模式识别与机器学习 – 实验报告 1

姓名: 吴卓阳 | 学号:23307130392 | 专业：计算机科学与技术 | 学院：计算与智能创新学院

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	模型预测正确的样本数占总样本数的比例	适用于类别分布比较均衡时
MSE	预测值与真实值之差的平方的平均值	多用于回归问题
Precision	模型预测为正的样本数中，真正为正样本的比例	当“误报”的代价很高时
Recall	所有真正的正样本中，被准确预测为正样本的比例	当“漏报”的代价很高时
F1	精确率和召回率的调和平均	需要综合考虑精确率和召回率

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()

```
    correct = 0
    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            correct += 1

    return correct / len(y_true)
```

通过correct来一一对比统计预测正确的样本数，返回所占比例

mean_squared_error()

```
    total_error = 0
    for i in range(len(y_true)):
        diff = y_true[i] - y_pred[i]
        total_error += diff ** 2

    return total_error / len(y_true)
```

——计算差值并平方，累加到total_error之中，返回平均数

precision_score()

```
    return tp / (tp + fp)
```

tp+fp为所有预测为正的样本数，返回其中tp即真正为正样本数的比例

recall_score()

```
    return tp / (tp + fn)
```

tp+fn为所有真正的正样本的数量，返回其中tp即被正确预测为正样本的比例

f1_score()

```
    return 2 * precision_score(y_true, y_pred) * recall_score(y_true, y_pred) / (precision_score(y_true, y_pred) +
recall_score(y_true, y_pred))
```

计算精确率和召回率的调和平均

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：

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
0.67	0.6667	0.60	0.75	0.67

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进行决策的模型，通过不断对特征进行条件判断，把样本划分成不同的类别或预测值；每个内部节点表示一次特征判断，每个叶子节点表示预测结果。	优点：不需要太多数据预处理，测试速度快，可解释性强 缺点：每次划分仅考虑单个属性，单个决策树准确率往往不高

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(...)

```
def entropy(label_dict):
    total = sum(label_dict.values())
    ent = 0.0
    for c in label_dict.values():
        p = c / total
        ent -= p * math.log2(p)
    return ent


total = len(y)
left_total = len(l_y)
right_total = len(r_y)


before = entropy(all_labels)
after = (left_total / total) * entropy(left_labels) + (right_total / total) * entropy(right_labels)
info_gain = before - after
```

先定义函数entropy，按照公式计算单个节点的信息熵；再分别计算父节点的信息熵、加权计算分裂为子节点之后的信息熵，按算法计算二者差值

- __info_gain_ratio(...)

```
def split_info(l_y, r_y):
    total = len(l_y) + len(r_y)
    left_total = len(l_y)
    right_total = len(r_y)
    split_info = 0.0
    for part in [left_total, right_total]:
        if part == 0:
            continue
        p = part / total
        split_info -= p * math.log2(p)
    return split_info


split_information = split_info(l_y, r_y)
if split_information == 0:
    return 0


info_gain /= split_information
```

定义函数split_info，计算左右子集样本比例的熵（即划分本身的不确定性）；再按照算法计算info_gain

- __gini_index(...)

```
def gini(label_dict):
    total = sum(label_dict.values())
    gini = 1.0
    for c in label_dict.values():
        p = c / total
        gini -= p ** 2
    return gini


total = len(y)
left_total = len(l_y)
right_total = len(r_y)


before = gini(all_labels)
after = (left_total / total) * gini(left_labels) + (right_total / total) * gini(right_labels)
```

定义函数gini，计算各个标签频率的平方的和的负数，按公式计算基尼系数；再分别计算父节点的基尼系数和加权计算子节点的基尼系数

- __error_rate(...)

