模式识别与机器学习 – 实验报告 1

姓名：王镜凯 学号: 22300240022 专业：计拔 学院：CS

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/总样本	简单直观，适用于**类别平衡**的分类任务
MSE	均方误差	对误差敏感，大误差会被放大，回归常用
Precision	TP/TP+FP 预测为正的样本中，有多少是真的正的。	能衡量“预测为正”的可信度
Recall	实际为正的样本中，有多少被成功预测出来。	衡量模型“捕捉正样本”的能力
F1	Precision 和 Recall 的 调和平均数	综合考虑了 Precision 和 Recall 对类别不平衡问题更加稳健

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()

```
accuracy = np.mean(y_true == y_pred)
```

- 求准确率

mean_squared_error()

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

precision_score()

```
if tp + fp == 0:
return 0.0 # 避免除零错误
return tp / (tp + fp)
```

recall_score()

```
if tp + fn == 0:
return 0.0 # 避免除零错误
return tp / (tp + fn)
```

f1_score()

```
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
if precision + recall == 0:
return 0.0 # 避免除零错误
return 2 * (precision * recall) / (precision + recall)
```

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：

```
(/Users/Zhuanz/Documents/Senior/PRML/.conda) Zhuanz@MacBook-Air-9 classification % python test.py
============================================================
实验: Part 1 - 分类评测指标
姓名: 王镜凯    学号: 22300240022
时间: 2025-10-10 09:03:35
============================================================
[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
```

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
2/3	1/3	0.6	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
决策树是一种监督学习模型，可用于分类和回归任务。它的结构就像一棵“树”，通过一系列“是 / 否”的问题，逐步划分数据，直到达到分类或回归的目标。	优点 简单直观，易于理解和解释 训练完成后可以画成一棵树，非常适合可视化。 不像神经网络那样是“黑箱”。 不需要特征标准化 不用做归一化、标准化，对特征分布要求低。 可以处理数值型和类别型特征 很灵活，适用范围广。 能自动选择重要特征 在分裂过程中，算法会自动找“最有信息量”的特征。 训练和预测速度都比较快 缺点 容易过拟合 �� 如果不做剪枝（pruning），树会长得很“深”，对训练集拟合太好，但泛化能力差。

	对小的扰动敏感 �� 训练数据稍有变化，树的结构可能完全不同。 难以处理连续变化的复杂边界 决策树划分是“轴对齐”的，拟合复杂决策边界的能力有限。 偏好取值较多的特征 信息增益容易偏向特征取值多的属性，需要额外控制。

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(...)

```
# 计算父节点熵
total_samples = sum(all_labels.values())
parent_entropy = 0.0
for count in all_labels.values():
if count > 0:
p = count / total_samples
parent_entropy -= p * math.log2(p)
# 计算左子节点熵
left_samples = sum(left_labels.values())
left_entropy = 0.0
if left_samples > 0:
for count in left_labels.values():
if count > 0:
p = count / left_samples
left_entropy -= p * math.log2(p)
# 计算右子节点熵
right_samples = sum(right_labels.values())
right_entropy = 0.0
if right_samples > 0:
for count in right_labels.values():
if count > 0:
p = count / right_samples
right_entropy -= p * math.log2(p)
# 计算加权子节点熵
weighted_child_entropy = (left_samples / total_samples) * left_entropy +
(right_samples / total_samples) * right_entropy
info_gain = parent_entropy - weighted_child_entropy
```

   - __info_gain_ratio(...)

```
# 计算分裂信息
all_labels, left_labels, right_labels = __label_stat(y, l_y, r_y)
total_samples = sum(all_labels.values())
```

```
left_samples = sum(left_labels.values())
right_samples = sum(right_labels.values())
# 计算分裂信息
split_info = 0.0
if left_samples > 0:
p_left = left_samples / total_samples
split_info -= p_left * math.log2(p_left)
if right_samples > 0:
p_right = right_samples / total_samples
split_info -= p_right * math.log2(p_right)
# 计算信息增益率
if split_info > 0:
info_gain = info_gain / split_info
else:
info_gain = 0.0
```

- __gini_index(...)

