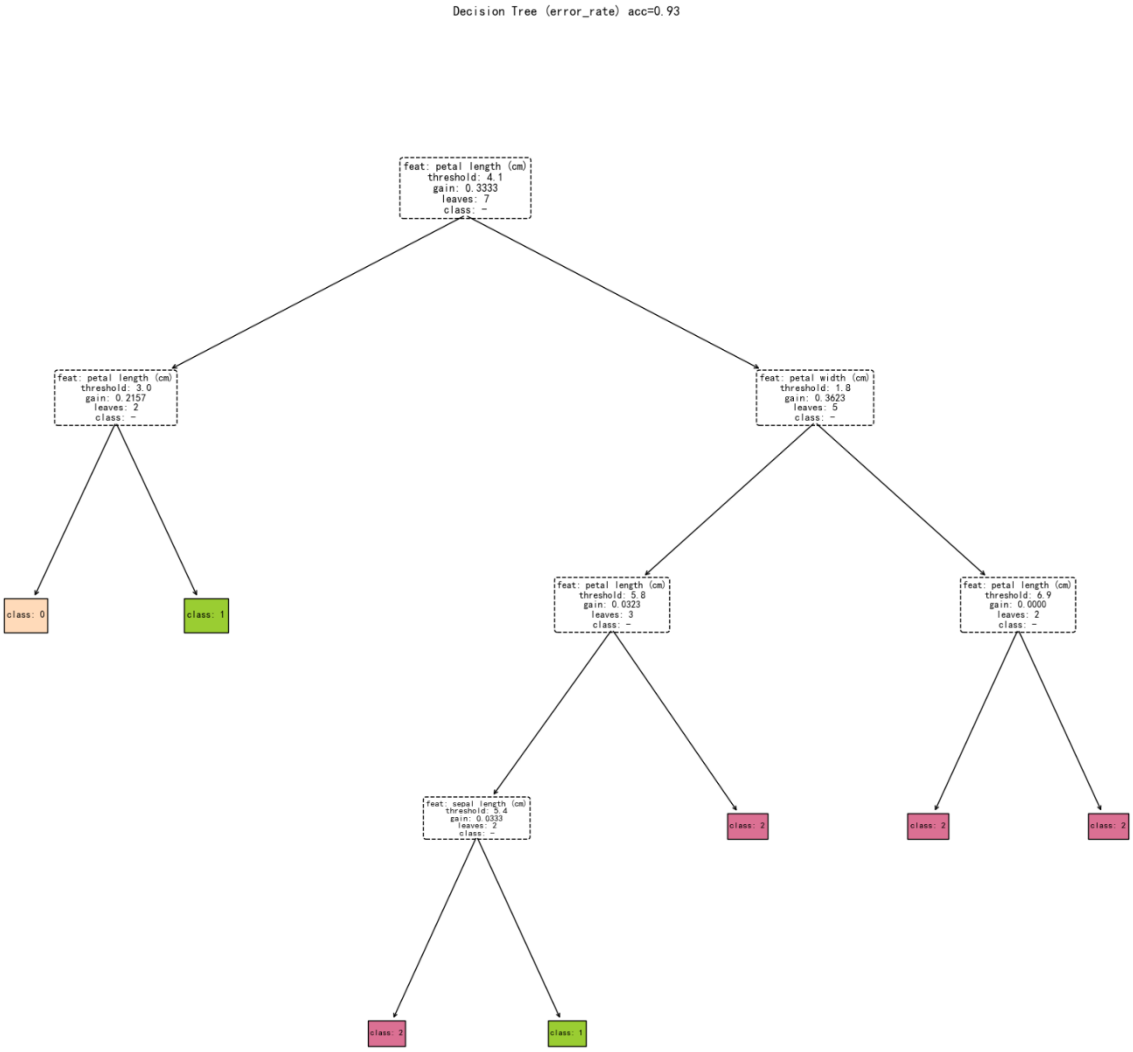
- __error_rate(...)

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
def error_rate(labels, total_count):
    if total_count == 0:
        return 0
    max_count = max(labels.values())
    return 1 - (max_count / total_count)
total_count = len(y)
left_count = len(l_y)
right_count = len(r_y)
before = error_rate(all_labels, total_count)
after = (left_count / total_count) * error_rate(left_labels, left_count) + (right_count / total_count) * error_rate(right_labels, right_count)
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

输入参 同上。
error_rate = 1 - max_class_prob：把节点全预测成多数类时的错误率。
也是计算分裂后的加权误分类率，比较 before - after。