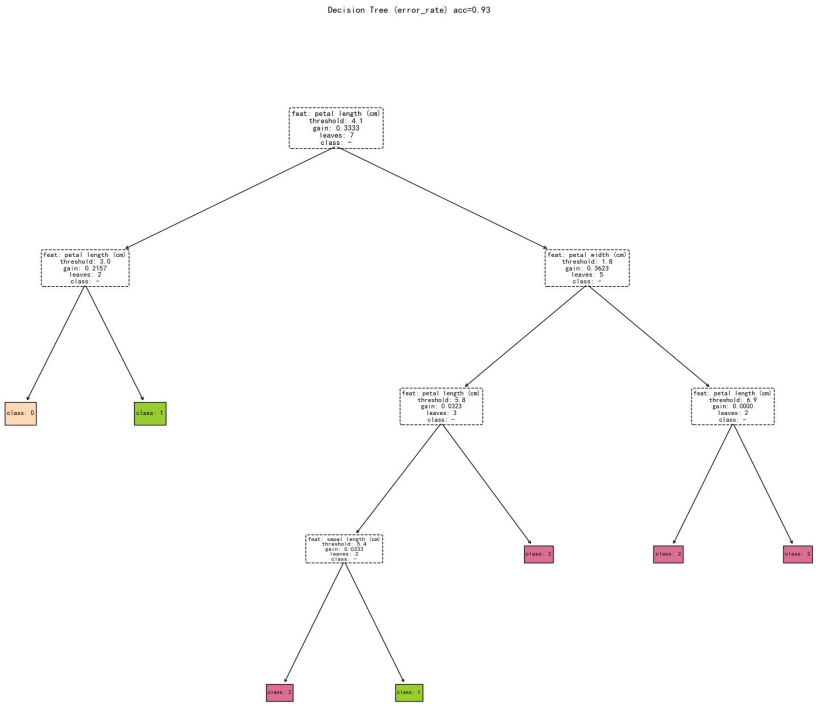
```
def error_rate(label_dict):
    total = sum(label_dict.values())
    if total == 0:
        return 0.0
    max_count = max(label_dict.values())
```

```
    return 1.0 - (max_count / total)


total = len(y)
left_total = len(l_y)
right_total = len(r_y)


before = error_rate(all_labels)
after = (left_total / total) * error_rate(left_labels) + (right_total / total) * error_rate(right_labels)
```