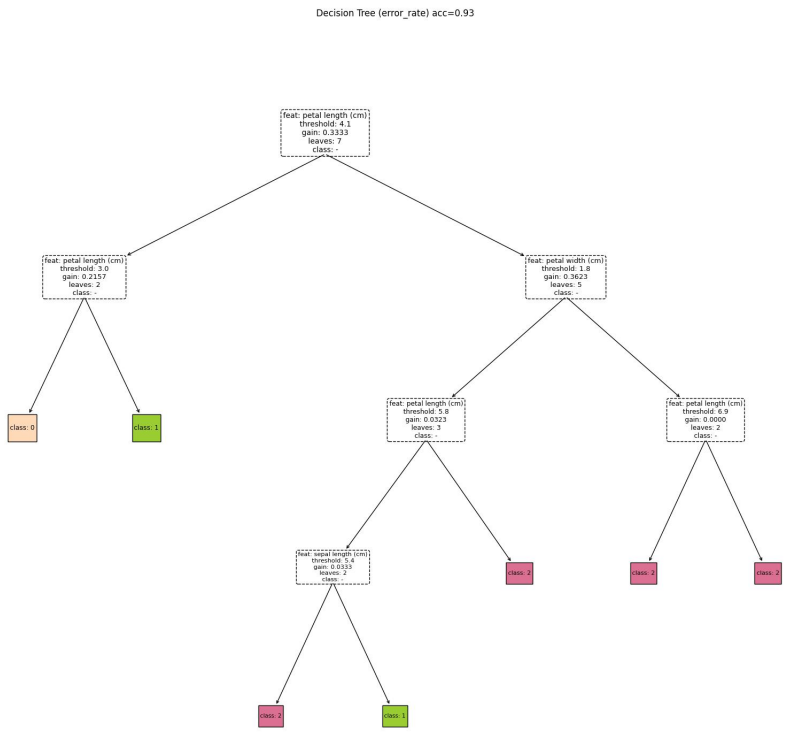
```
# 计算分裂前的 Gini 指数
total_samples = sum(all_labels.values())
before = 1.0
for count in all_labels.values():
if count > 0:
p = count / total_samples
before -= p * p
# 计算分裂后的 Gini 指数
left_samples = sum(left_labels.values())
right_samples = sum(right_labels.values())
left_gini = 1.0
if left_samples > 0:
for count in left_labels.values():
if count > 0:
p = count / left_samples
left_gini -= p * p
right_gini = 1.0
if right_samples > 0:
for count in right_labels.values():
if count > 0:
p = count / right_samples
right_gini -= p * p
after = (left_samples / total_samples) * left_gini + (right_samples /
total_samples) * right_gini
```

- __error_rate(...)

```
# 计算分裂前的分类误差率
total_samples = sum(all_labels.values())
max_count = max(all_labels.values()) if all_labels else 0
before = 1.0 - (max_count / total_samples) if total_samples > 0 else 0.0
# 计算分裂后的分类误差率
left_samples = sum(left_labels.values())
right_samples = sum(right_labels.values())
left_max_count = max(left_labels.values()) if left_labels else 0
left_error = 1.0 - (left_max_count / left_samples) if left_samples > 0 else
0.0
right_max_count = max(right_labels.values()) if right_labels else 0
right_error = 1.0 - (right_max_count / right_samples) if right_samples > 0
else 0.0
after = (left_samples / total_samples) * left_error + (right_samples /
total_samples) * right_error
```