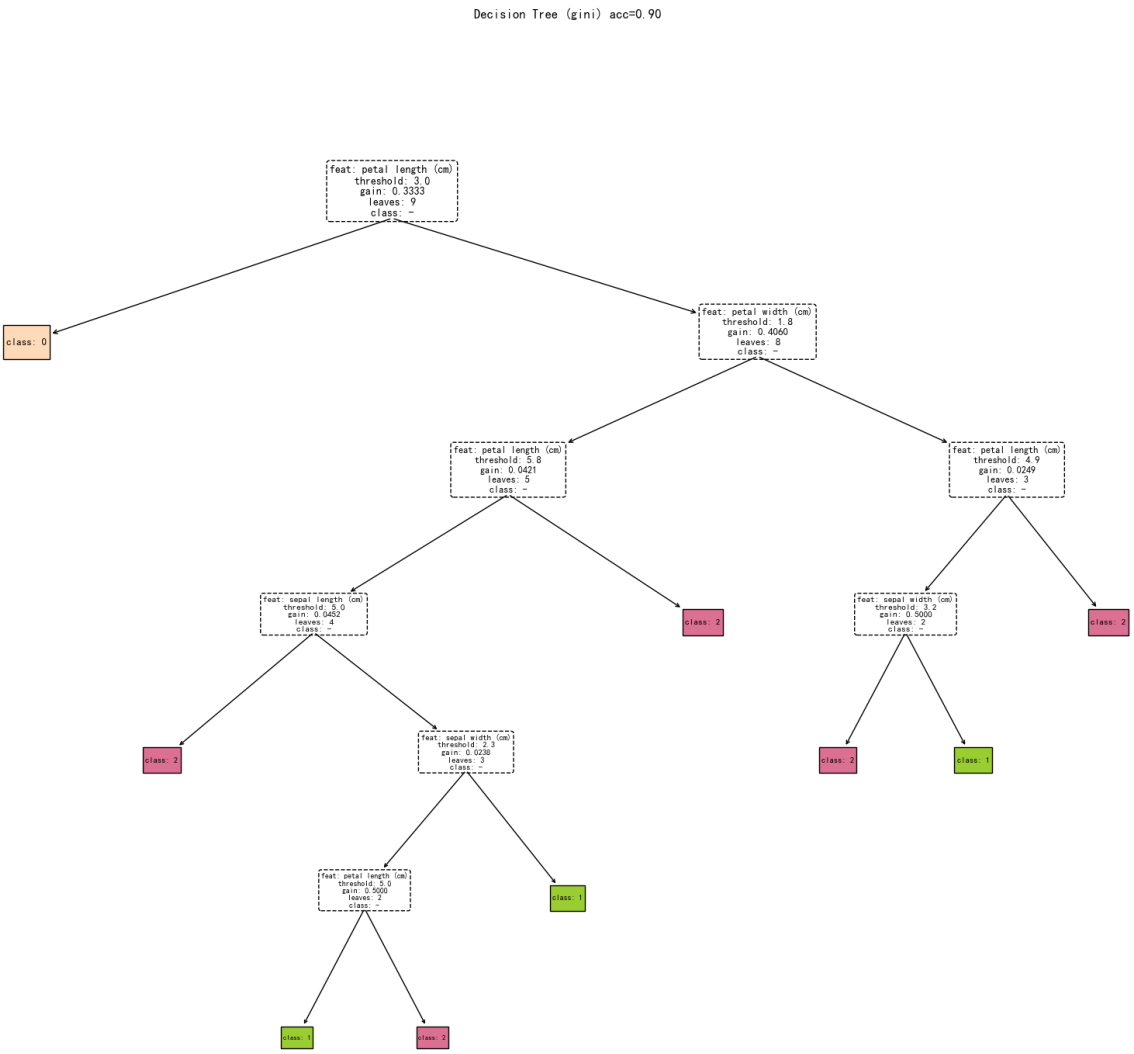
3. 测试和可视化 – 15%

完成criterion.py的四个函数后，运行test_decision_tree.py，将会输出对应的accuracy和四张图片，只需要把图片粘贴在下面，每张图的图注写明：**Accuracy、树深度、叶子数**