定义函数error_rate，找出最大标签频率，按公式计算误分类率；再分别计算父节点和加权计算子节点的误分类率

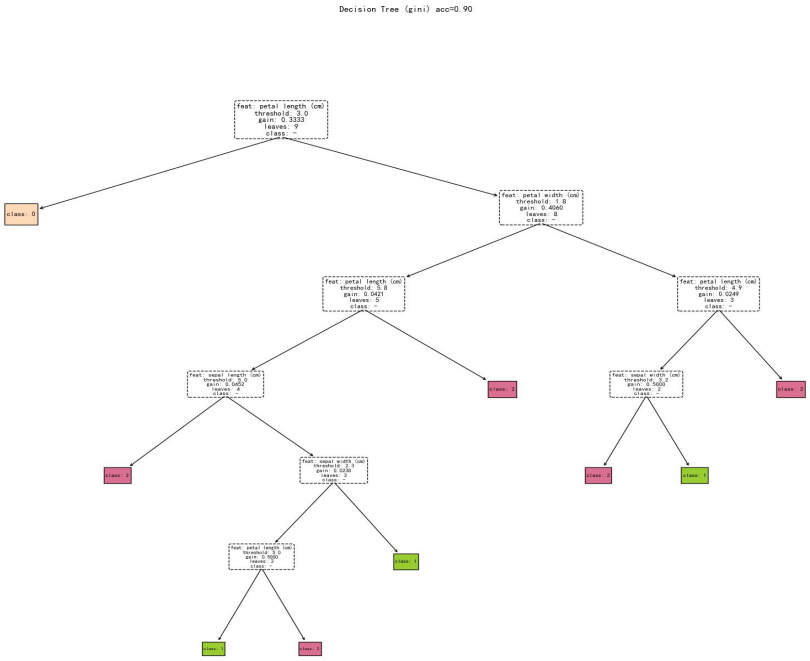
3. 测试和可视化 – 15%

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**



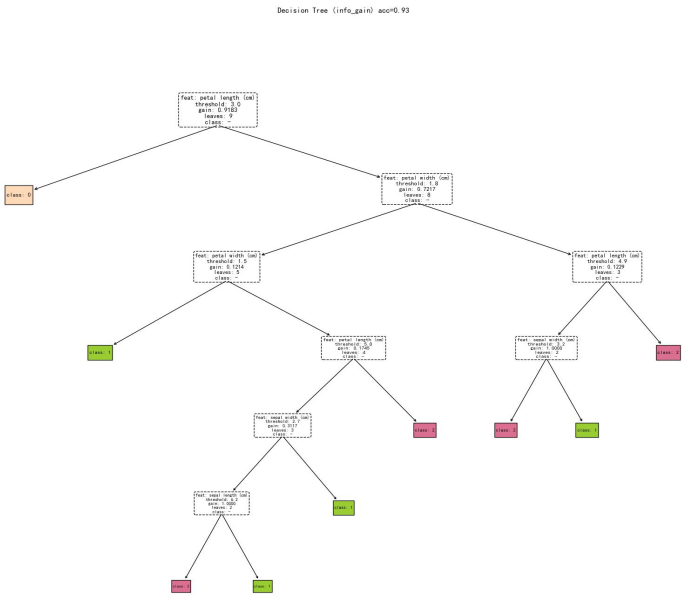
iris_error_rate

Accuracy = 0.933 | Tree_depth =5 | Tree_leaf_num = 7



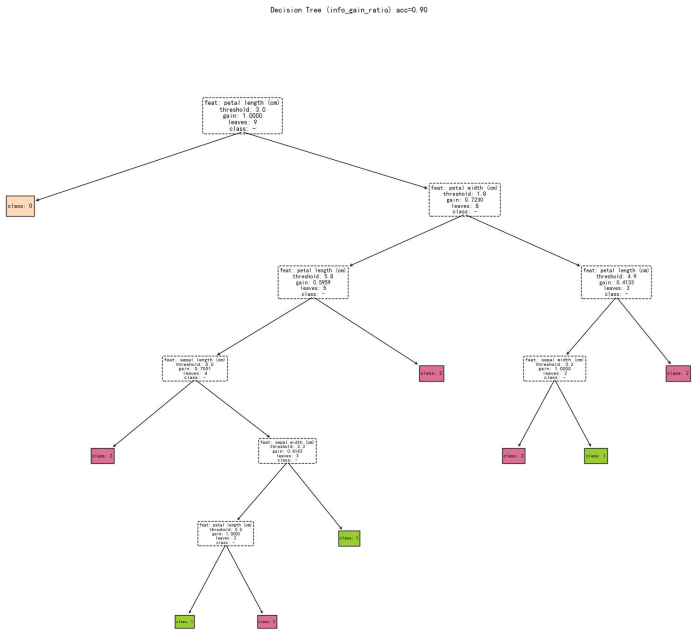
iris_gini

Accuracy =0.9000 | Tree_depth =7 | Tree_leaf_num = 9



iris_info_gain

Accuracy = 0.9333 | Tree_depth = 7 | Tree_leaf_num = 9



iris_info_gain_ratio

Accuracy = 0.9000 | Tree_depth = 7 | 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。

