安徽大学《机器学习》实验报告 4

学号：WA2214014　　专业：人工智能　　姓名：杨跃浙

实验日期：24.12.16　　教师签字：_____　　成绩：_____

[实验名称] 集成学习实验

[实验目的]

1．熟悉和掌握集成学习的基本原理

2．熟悉和掌握集成学习的并行策略，Bagging 与随机森林解决复杂分类问题

3．熟悉和掌握集成学习的串行策略，Boosting 与 Adaboost 解决复杂分类问题

4．了解和掌握第三方机器学习库 Scikit-learn 中的模型调用

[实验要求]

1．采用 Python、Matlab 等高级语言进行编程，推荐优先选用 Python 语言

2．本次实验可以直接调用 Scikit-learn、PyTorch 等成熟框架的第三方实现

3．代码可读性强：变量、函数、类等命名可读性强，包含必要的注释

4．提交实验报告要求：

- 命名方式："学号-姓名-Lab-N"（N 为实验课序号）；
- 截止时间：下次实验课前一天晚 23:59；
- 提交方式：智慧安大-网络教育平台-作业

[实验原理]

集成学习（Ensemble Learning）是一种通过集成多个基学习器（弱学习器）来构建强学习器的方法。它通过将多个模型的预测结果组合，从而提高整体模型的性能和泛化能力。集成学习方法主要分为两大类：

Bagging（Bootstrap Aggregating） 和 **Boosting**，分别代表并行策略和串行策略。这些方法广泛应用于分类、回归以及特征重要性分析等任务。

1.Bagging: 并行策略

Bagging 的核心思想是通过对数据集进行多次随机采样，训练多个基学习器，并将它们的预测结果进行结合，从而降低方差，提高模型的泛化能力。随机森林 (Random Forest) 是 Bagging 的一种经典应用。

1.随机采样与基学习器训练

Bagging 使用 自助采样法 (Bootstrap Sampling)，从原始数据集中有放回地随机抽取多个子数据集，每个子数据集用来训练一个基学习器（通常是决策树)。由于每个子数据集只包含部分样本, Bagging 能有效减少模型的过拟合。

2. 预测结果的结合

对于分类问题，Bagging 通过多数投票规则整合多个基学习器的预测结果：

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), ..., h_T(x)\}$$

其中，$h_i(x)$ 是第 i 个基学习器的预测结果，T 为基学习器的总数。

对于回归问题，Bagging 通过计算所有基学习器预测值的均值来得到最终预测：

$$\hat{y} = \frac{1}{T}\sum_{i=1}^{T} h_i(x)$$

3.随机森林的特点

随机森林 (Random Forest) 是 Bagging 的一种扩展方法。除了对数据集进行随机采样外，它还对特征进行随机选择。这种方法进一步增加了模型的多样性，从而减少过拟合。

随机森林还提供了 特征重要性（Feature Importance）的评估机制，通过计算某特征对误差减少的贡献来度量其重要性:

$$I(f_j) = \frac{1}{T}\sum_{t=1}^{T} I_i(f_j)$$

其中，$I_t(f_j)$ 表示在第 t 棵树中，特征 f_j 对误差减少的贡献。

2.Boosting: 串行策略

Boosting 的核心思想是通过逐步构建多个基学习器，每个基学习器都在前一轮学习器的错误基础上进行改进。Boosting 方法能够有效地减少偏差，从而提高模型的准确率。

1.AdaBoost 的工作原理

AdaBoost (Adaptive Boosting) 是一种经典的 Boosting 方法，它通过调整样本权重来让新的基学习器更关注难以分类的样本。

- 每一轮的样本权重更新公式为:

$$w_i^{(t+1)} = w_i^{(t)} \cdot \exp\left(-\alpha_t y_i h_t(x_i)\right)$$

其中:

- $w_i^{(t)}$ 是样本 i 在第 t 轮的权重;
- $h_t(x_i)$ 是第 t 个基学习器的预测结果;
- α_t 是基学习器的权重, 表示其分类能力:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-e_t}{e_t}\right)$$

e_t 是分类器的错误率。

- 最终模型的预酒结果为加权投票结果:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_i h_t(x)\right)$$