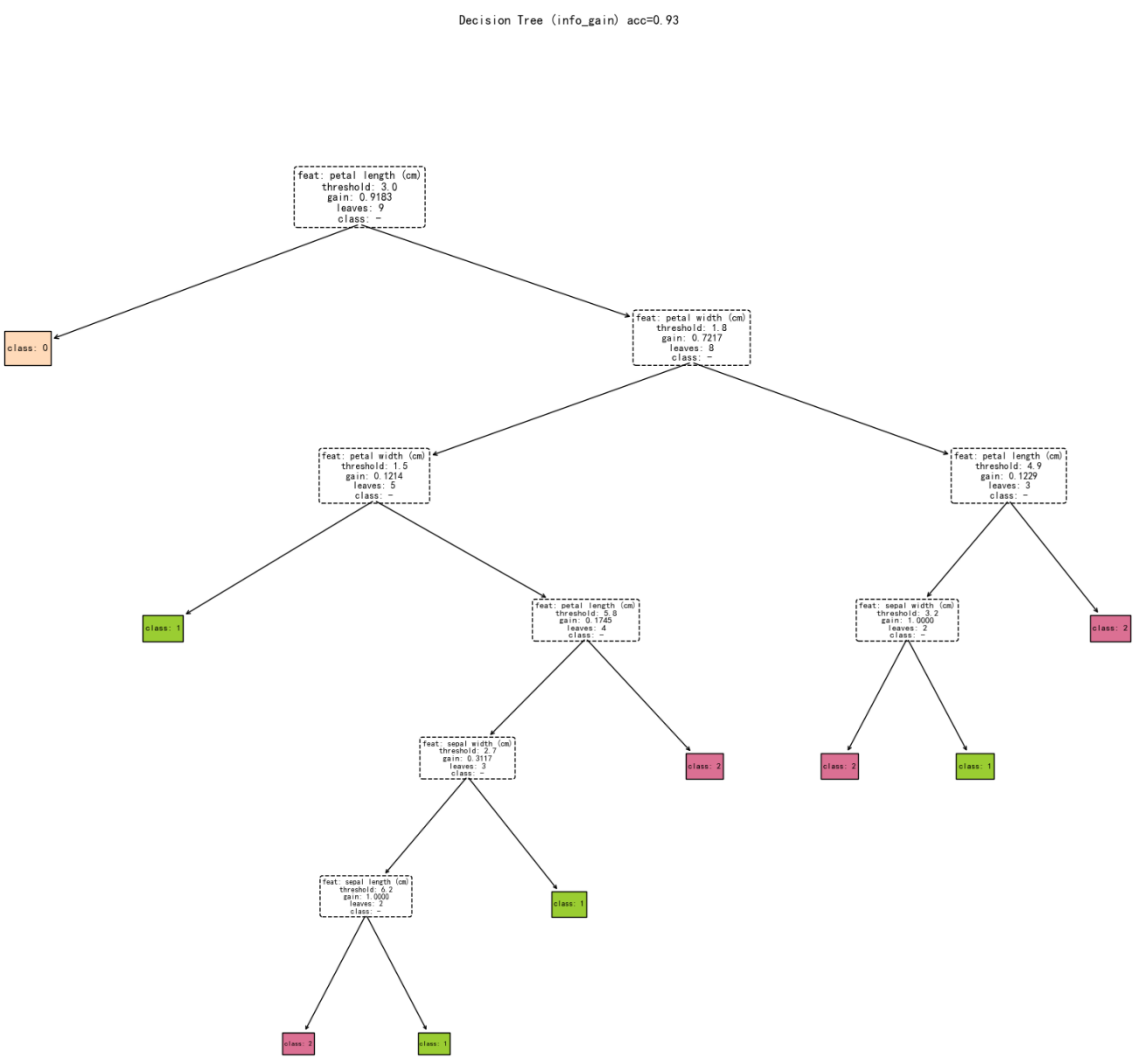
iris_error_rate

Accuracy = 0.9333 | Tree_depth = 4 | Tree_leaf_num = 7



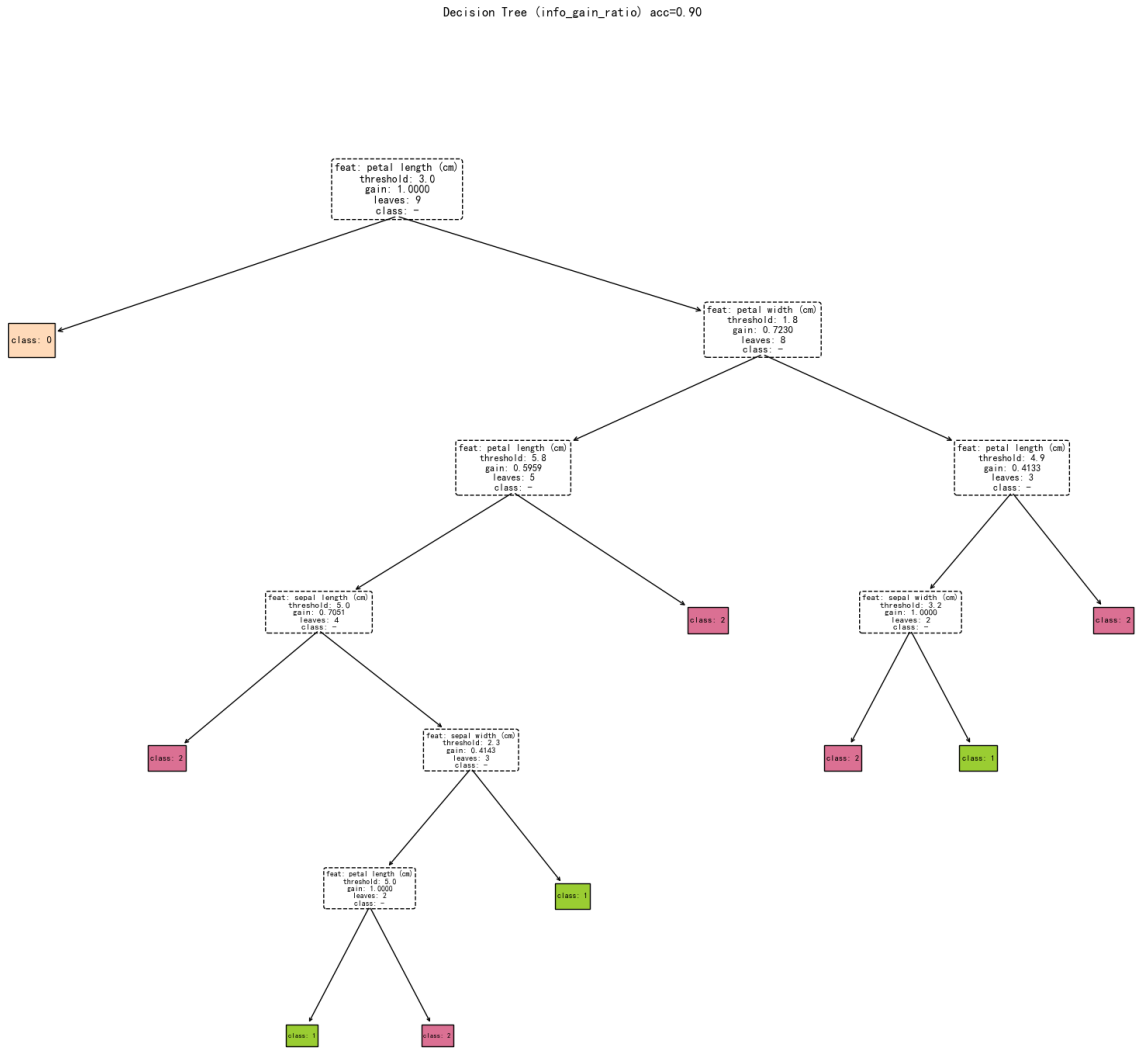
iris_gini

Accuracy = 0.9000 | Tree_depth = 6 | Tree_leaf_num = 9



iris_info_gain

Accuracy = 0.9333 | Tree_depth = 6 | Tree_leaf_num = 9



iris_info_gain_ratio

Accuracy = 0.9000 | Tree_depth = 6 | Tree_leaf_num = 9

4. 进一步探索 – 8%

本部分希望同学们在固定训练/验证/测试划分下，调参使测试集 Accuracy 尽可能高。

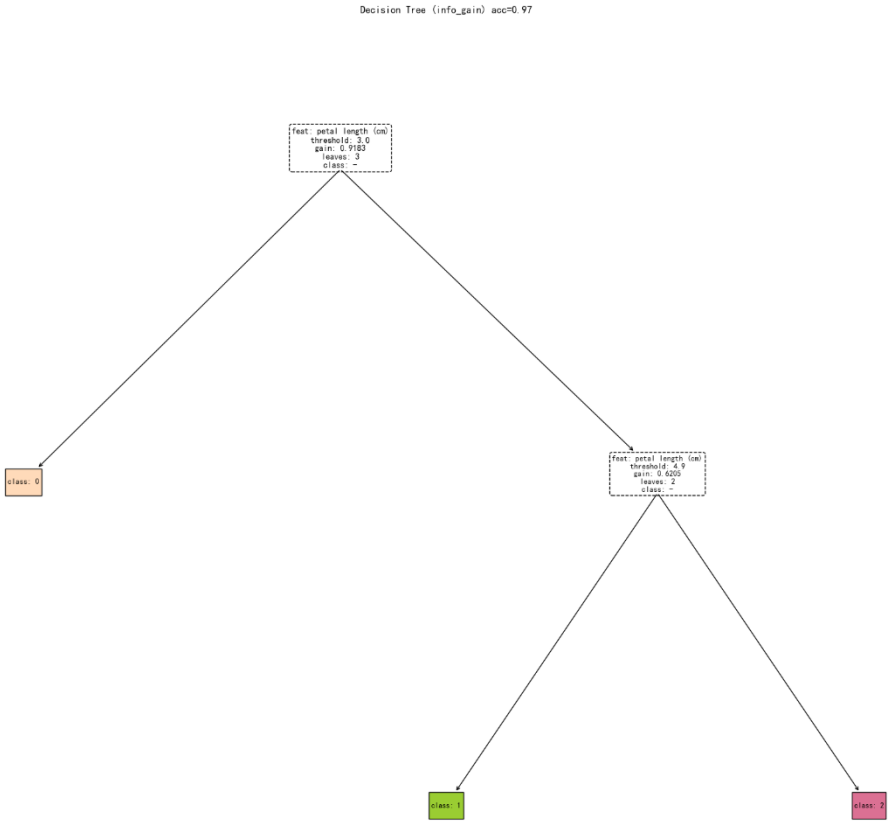
下表给出decision_tree.py中的可调参数

可调的参数	参数说明	可调值
criterion	分裂度量不同，偏好不同；可先粗选再细调	{info_gain, info_gain_ratio, gini, error_rate}
splitter	random 随机阈值更具多样性，配合多次 seed 取较稳的方案	{best, random}
max_depth	控制树深，限制过拟合	{None, 2-10}
min_samples_split	节点最小样本数，越大越保守	{2, 3, 4, 5, 10, 20.....}
min_impurity_split	最小分裂增益阈值，越高越保守	{0, 1e-4, 1e-3, 1e-2.....}
max_features	每次候选的特征子集大小，和随机森林思想类似	{ None, "sqrt", "log2", 1..d, 0.5..1.0}