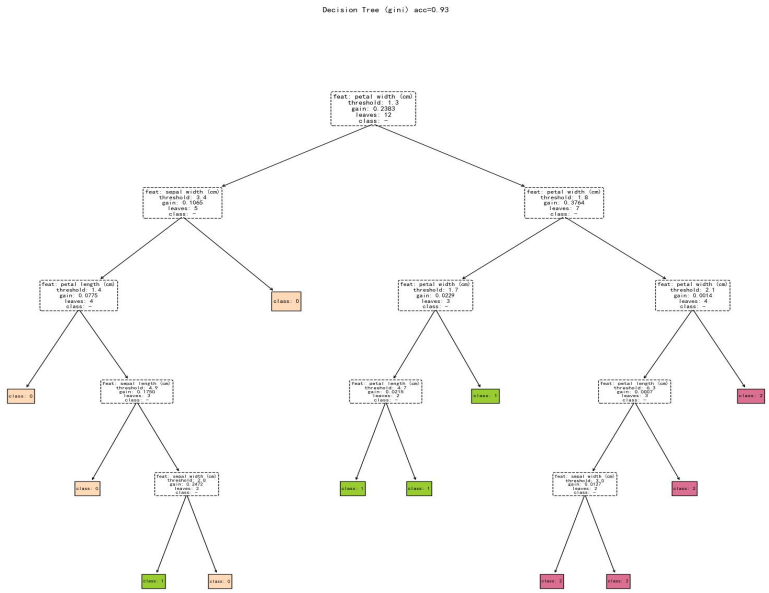
下表给出decision_tree.py中的可调参数

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	gini	gini	gini	
splitter	random	random	random	
max_depth	6	6	6	
min_samples_split	20	10	15	
min_impurity_split	0.0001	0.0001	0.0001	
max_features	None	None	None	
accuracy	0.9000	0.9000	0.9333	

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
给定一个待分类样本，算法计算它与训练集中所有样本的距离，选出距离最近的 K 个样本，根据多数投票（分类）或平均值（回归）来决定该样本的预测结果。	优点：思想简单、无需训练，效果有时会优于一些复杂的分类器 缺点：测试时速度很慢、会产生维度灾难

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

```
dists = np.zeros((Nte, Ntr))
for i in range(Nte):
    for j in range(Ntr):
        dists[i, j] = np.linalg.norm(X_test[i] - X_train[j])
return dists
```

创建一个形状为（Nte, Ntr）的 NumPy 数组 dists，并初始化为全零；通过双循环遍历每个测试样本索引和训练样本索引，再调用函数计算二者之间的l2距离，存储在dists[i,j]中

- L2 – noloop

```
sq_test = np.sum(X_test ** 2, axis=1)[:, np.newaxis]
sq_train = np.sum(X_train ** 2, axis=1)[np.newaxis, :]
dists = np.sqrt(sq_test + sq_train - 2 * np.dot(X_test, X_train.T))
return dists
```

先对 X_test 矩阵中的每个元素进行平方运算，再沿着特征维度（axis=1）对每个测试样本的平方和求和，得到一个形状为（Nte,）的向量，表示每个测试样本的平方和，将结果转换为列向量，形状变为（Nte,

1)，以便后续广播操作。类似的处理 X_train 矩阵。之后将 sq_test（(Nte, 1)）和 sq_train（(1, Ntr)）通过 NumPy 的广播机制相加，生成一个（Nte, Ntr）的矩阵，其中每个元素是对应测试样本和训练样本的平方和之和。再计算 X_test（(Nte, D)）和 X_train 的转置（(D, Ntr)）的矩阵乘法，得到（Nte, Ntr）的矩阵，表示所有测试样本和训练样本之间的点积并乘以 2，这是 L2 距离公式的分母部分；根据 L2 距离的向量化公式，计算每个测试样本和训练样本之间的平方距离，再对结果取平方根，得到最终的 L2 距离矩阵（Nte, Ntr)。

- Cosine

```
    norms_test = np.sqrt(np.sum(X_test ** 2, axis=1))[:, np.newaxis]
    norms_train = np.sqrt(np.sum(X_train ** 2, axis=1))[np.newaxis, :]
    sim = np.dot(X_test, X_train.T) / (norms_test * norms_train)
    dists = 1 - sim
    return dists
```