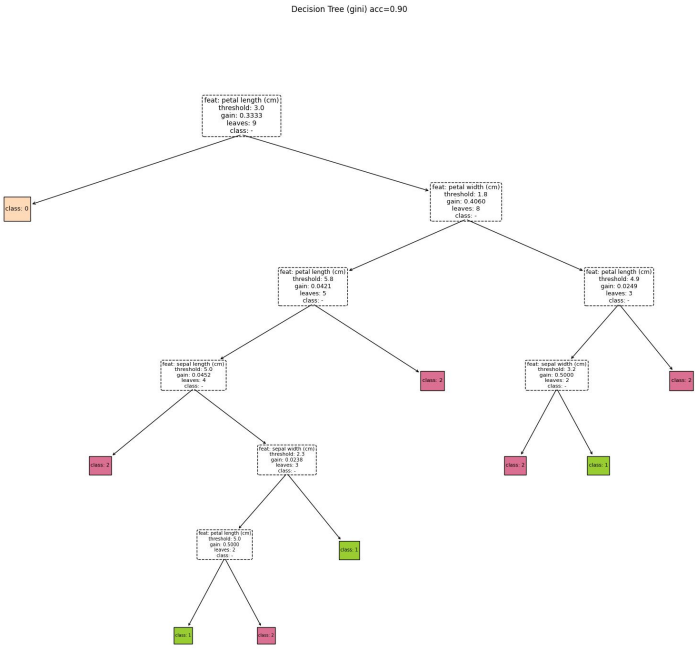
3. 测试和可视化 – 15%

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**



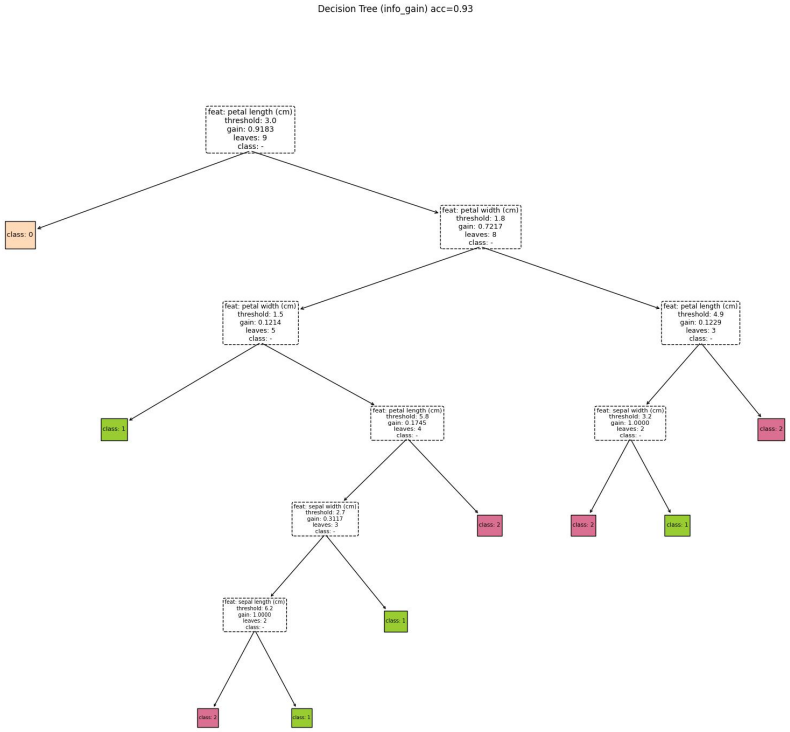
iris_error_rate

Accuracy = 0.9333 Tree_depth = 4 Tree_leaf_num = 7



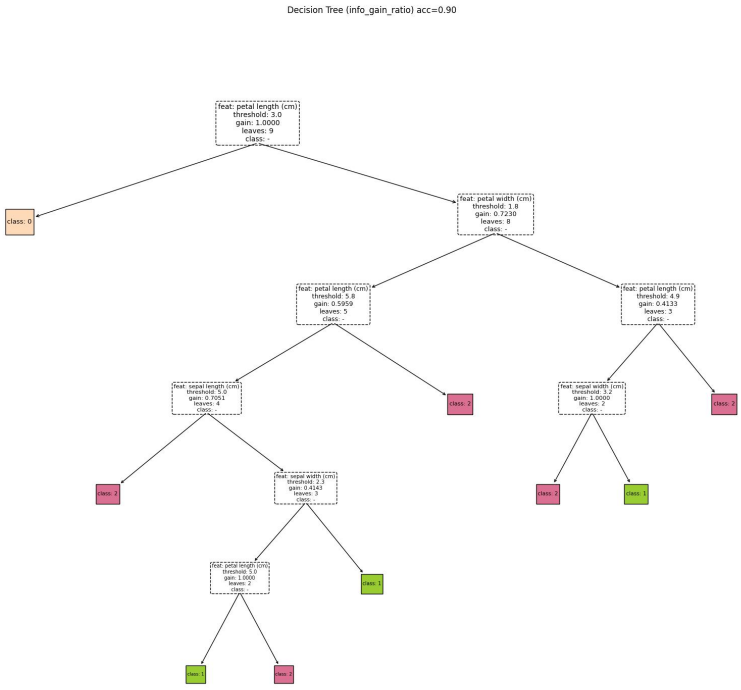
iris_gini