你可以在下表中进行参数的尝试（至少3组），准确率**至少要95%以上**，越高越好

	1	2	3	4
criterion	info_gain	info_gain_ratio	gini	error_rate
splitter	best	best	best	best
max_depth	2	2	2	2
min_samples_split	2	2	2	2
min_impurity_split	0	0	0	0
max_features	1	1	1	1
accuracy	0.9667	0.9667	0.9667	0.9667

高亮你选中的最佳参数组合，在下面粘贴输出的树图



最近邻问题 – 40%

1. 简要解释下knn算法，以及其优缺点 – 2%

knn概述	优缺点
KNN是一种基于实例的监督学习算法。它在预测时直接根据样本之间的距离进行分类或回归：对一个测试样本，寻找训练集中距离最近的 K 个邻居，然后通过多数表决（分类任务）或平均值（回归任务）来得到预测结果	**优点：** 思路简单直观，无需显式训练过程； 能处理多分类问题和非线性边界； 适用于小规模数据集，效果较好； **缺点：** 计算复杂度高，预测时需要遍历训练集；

	需要合适的 K 值选择，过小容易过拟合，过大可能欠拟合； 对噪声和无关特征敏感，需要特征选择或归一化处理。

2. 代码实现（见Part3_ReadMe.md) – 15%

请在报告中贴出你实现的三个核心函数的代码并简单说明实现的逻辑：

pairwise_dist()

- L2 – twoloop

```
if metric == "l2":
    if mode == "two_loops":
        # ============== TODO (students, REQUIRED) ==============
        dists = np.zeros((Nte, Ntr), dtype=np.float64)
        for i in range(Nte):
            for j in range(Ntr):
                diff = X_test[i] - X_train[j]
                dists[i, j] = np.sqrt(np.dot(diff, diff))
        return dists
        # ========================================================
```

X_test：测试样本集合（Nte, D)

X_train：训练样本集合（Ntr, D)

Nte：测试样本数

Ntr：训练样本数

初始化一个（Nte, Ntr）矩阵存放所有测试点与训练点的距离

双循环：对每个测试点 i 和训练点 j，计算 欧几里得距离：

- L2 – noloop

```
elif mode == "no_loops":
    # ============== TODO (students, REQUIRED) ==============
    dists = np.zeros((Nte, Ntr), dtype=np.float64)
    X_test_sq = np.sum(X_test**2, axis=1).reshape(-1, 1)
    X_train_sq = np.sum(X_train**2, axis=1).reshape(1, -1)
    cross_term = np.dot(X_test, X_train.T)
    dists = np.sqrt(X_test_sq + X_train_sq - 2 * cross_term)
    return dists
    # ========================================================
```

向量化公式：

$$\|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2x \cdot y$$

通过矩阵运算避免双循环，加速计算

- Cosine

```
elif metric == "cosine":
    # ================ TODO (students, REQUIRED) ================
    X_test_norm = X_test / np.linalg.norm(X_test, axis=1, keepdims=True)
    X_train_norm = X_train / np.linalg.norm(X_train, axis=1, keepdims=True)
    cosine_sim = np.dot(X_test_norm, X_train_norm.T)
    dists = 1 - cosine_sim
    return dists
    # ===========================================================
```

np.linalg.norm(..., axis=1)：按行计算向量模长

余弦相似度：点乘/模长之积
转换为距离：1-cosine_sim

knn_predict()

```
    # ================ TODO (students, REQUIRED) ================
    counts = np.bincount(neighbors)
    y_pred[i] = np.argmax(counts)
    # ===========================================================
```

neighbors：测试样本 i 的 k 个最近邻训练样本标签
np.bincount(neighbors) 统计每个类别的出现次数
np.argmax(counts) 返回出现次数最多的类别，作为预测结果

select_k_by_validation()

```
# ================ TODO (students, REQUIRED) ================
accs = []
for k in ks:
    y_val_pred = knn_predict(X_val, X_train, y_train, k, metric, mode)
    accuracy = np.mean(y_val_pred == y_val)
    accs.append(accuracy)
best_k = ks[np.argmax(accs)]
return best_k, accs
# ===========================================================
```

ks：候选的 k 值列表
X_val, y_val：验证集数据及标签

遍历所有候选 k，调用 knn_predict 预测验证集，计算预测准确率
选择准确率最高的 k，作为最优 k。

3. 测试与结果可视化 – 15%

下面是我们在固定测试中使用的数据集参数设定，你可以在data_generate.py中查看和改变这些参数：

```
# ---- 数据规模与难度（可按需微调）----
RANDOM_STATE = 42       # 随机种子
N_SAMPLES    = 500      # 数据组数
N_CLASSES    = 4        # 希望有几类数据（在boundary图上能看到几个色块）
CLUSTER_STD  = 4        # 数值越大，数据点间越模糊，越不会形成明显的数据团
TEST_SIZE    = 0.25     # 测试集比例
VAL_SIZE     = 0.25     # 验证集比例
```