对 X_test 矩阵中的每个元素求平方，沿着特征维度（axis=1）对每个测试样本的平方和求和，得到一个形状为（Nte,）的向量，表示每个测试样本的平方和，对每个平方和取平方根，计算每个测试样本的 L2 范数（欧几里得范数），结果仍为（Nte,)，将结果转换为列向量，形状变为（Nte, 1)，对 X_train 进行类似操作。计算 X_test（(Nte, D)）和 X_train 的转置（(D, Ntr)）的矩阵乘法，得到（Nte, Ntr）的矩阵，表示所有测试样本和训练样本之间的点积；通过广播机制，norms_test ((Nte, 1)) 和 norms_train ((1, Ntr)）相乘，生成（Nte, Ntr）的矩阵，其中每个元素是对应测试样本和训练样本的 L2 范数乘积。再将点积除以范数乘积，计算余弦相似度。

knn_predict()

```
    unique, counts = np.unique(neighbors, return_counts=True)
    max_count = np.max(counts)
    candidates = unique[counts == max_count]
    y_pred[i] = np.min(candidates)
```

先对neighbors中的唯一标签进行统计，返回每个唯一标签出现的次数，找到其最大值，并从中选择一个预测结果（若平票，则选择标签值最小的类别作为预测结果）

select_k_by_validation()

```
accs = []
for k in ks:
    y_pred_val = knn_predict(X_val, X_train, y_train, k, metric, mode)
    acc = np.mean(y_pred_val == y_val)
    accs.append(acc)
best_k = ks[np.argmax(accs)]
```

```
return best_k, accs
```

先初始化一个空列表accs用于存储每个k值对应的验证准确率，再遍历每个k值，调用函数进行预测并返回预测结果，比较预测结果与真实标签并计算准确率，将其存入列表accs中，最后找到最大准确率作为索引获取对应的k值

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

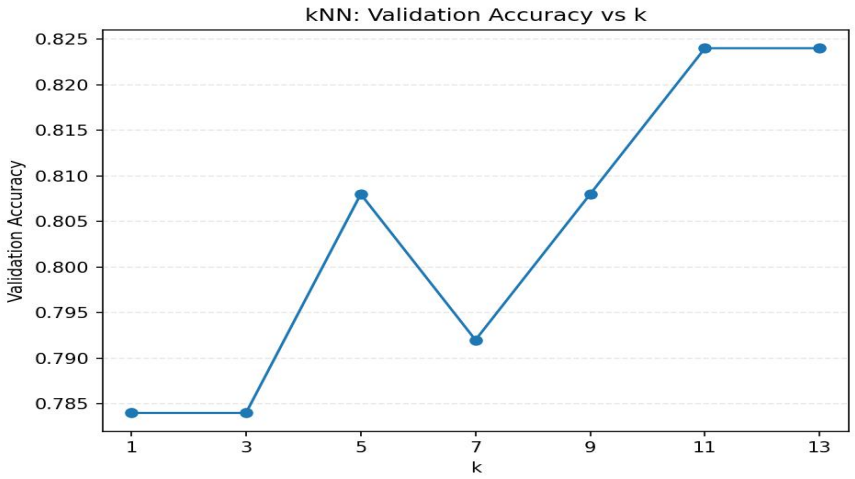
完成knn_student.py的代码块后，在data_generate.py中设置以上的参数，然后运行test_knn.py，把输出的log粘贴在下面，

```
实验：Part 3 - kNN 分类
姓名：吴卓阳    学号：23307130392    时间：2025-10-10 17:19:10
======================================================================
[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.
```

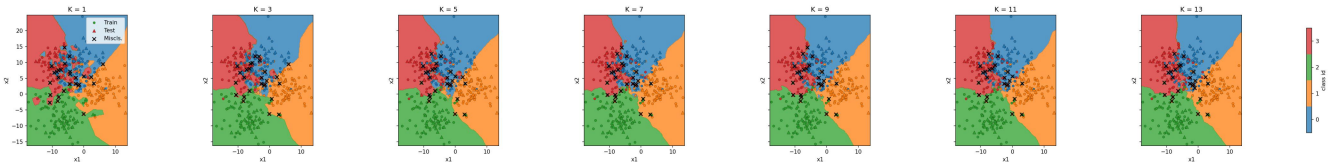
3.2

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’