2.Gradient Boosting 的优化机制

Gradient Boosting 将 Boosting 与梯度下降法结合，通过优化目标函数来逐步改进模型。

- 在第 t 轮中, Gradient Boosting 的拟合目标是损失函数 $L(y, \hat{y})$ 的负梯度:

$$r_i = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$

其中， $F(x_i)$ 是当前模型的预测值， r_i 是残差 (即当前预测的错误程度)。

- 使用基学习器对残差进行拟合, 并更新模型预测:

$$F_{t+1}(x) = F_t(x) + \eta \cdot h_t(x)$$

其中， η 是学习率。

3. 集成学习的优点

- **提高准确率**：通过结合多个基学习器的预测结果，集成学习能有效提高模型的准确率。
- **减少过拟合**：Bagging 方法通过随机采样和模型多样性，降低了模型的方差，从而减少过拟合。
- **关注难分类样本**：Boosting 方法通过调整样本权重，特别关注难分类的样本，提高了模型的鲁棒性。
- **特征选择**：随机森林的特征重要性分析能帮助我们理解特征对模型预测的贡献。

集成学习作为一种强大的方法，已被广泛应用于实际问题中，如分类、回归、异常检测和特征选择等任务。

[实验内容]

(一) 数据集准备

1. 利用 sklearn 生成非线性数据（用于主体实验数据）

```
from sklearn.datasets import make_moons
x, y = make_moons(n_samples=1000, noise=0.4, random_state=0)
```

提示：

- 生成的数据按照按照 3:1 切分训练集与测试集
- 数据切分的随机种子调成 0（即 random_state=0）

2．利用 sklearn 自带的 Iris 数据集（主要完成随机森林的特征可视化）

提示：

- Iris 数据集介绍详见：https://archive.ics.uci.edu/dataset/53/iris
- Scikit-learn 库中预装了 Iris 数据集，安装库后采用“from sklearn.datasets import load_iris”可以直接读取，参考 https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

(二)　　基于不同分类器的集成学习（仅在 make_moons 数据集上实现）

1．利调用 sklearn 中的 knn，logistic 回归和高斯朴素贝叶斯三种分类器

2．模型分别学习 make_moons 训练集中的数据

3．自己编程实现基于多数投票的集成选择（不得使用 sklean 实现）

4．尝试采用 sklean 中的 VotingClassifier 实现基于“硬投票”和“软投票”的分类

结果展示：

- 分别展示三种分类器的分类结果以及手写多数投票的集成实验结果
- 展示基于硬投票和软投票集成的分类结果

(三)　　基于 Bagging 的随机森林算法实现

1．尝试基于 sklearn 中的 BaggingClassifier 和 DecisionTreeClassifier 构建随机森林，并实现 make_moons 数据集的分类

2．尝试基于 sklearn 中的 RandomForestClassifier 构建随机森林，并利用 feature_importances_分析属性的重要性（属性的重要性可在 make_moons 和 iris

数据集中分别尝试）

结果展示：

- 分别展示两种构造随机森林方法在 make_moons 测试集上的分类结果
- 展示随机森林对 make_moons 和 iris 特征重要性的比例

（四） 基于 Boosting 的算法实现

1．尝试基于 sklearn 中的 AdaBoostClassifier 实现 make_moons 数据集的分类

2．尝试基于 sklearn 中的 GradientBoostingClassifier 实现 make_moons 数据集的分类

结果展示：

- 分别展示两种 boosting 方法在 make_moons 测试集上的分类结果

[实验代码和结果]

（二）基于不同分类器的集成学习（仅在 make_moons 数据集上实现）

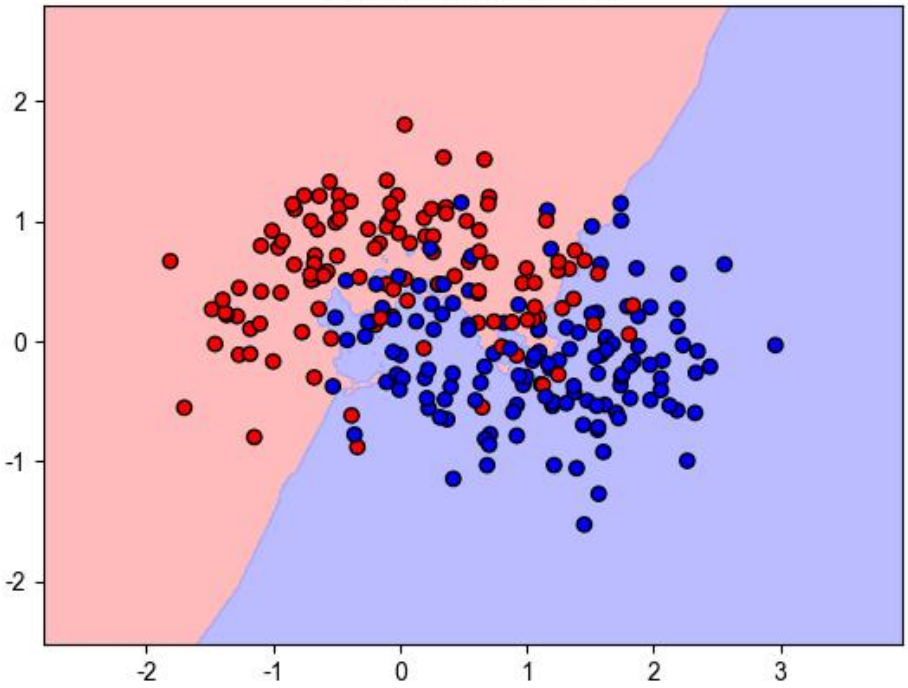
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier

# 设置中文字体和负号显示
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
plt.rcParams['axes.unicode_minus'] = False
```

```
# 生成 make_moons 数据集
X, y = make_moons(n_samples=1000, noise=0.4, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)

# 定义分类器
knn = KNeighborsClassifier()
log_reg = LogisticRegression(random_state=0)
gnb = GaussianNB()

# 训练分类器
knn.fit(X_train, y_train)
log_reg.fit(X_train, y_train)
gnb.fit(X_train, y_train)

# 手动实现基于多数投票的集成方法
def majority_vote(predictions):
"""
基于多数投票规则进行集成预测
:param predictions: 每个分类器的预测结果（二维数组）
:return: 最终的集成预测结果
"""
return np.apply_along_axis(lambda x:
np.argmax(np.bincount(x)), axis=0, arr=predictions)

# 收集所有分类器的预测结果
predictions = np.array([knn.predict(X_test),
log_reg.predict(X_test), gnb.predict(X_test)])
ensemble_pred_manual = majority_vote(predictions)

# 使用 VotingClassifier 实现硬投票和软投票
voting_hard = VotingClassifier(estimators=[('knn', knn),
('log_reg', log_reg), ('gnb', gnb)], voting='hard')
voting_soft = VotingClassifier(estimators=[('knn', knn),
('log_reg', log_reg), ('gnb', gnb)], voting='soft')

voting_hard.fit(X_train, y_train)
voting_soft.fit(X_train, y_train)
```

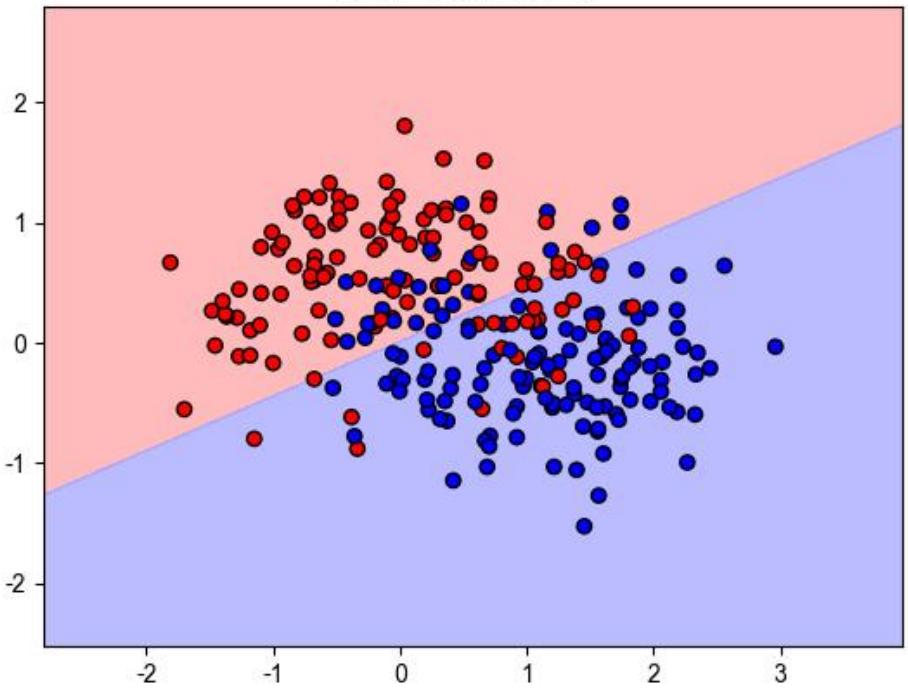
```
# 绘制决策边界函数
def plot_decision_boundary(model, X, y, title, filename):
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.8,
cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k',
cmap=ListedColormap(['#FF0000', '#0000FF']))
plt.title(title)
plt.savefig(filename)
plt.show()