3.1

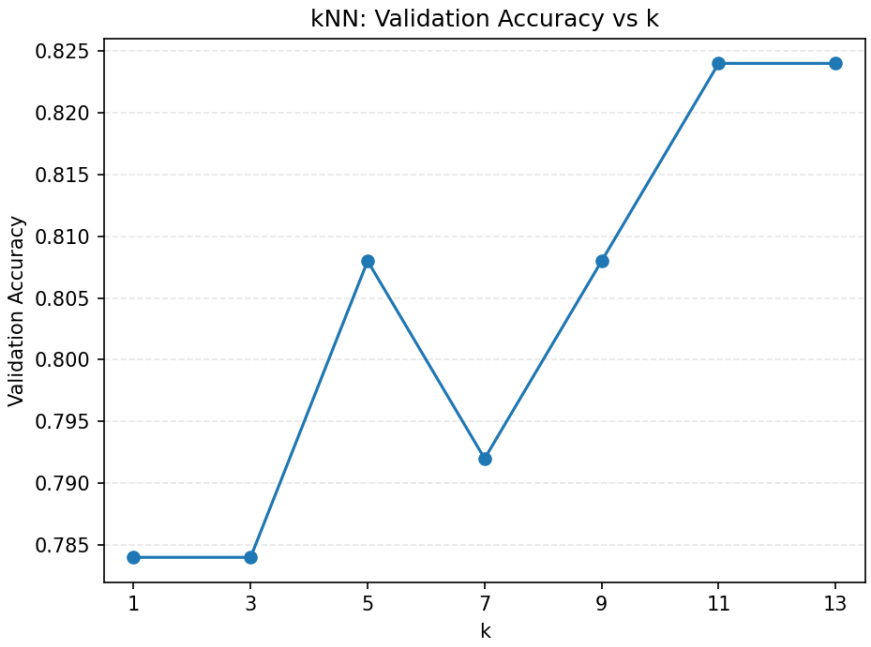
完成knn_student.py的代码块后，在data_generate.py中设置以上的参数，然后运行test_knn.py，把输出的log粘贴在下面，

```
======================================================================
实验: Part 3 - kNN 分类
姓名: 年骏杰    学号: 24307140012    时间: 2025-10-07 13:33:41
======================================================================
[DATA] Summary
 - train = 250, val = 125, test = 125, D = 2, classes = 4
   · train per-class: class 0: 62, class 1: 63, class 2: 62, class 3: 63
   · val per-class: class 0: 31, class 1: 31, class 2: 32, class 3: 31
   · test per-class: class 0: 32, class 1: 31, class 2: 31, class 3: 31
----------------------------------------------------------------------
[DIST] L2 two_loops vs no_loops
 - shape: (2, 4), max|diff| = 0.000e+00  -> PASS
[DIST] Cosine basic case
 - expected = [0.292893, 0.292893], got = [0.292893, 0.292893]  -> PASS
----------------------------------------------------------------------
[PRED] sanity checks (k=1 / k=3 / tie rule)
 - k=1 pred = [0, 1, 0]  (expect [0,1,0])  -> PASS
 - k=3 pred = [0, 1, 0]  (expect [0,1,0])  -> PASS
 - tie case (k=4) -> pred = 0 (expect 0) -> PASS
----------------------------------------------------------------------
[MODEL SELECTION] validation curve
 - ks & val_accs: k=1:0.7840, k=3:0.7840, k=5:0.8080, k=7:0.7920, k=9:0.8080, k=11:0.8240, k=13:0.8240
 - best_k = 11 (val_acc = 0.8240)
----------------------------------------------------------------------
[E2E] train+val -> test (metric=l2, mode=no_loops)
 - test_acc(best_k=11) = 0.8080
======================================================================
All kNN tests passed.
```

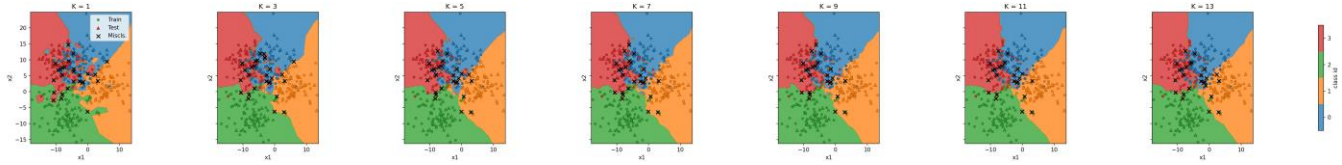
3.2

当所有的测试都通过后，运行knn_student.py，程序会调用viz_knn.py中的可视化函数，输出knn_k_curve.png和knn_boundary_grid.png两个图像。需要你在knn_student.py中修改matric参数，分别生成使用‘L2’和‘cosine’的图像，贴在下面。

a. Matric = ‘L2’

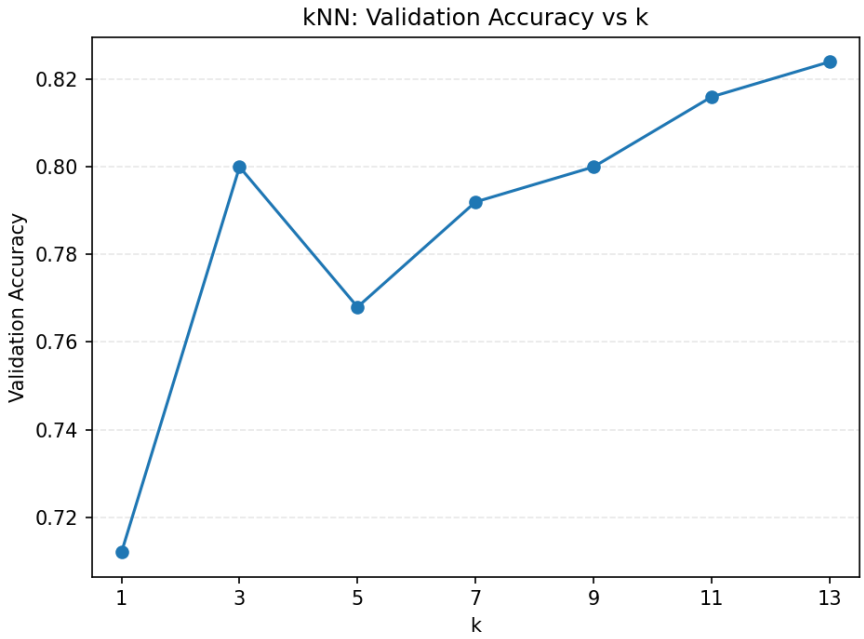


knn_k_curve

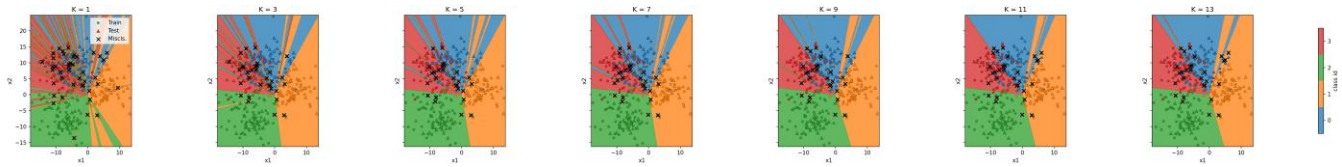


knn_boundary_grid.

b. Metric = ‘cosine’



knn_k_curve



knn_boundary_grid.

3.3

观察你得到的测试报告和图像，回答以下的问题：

a. 准确率最高的k值是多少?

	Best_k	Accuracy
L2	11	0.8240
Cosine	13	0.8240

b. K 对边界的影响，K=1 时边界为何“锯齿/细碎”？K 增大为何更平滑?