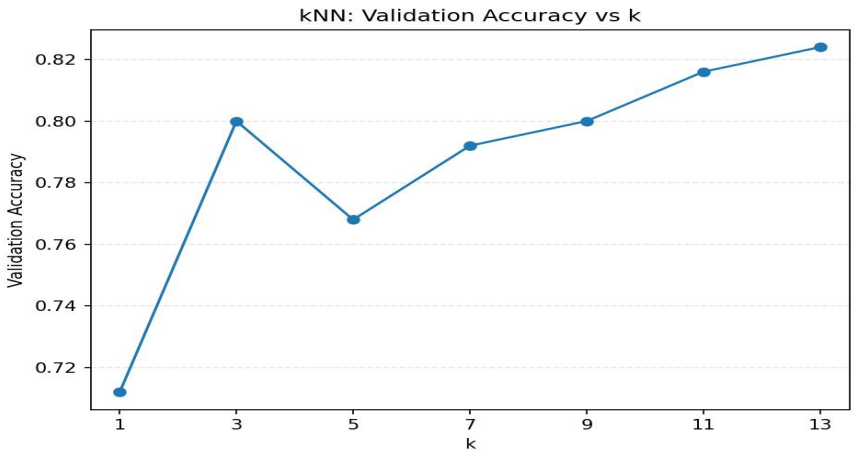


knn_k_curve

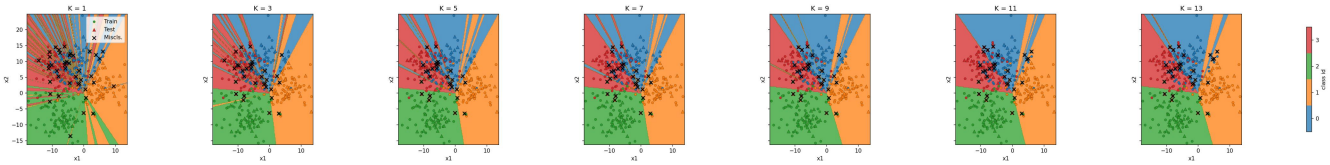


knn_boundary_grid.

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、13	0.824
Cosine	13	0.824

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?

当 K=1 时，kNN 算法只考虑每个测试点最近的一个训练点来决定其类别，边界会非常敏感地反映训练数据点的具体位置。即使训练数据中存在噪声或孤立点，边界也会围绕这些点产生尖锐的、锯齿状的变化，分类决策完全依赖局部信息，缺乏平滑或全局的趋势。

当 K 增大时，kNN 算法考虑更多邻近点的多数表决，较大的 K 值使得分类决策基于更广泛的区域信息，而不是单一点的影响，局部噪声或异常点的影响被平均化，分类结果更能反映数据整体的分布趋势。

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?

L2 距离：在 K=1 时，验证准确率较低（约 0.785），随着 K 增加，准确率逐步上升，到 K=13 时达到 0.824。边界图显示，当 K 较小时，边界较为复杂且细碎；K 增大后，边界变得更平滑。L2 距离对数据点的绝对位置和尺度敏感，因此在数据分布较分散或存在显著尺度差异时，边界可能更复杂。

Cosine 距离：初始准确率 (K=1) 较低 (约 0.72)，但随着 K 增加，准确率迅速上升，到 K=13 时达到 0.822。边界图显示，Cosine 距离的边界在小 K 值时也较细碎，但随着 K 增大，平滑趋势与 L2 类似。Cosine 距离更关注向量方向而非绝对大小，因此对数据点的尺度不敏感，可能导致在某些 K 值下（例如 K=5）准确率出现波动。