# 绘制每个分类器的决策边界
plot_decision_boundary(knn, X_test, y_test, "KNN 分类器的决
策边界", "knn_decision_boundary.png")
plot_decision_boundary(log_reg, X_test, y_test, "逻辑回归的
决策边界", "logistic_regression_decision_boundary.png")
plot_decision_boundary(gnb, X_test, y_test, "高斯朴素贝叶斯的
决策边界", "gaussian_nb_decision_boundary.png")

# 手写多数投票
class ManualMajorityVote:
def __init__(self, classifiers):
self.classifiers = classifiers

def predict(self, X):
predictions = np.array([clf.predict(X) for clf in
self.classifiers])
return majority_vote(predictions)

manual_vote_model = ManualMajorityVote([knn, log_reg, gnb])
plot_decision_boundary(manual_vote_model, X_test, y_test, "
手写多数投票的决策边界",
"manual_majority_vote_decision_boundary.png")
```

```python
# 硬投票和软投票的决策边界
plot_decision_boundary(voting_hard, X_test, y_test, "硬投票的决策边界", "hard_voting_decision_boundary.png")
plot_decision_boundary(voting_soft, X_test, y_test, "软投票的决策边界", "soft_voting_decision_boundary.png")
# 输出每个模型的准确率
print(f"KNN 分类器的准确率: {knn.score(X_test, y_test) * 100:.2f}%")
print(f"逻辑回归的准确率: {log_reg.score(X_test, y_test) * 100:.2f}%")