K = 1时：每个测试点的分类由最近的一个训练样本决定。会导致边界完全由训练数据的分布决定，每个训练点都会占据一个区域，哪怕是数据中的噪声点、孤立点，也会形成自己的一块区域。因此边界呈现出非常锯齿状、细碎的形态。

K增大时：预测类别由K个最近邻投票决定，单个孤立点的影响被削弱，必须要有更多邻居支持才能左右分类结果。分类结果更能体现整体分布趋势，而不是单个样本。边界因此变得更平滑，不会紧贴训练数据的细节。

c. 在相同的数据下‘L2’和‘cosine’有什么差异（结合图像解释）？他们各自测量的‘距离’是什么？

差异：L2聚类结果形状更接近“圆形/椭圆形”区域。边界呈现较平滑，稍带弯曲的径向划分。Cosine聚类结果呈现扇形/辐射状分布，能看到很多放射状边界。

L2测量真实的欧氏距离，cosine关心方向，忽略模长，不同类别被划分成角度区域，而不是按距离远近划分。

4. 进一步探索 - 8%

本部分希望同学们能够选择自己感兴趣的问题进行探索，并完成一份简单的实验报告，我们提供三个样例问题，同学们可以选择其中之一进行探索，更加鼓励自己寻找一个问题进行实验。

样例问题1：探索适合‘L2’和‘Cosine’的数据场景
通过修改data_generate.py中的数据参数，和k的选择，分别搜索能够在‘L2’和‘Cosine’方法下达到高准确度(95%以上)的数据集，分析两种距离计算方式适配的场景和数据结构。

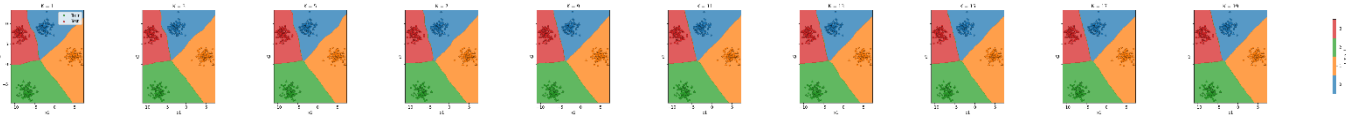
样例问题2：类内方差（重叠程度）对 k 的影响
通过修改data_generate.py中的CLUSTER_STD参数，探索其和k的联系。可以通过下面的问题展开：
CLUSTER_STD ↑（更模糊）时，best_k 是否趋向更大?
为什么从“锯齿→平滑”的边界有助于抗噪?
对比 k=1 与 k=best_k 的误分类点分布（图表 ×），哪些区域最难?

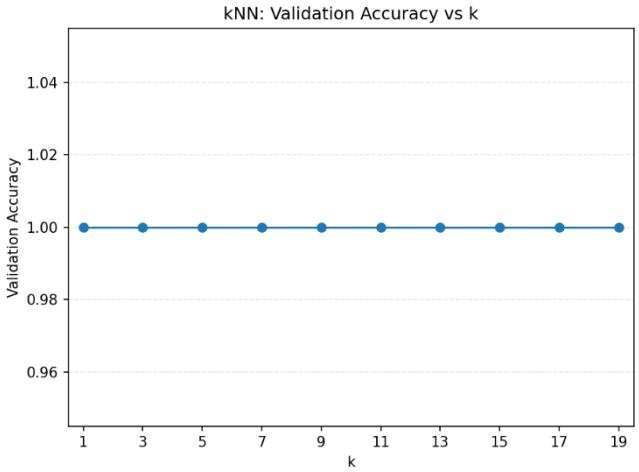
样例问题3：探索数据结构与过拟合/欠拟合的关系
过拟合/欠拟合的定义以及他们呈现的结果是什么样的?
什么样的参数会导致过拟合/欠拟合的发生?

探索问题:
其他条件不变的情况下，类内方差（重叠程度）对 best_k 的影响

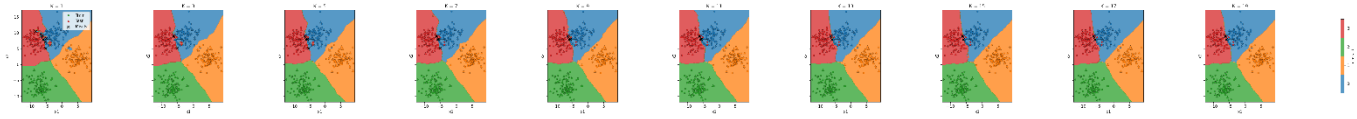
探索方法:
修改CLUSTER_STD，从1-10，步长为1。将待选的k调整至1-19，步长为2，并将模式设为默认L2 no loop. 然后找出不同类内标准差生成的数据所对应的best_k，并比较不同k的决策边界图像。由于CLUSTER_STD大于10时，几个类别之间重合过大，使用KNN已经难以达成区分效果，因此不在考虑范围。

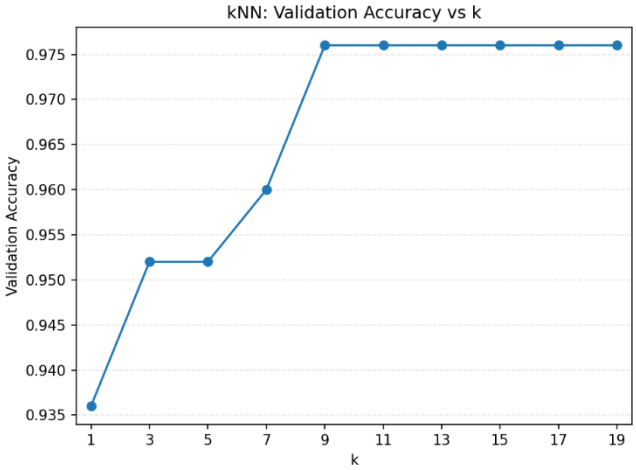
输出结果:
CLUSTER_STD=1, best_k=1,3,...19, val_acc=1



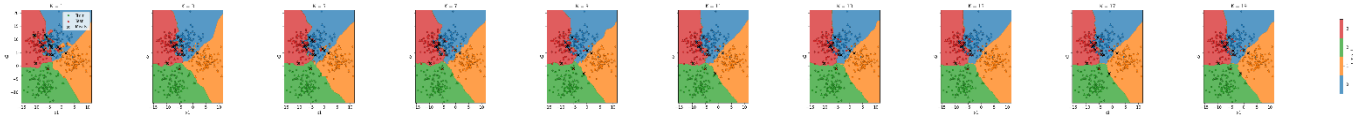


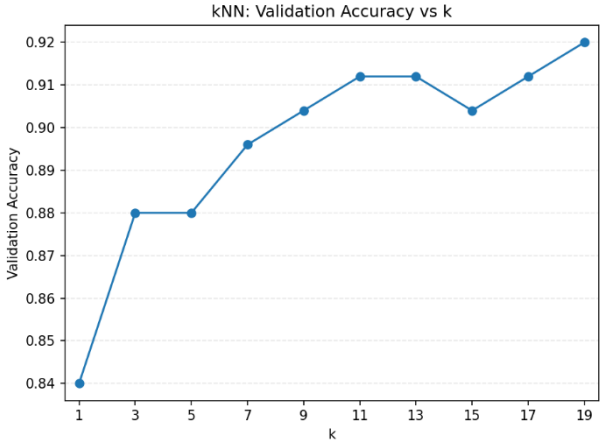
CLUSTER_STD=2, best_k=9,11,..,19, val_acc=0.975