Accuracy = 0.9 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 = 0.9 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}

建议的搜索顺序

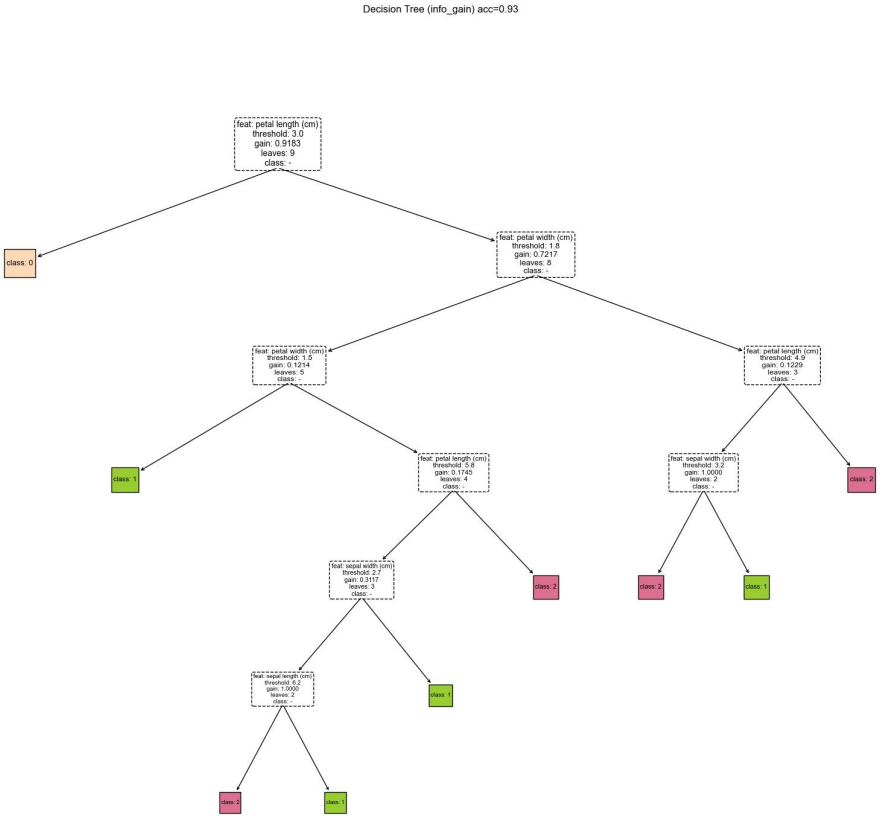
1. 先在 {criterion}×{splitter} 上做粗选
2. 固定上一步较优组合，再对 {max_depth,min_samples_split,min_impurity_split} 做粗到细的网格
3. 最后微调 max_features，在不降 Acc 的前提下简化树（更浅/更少叶）