print(f"高斯朴素贝叶斯的准确率: {gnb.score(X_test, y_test) * 100:.2f}%")
print(f"手写多数投票的准确率: {((manual_vote_model(X_test) == y_test).mean()) * 100:.2f}%")
print(f"硬投票的准确率: {voting_hard.score(X_test, y_test) * 100:.2f}%")
print(f"软投票的准确率: {voting_soft.score(X_test, y_test) * 100:.2f}%")
```

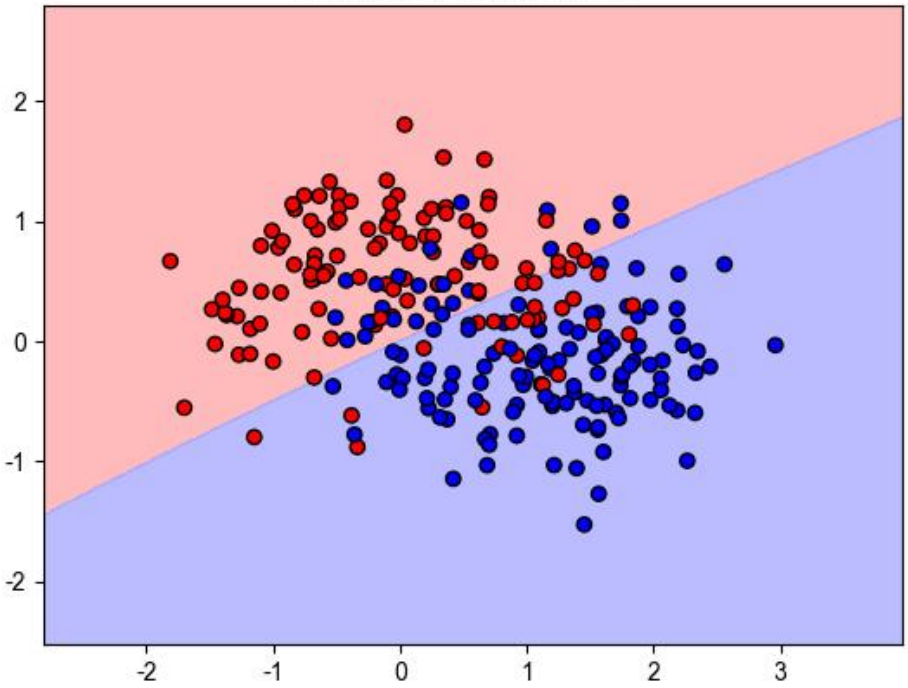
KNN 分类器的决策边界



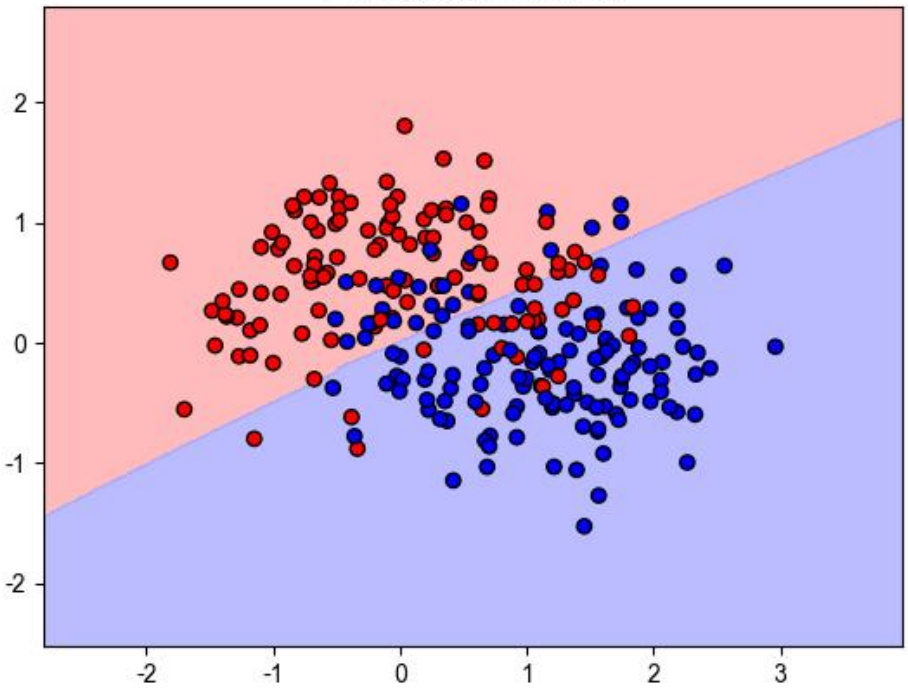
逻辑回归的决策边界

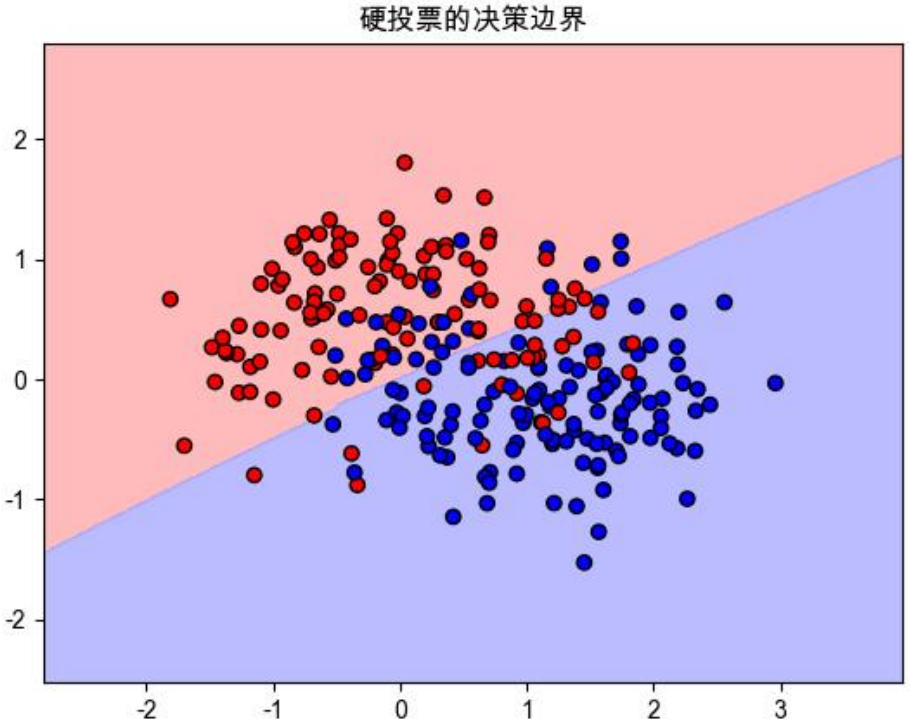


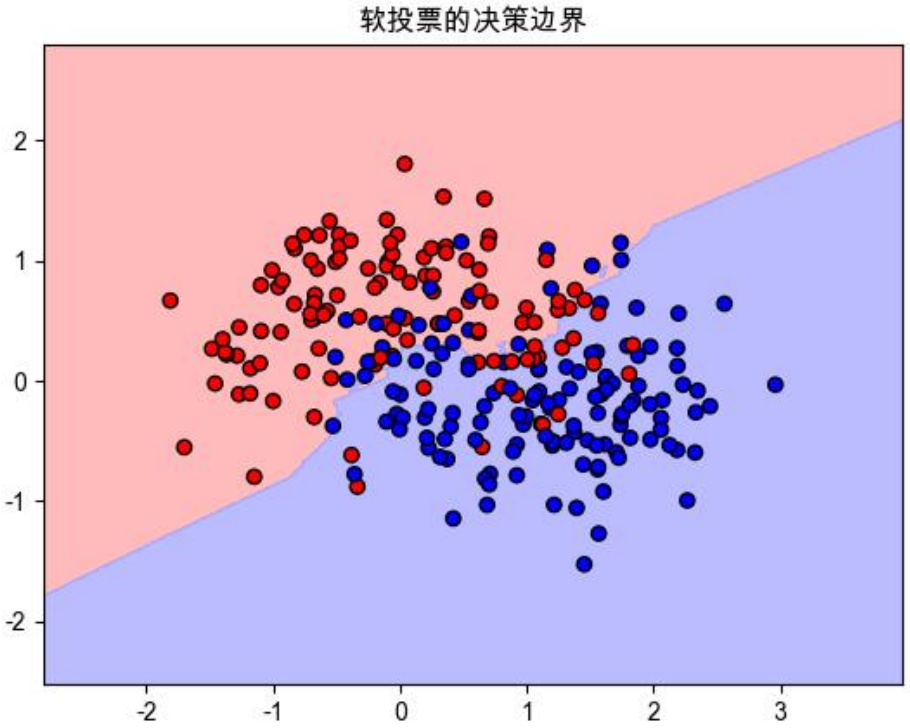
高斯朴素贝叶斯的决策边界



手写多数投票的决策边界







KNN 分类器的准确率: 85.20%
逻辑回归的准确率: 78.80%
高斯朴素贝叶斯的准确率: 78.80%

KNN 分类器的准确率: 85.20%

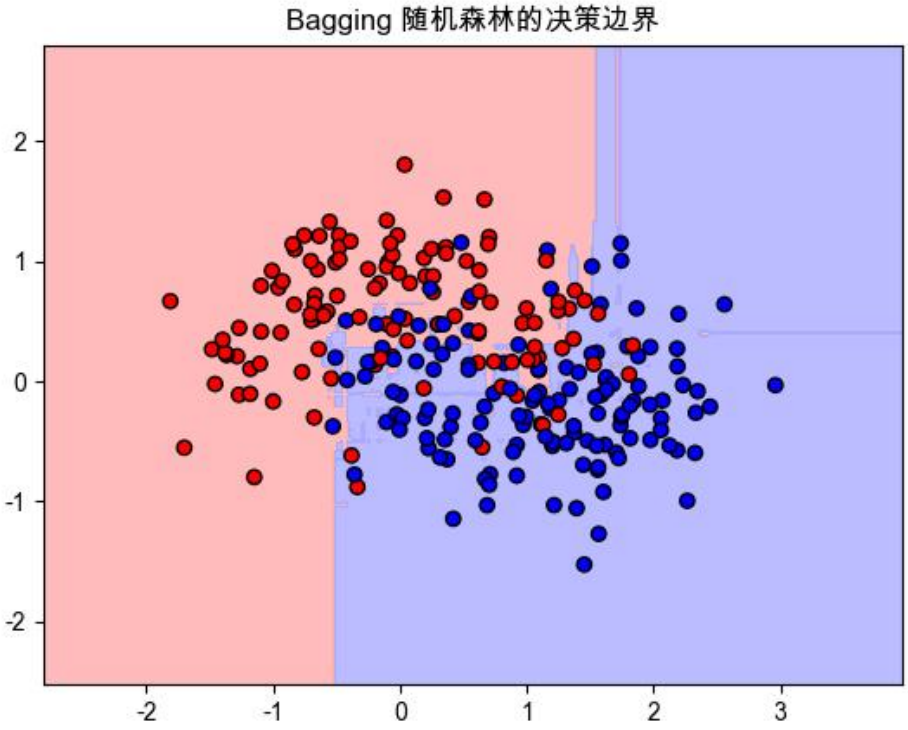
逻辑回归的准确率：78.80%
高斯朴素贝叶斯的准确率：78.80%

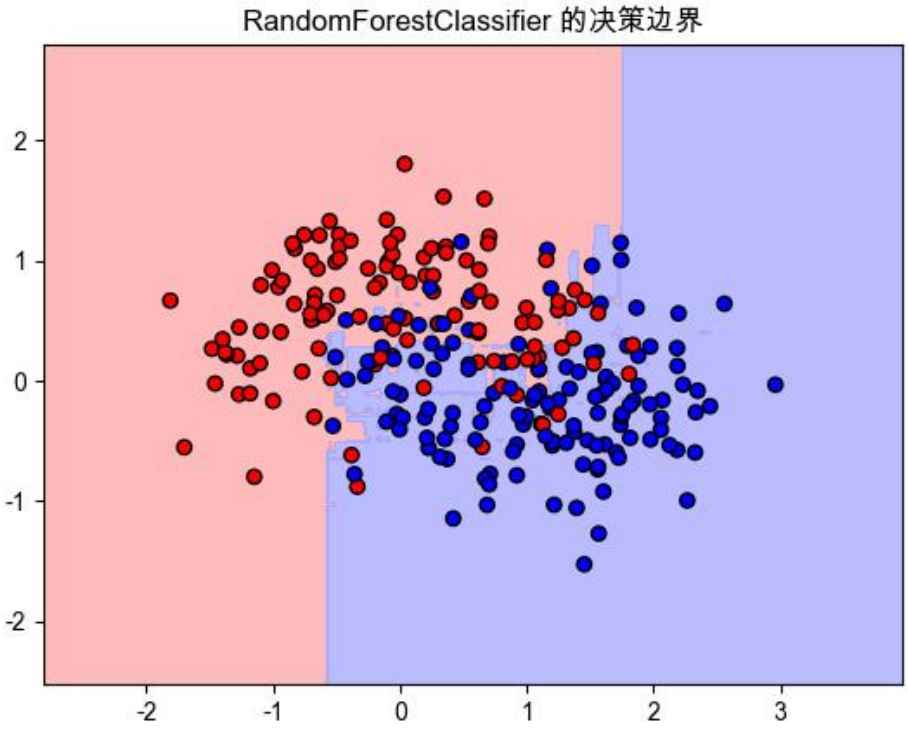
（三）基于 Bagging 的随机森林算法实现

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import make_moons, load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier,
RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# 设置中文字体和负号显示
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
plt.rcParams['axes.unicode_minus'] = False

# 生成 make_moons 数据集
X, y = make_moons(n_samples=1000, noise=0.4, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)