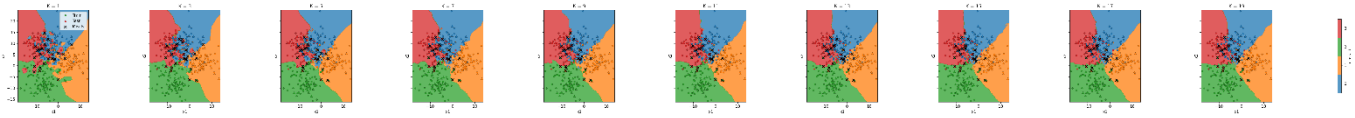


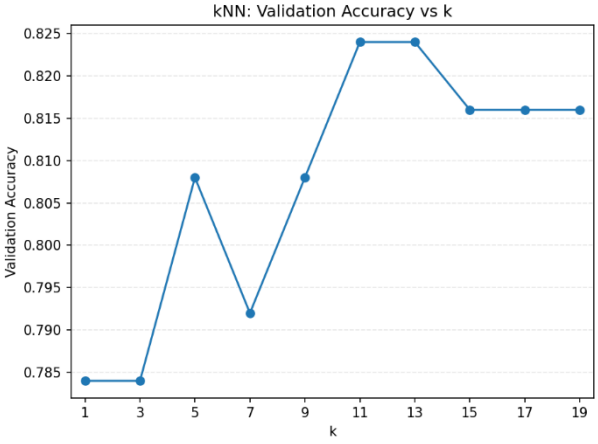
CLUSTER_STD=3, best_k=19, val_acc=0.92



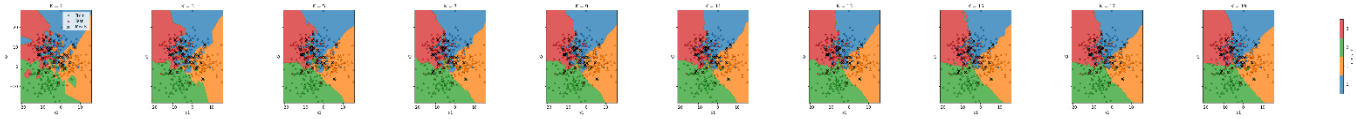


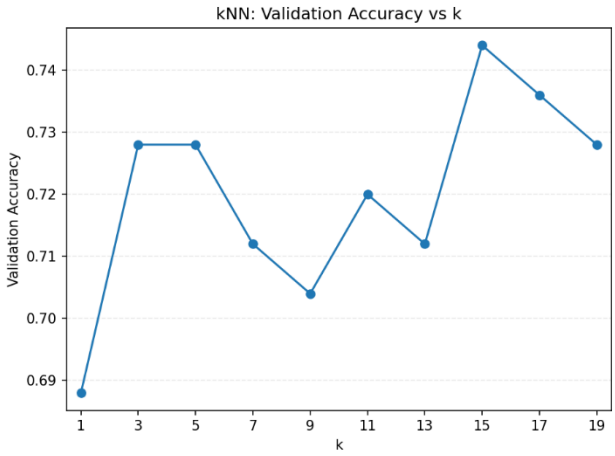
CLUSTER_STD=4, best_k=11, val_acc=0.824



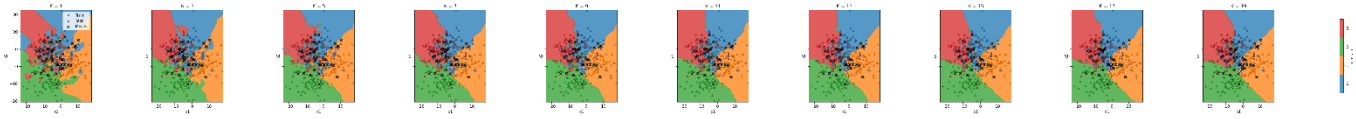


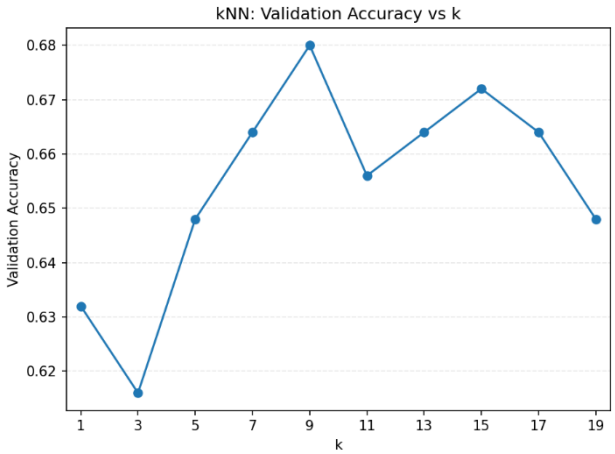
CLUSTER_STD=5, best_k=15, val_acc=0.744



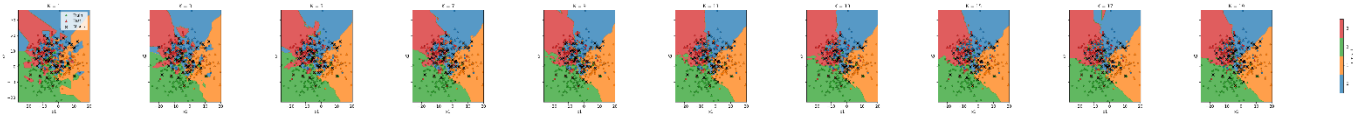


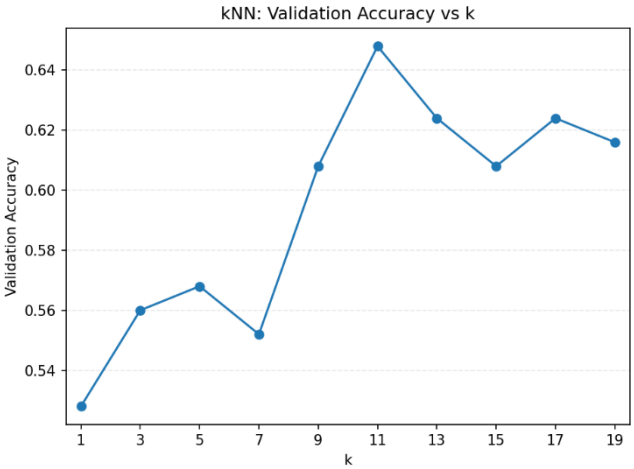
CLUSTER_STD=6, best_k=9, val_acc=0.68



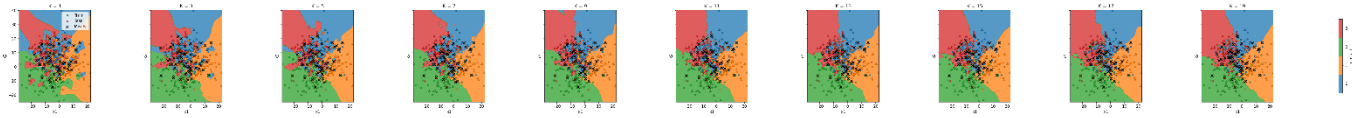


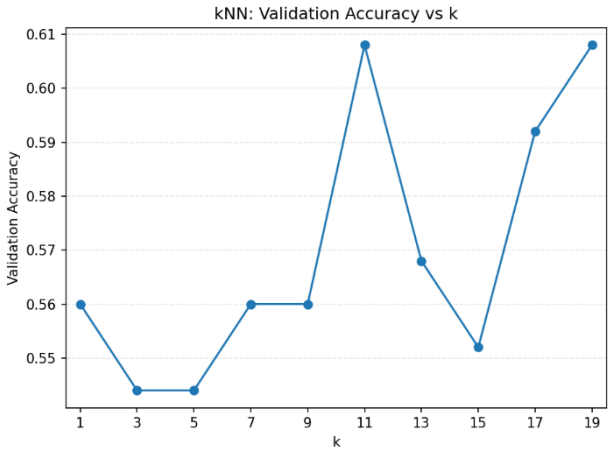
CLUSTER_STD=7, best_k=11, val_acc=0.648



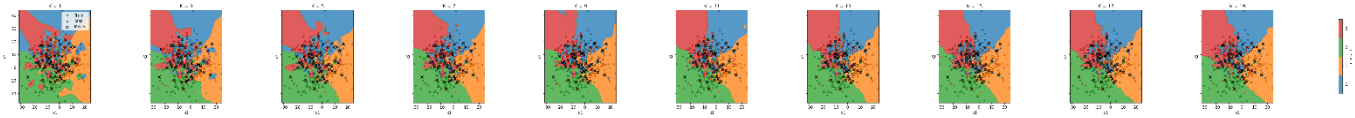


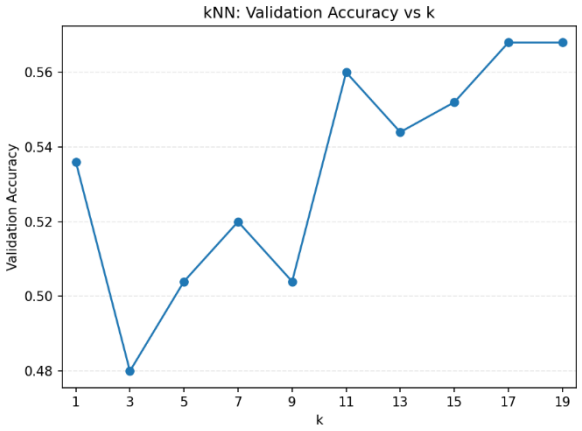
CLUSTER_STD=8, best_k=11, val_acc=0.608



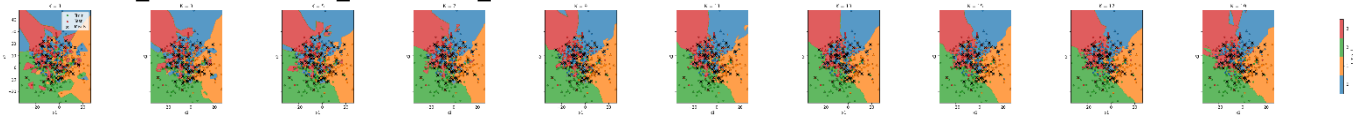


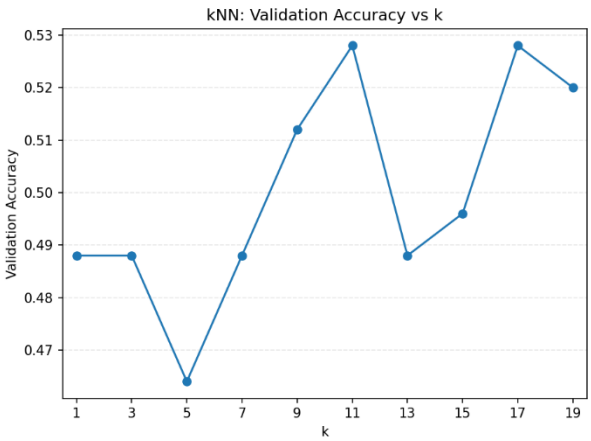