你可以在下表中进行参数的尝试（至少3组），越高越好

	1	2	3	4

criterion	info_gain	info_gain	info_gain	info_gain
splitter	best	best	best	best
max_depth	5	6	7	7
min_samples_split	2	2	2	4
min_impurity_split	1-e4	1-e4	0.1	0.001
max_features	None	None	None	2
accuracy	0.9333	0.9	0.9333	0.9

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
计算该样本与训练集中所有样本的距离，选择距离最近的K 个邻居。	优点 原理简单、实现容易，不需要复杂的训练过程。 对异常值不敏感（如果 K 取值合适）。 适合多分类问题。 可以灵活选择距离度量方式，适应不同数据类型。 缺点

	计算成本高：每次预测都要与所有样本计算距离。 对数据规模和维度敏感，维度高时容易出现“距离不可靠”（维度灾难）。 需要选择合适的 K 值，太小容易过拟合，太大容易欠拟合。 无法直接解释模型内部机制，不是“可解释性强”的算法。

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.sqrt(np.sum((X_test[i] - X_train[j]) ** 2))
return dists
```

计算欧式距离

- L2 – noloop

```
X_test_expanded = X_test[:, np.newaxis, :] # (Nte, 1, D)
X_train_expanded = X_train[np.newaxis, :, :] # (1, Ntr, D)
# 计算平方差并求和，然后开方
dists = np.sqrt(np.sum((X_test_expanded - X_train_expanded) ** 2, axis=2))
return dists
```

利用numpy实现

- Cosine

```
# 计算点积
dot_product = np.dot(X_test, X_train.T) # (Nte, Ntr)
# 计算每个向量的 L2 范数
norm_test = np.linalg.norm(X_test, axis=1, keepdims=True) # (Nte, 1)
norm_train = np.linalg.norm(X_train, axis=1) # (Ntr,)
# 计算 cosine 相似度
cosine_sim = dot_product / (norm_test * norm_train)
# cosine 距离 = 1 - cosine 相似度
dists = 1 - cosine_sim
return dists
```

见注释

knn_predict()

```
for i in range(Nte):
idx = np.argsort(dists[i])[:k]
neighbors = y_train[idx]
```

```
# ================ TODO (students, REQUIRED) ================
```

```
from collections import Counter
# 统计 k 个最近邻的标签
label_counts = Counter(neighbors)
# 找到最高票数
max_votes = max(label_counts.values())
# 找到所有获得最高票数的标签
candidates = [label for label, count in label_counts.items() if count ==
max_votes]
# 平票时返回最小标签
y_pred[i] = min(candidates)
# ============================================
```

```
return y_pred
```

select_k_by_validation()