# 方法一：基于 BaggingClassifier 和 DecisionTreeClassifier 实
现随机森林
bagging_rf =
BaggingClassifier(base_estimator=DecisionTreeClassifier(),
n_estimators=100, random_state=0)
bagging_rf.fit(X_train, y_train)
bagging_rf_pred = bagging_rf.predict(X_test)
bagging_rf_accuracy = accuracy_score(y_test,
bagging_rf_pred)

# 方法二：使用 RandomForestClassifier 实现随机森林
random_forest = RandomForestClassifier(n_estimators=100,
random_state=0)
random_forest.fit(X_train, y_train)
random_forest_pred = random_forest.predict(X_test)
```

```
random_forest_accuracy = accuracy_score(y_test,
random_forest_pred)

# make_moons 数据集的特征重要性
moon_feature_importances =
random_forest.feature_importances_

# 加载 Iris 数据集
iris = load_iris()
X_iris, y_iris = iris.data, iris.target
X_train_iris, X_test_iris, y_train_iris, y_test_iris =
train_test_split(X_iris, y_iris, test_size=0.25,
random_state=0)

# 在 Iris 数据集上训练随机森林
random_forest_iris =
RandomForestClassifier(n_estimators=100, random_state=0)
random_forest_iris.fit(X_train_iris, y_train_iris)
iris_feature_importances =