CLUSTER_STD=9, best_k=17, val_acc=0.568





CLUSTER_STD=10, best_k=11, val_acc=0.528





结果分析:

CLUSTER_STD ↑（更模糊）时，best_k 是否趋向更大?

可以看出，当CLUSTER_STD较小时，如1,2,3时，best_k随其增长而增长，而且一旦k>=best_k 时，预测结果一直保持最佳val_acc；然而当CLUSTER_STD逐渐变大至4以上时，best_k一直维持在11左右，在大部分时候，k大于或小于best_k都会导致准确率下降，决策边界也开始变得扭曲，甚至出现区块割裂，如CLUSTER_STD>=8时。

为什么从“锯齿→平滑”的边界有助于抗噪?

锯齿：每个点都能撑起一个类别区域，因此噪声点也能被单独划分出一个类别区域。

平滑：要有足够多的同类邻居才能改变结果，噪声点不容易左右局部分类。

对比k=1与k=best_k的误分类点分布（图表 ×），哪些区域最难?

最难在几个类别重合的地方。因为类内标准差过高，导致坐标轴中间区域重叠许多不同类型的数据点，k=1只考虑重叠区域点内最附近的点的类别，会导致很高的错误率，而且导致决策边界出现割裂。而k_best采用更加平滑的边界，抵抗重叠区域内的数据点的干扰，能够得到相较k=1时更好的分类结果。然而由于有重叠部分，因此数据本身的分类边界就是扭曲锯齿状的，因此k_best预测出的光滑边界也并不能非常好的反应原有的数据类别，这也为什么当类内标准差很高且k>=k_best时，更光滑的边界会导致更低的准确率，此时KNN就不能很好完成分类任务了。

提交

- 完成后删除所有红色字体的提示部分，不要改动黑色字体的题干部分
- 选择合适的字体和行间距，保证美观和可读性
- 保证粘贴的图像大小合适，图中内容清晰可见
- 完成后导出为**pdf**，把文件名改为 **PRML-实验1-姓名** ，提交到elearning上，**不需要**提交单独的图像，代码文件或压缩包
- 截至日期在开头