```
# ================ TODO (students, REQUIRED) ================
# 在验证集上网格搜索最优 k 值
accs = []
for k in ks:
# 使用当前 k 值在验证集上预测
y_pred_val = knn_predict(X_val, X_train, y_train, k, metric, mode)
# 计算准确率
accuracy = np.mean(y_pred_val == y_val)
accs.append(accuracy)
# 找到最高准确率对应的 k 值（如果有多个，选择最小的 k）
best_idx = np.argmax(accs)
best_k = ks[best_idx]
return best_k, accs
# ============================================================
```

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，

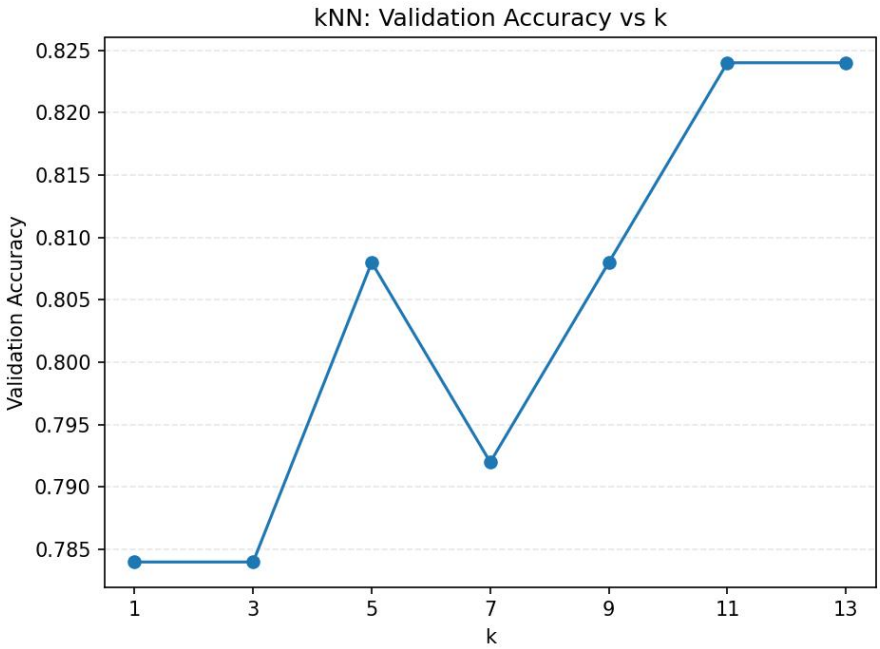
```
(/Users/Zhuanz/Documents/Senior/PRML/.conda) Zhuanz@MacBook-Air-9 k_nerest_neighbors % python test_k
nn.py
======================================================================
实验：Part 3 - kNN 分类
姓名：王镜凯    学号：22300240022    时间：2025-10-10 10:16:26
======================================================================
[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=13:0.82
40
 - 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.