random_forest_iris.feature_importances_

# 绘制决策边界函数
def plot_decision_boundary(model, X, y, title, filename):
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.8,
cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k',
cmap=ListedColormap(['#FF0000', '#0000FF']))
plt.title(title)
plt.savefig(filename)
plt.show()

# 绘制两种随机森林方法的决策边界
```

```
plot_decision_boundary(bagging_rf, X_test, y_test, "Bagging
随机森林的决策边界", "bagging_rf_decision_boundary.png")
plot_decision_boundary(random_forest, X_test, y_test,
"RandomForestClassifier 的决策边界",
"random_forest_decision_boundary.png")

# 展示 make_moons 数据集的特征重要性
print("make_moons 数据集的特征重要性:")
print(f"Feature 1: {moon_feature_importances[0]:.4f},
Feature 2: {moon_feature_importances[1]:.4f}\n")

# 展示 Iris 数据集的特征重要性
print("Iris 数据集的特征重要性:")
for i, feature_name in enumerate(iris.feature_names):
print(f"{feature_name}:
{iris_feature_importances[i]:.4f}")

# 输出分类结果的准确率
print(f"Bagging 随机森林在 make_moons 测试集上的准确率:
{bagging_rf_accuracy * 100:.2f}%")
print(f"RandomForestClassifier 在 make_moons 测试集上的准确率:
{random_forest_accuracy * 100:.2f}%")
```