L2 距离：也称为欧几里得距离，测量两个点在欧几里得空间中的直线距离。
Cosine 距离：基于余弦相似度的度量，它测量两个向量之间的夹角余弦值（方向相似性）

4. 进一步探索 - 8%

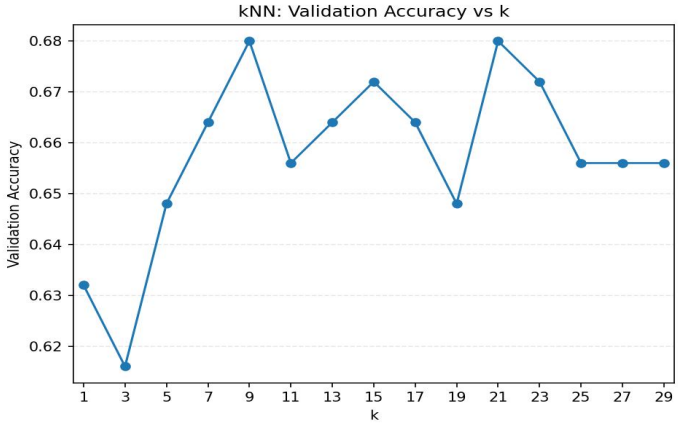
本部分希望同学们能够选择自己感兴趣的问题进行探索，并完成一份简单的实验报告，我们提供三个样例问题，同学们可以选择其中之一进行探索，更加鼓励自己寻找一个问题进行实验。

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。

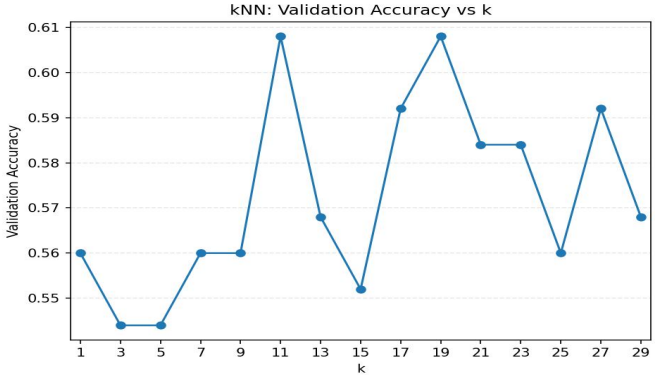
样例问题2：类内方差（重叠程度）对 k 的影响
通过修改data_generate.py中的CLUSTER_STD参数，探索其和k的联系。可以通过下面的问题展开：
CLUSTER_STD ↑（更模糊）时，best_k 是否趋向更大?
为什么从 “锯齿→平滑” 的边界有助于抗噪?
对比 k=1 与 k=best_k 的误分类点分布（图表 ×），哪些区域最难?

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?

对于样例问题二：先将k的取值范围拓宽到29
首先修改参数CLUSTER_STD为6.0（增大了2.0），使用l2距离进行测试，发现best k变为9：



再将参数修改为8.0（增大了4.0），仍旧使用l2距离进行测试，发现best k变为11：



因此CLUSTER_STD ↑（更模糊）时，best k不一定趋向于更大

为什么“锯齿→平滑”的边界有助于抗噪？ “锯齿”边界（k=1）仅依赖单个最近邻，易受噪声或孤立点影响，导致边界不稳定.“平滑”边界（较大k）通过多个邻居的投票，平均化了局部噪声的影响，使分类决策更合理、更全面，尤其在重叠或噪声较大的区域效果更明显。

对比 k=1 与 k= best k 的误分类点分布：

k=1时，误分类点集中在类间重叠区域和边界附近，反映其对噪声和重叠敏感。

k=best k时，误分类点减少，分布更均匀，一般边界附近误分类点减少（边界更加平滑，受噪声影响较小），但高重叠区域仍然难以十分准确地分类。

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