(/Users/Zhuanz/Documents/Senior/PRML/.conda) Zhuanz@MacBook-Air-9 k_nerest_neighbors %
```

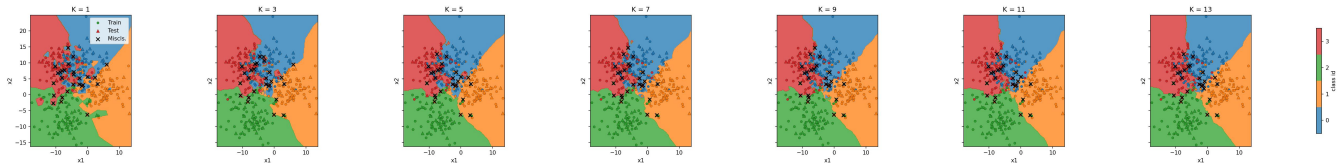
3.2

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’

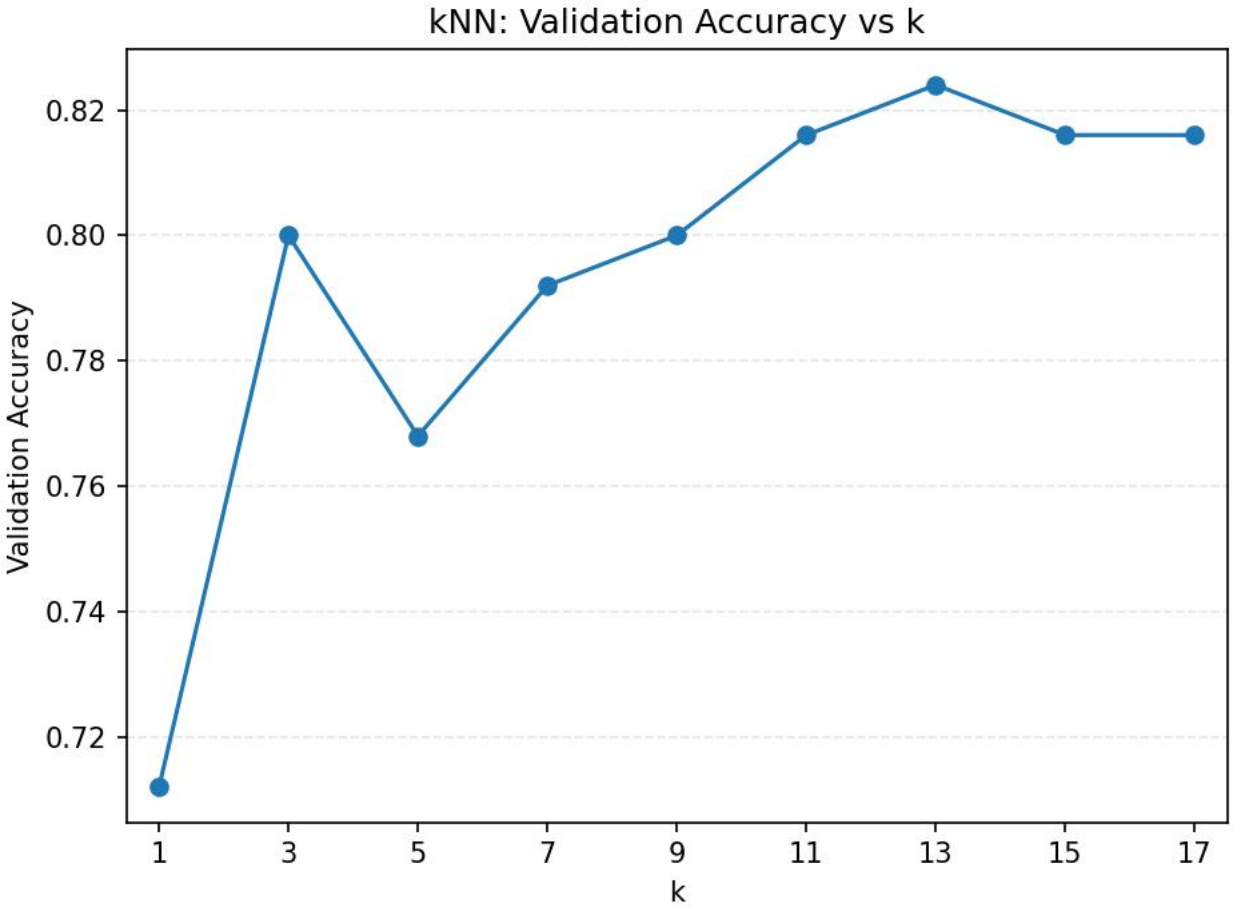


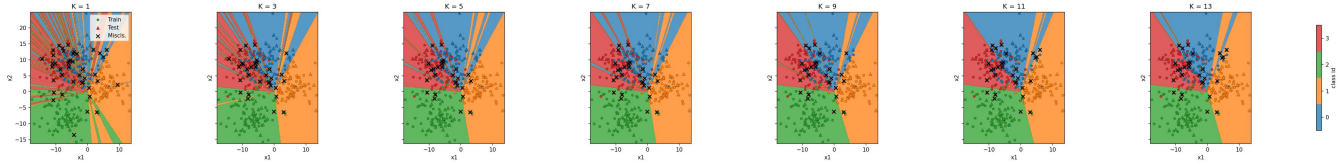
knn_k_curve



knn_boundary_grid.

b. Metric = ‘cosine’





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	12	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？

K=1时，算法只看最近的 一个邻居来决定类别。每个训练样本实际上都“拥有”自己周围的一小块区域。决策边界会紧紧贴着训练样本的分布，对局部噪声非常敏感。

随着K增大，算法会看更大范围的邻居，分类结果取决于一群邻居的多数表决，而不是某一个点。局部异常点的影响被“多数”抵消。决策边界变得更平滑，不容易被单个样本扰动；

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？

cosine呈现扇形，L2则是又曲线边缘形成分割。因为它们对距离的衡量方式不一样。
L2衡量欧几里得空间中的“直线距离”。cosine衡量两个向量之间的夹角。

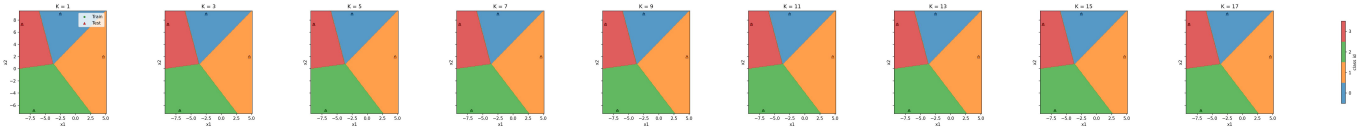
4. 进一步探索 - 8%

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。

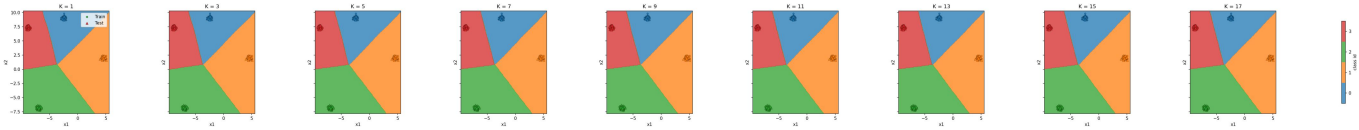
样例问题1：探索适合‘L2’和‘Cosine’的数据场景
通过修改data_generate.py中的数据参数，和k的选择，分别搜索能够在‘L2’和‘Cosine’方法下达到高准确度(95%以上)的数据集，分析两种距离计算方式适配的场景和数据结构。

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？

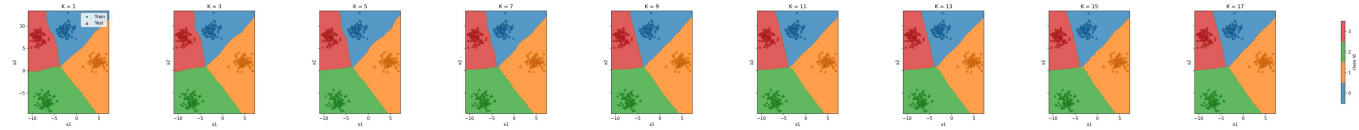
- best_k = 1(std = 0.1)



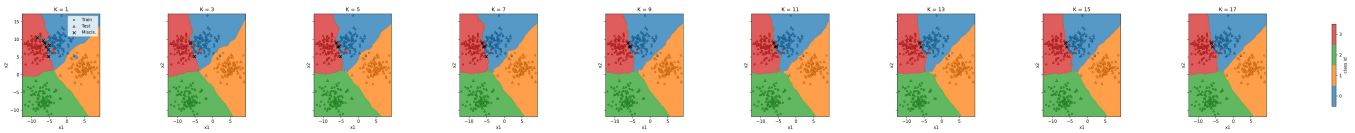
- best_k = 1 (std = 0.2)



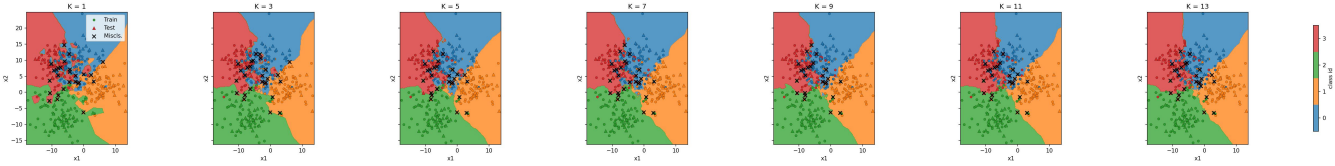
- best_k = 1(std = 1)



- best_k = 9 (std = 2)



- best_k = 12 (std = 4)



- best_k 是趋向变大，因为边缘噪声更多了，k增大有助于抗噪
- 边界越平滑，模型越不容易被少数噪声点牵着走。平滑边界是“看整体”，锯齿边界是“被局部绑架”。
- 两类数据重叠的部分容易判断错，这些区域容易被噪声干扰

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？

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