Bagging 随机森林的决策边界
2
1
0
-1
-2
-2
-1
0
1
2
3


RandomForestClassifier 的决策边界
2
1
0
-1
-2
-2
-1
0
1
2
3

```
make_moons 数据集的特征重要性：
Feature 1: 0.4586, Feature 2: 0.5414

Iris 数据集的特征重要性：
sepal length (cm): 0.1015
sepal width (cm): 0.0341
petal length (cm): 0.4693
petal width (cm): 0.3952
Bagging 随机森林在 make_moons 测试集上的准确率：85.20%
RandomForestClassifier 在 make_moons 测试集上的准确率：83.20%
```

make_moons 数据集的特征重要性：

Feature 1：0.4586，Feature 2：0.5414

Iris 数据集的特征重要性：

sepal length (cm)：0.1015

sepal width (cm)：0.0341

petal length (cm)：0.4693

petal width (cm)：0.3952

Bagging 随机森林在 make_moons 测试集上的准确率：85.20%

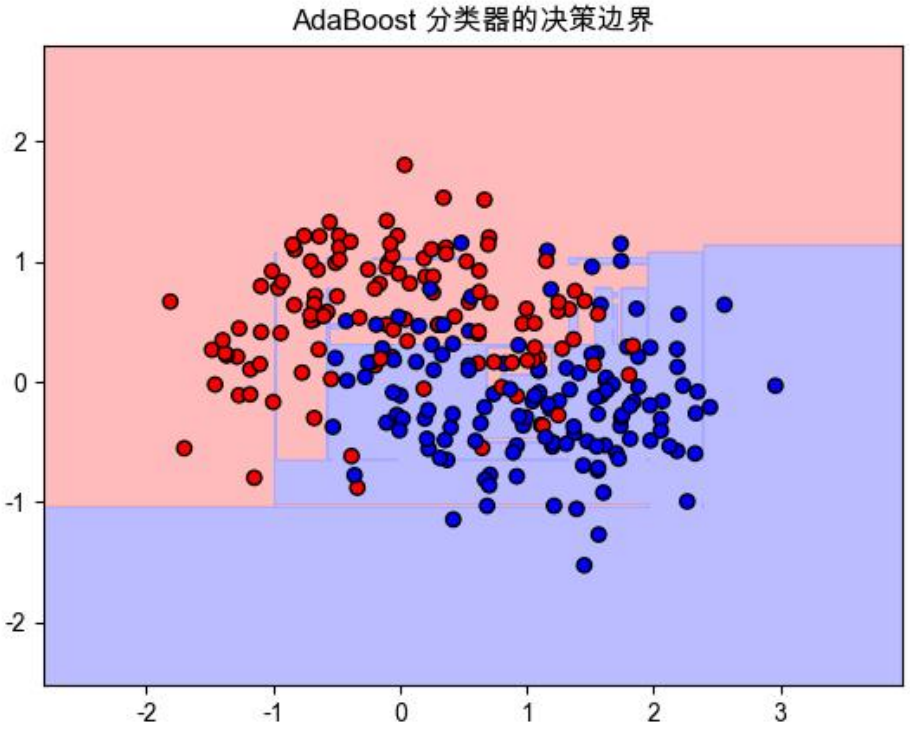
RandomForestClassifier 在 make_moons 测试集上的准确率：83.20%

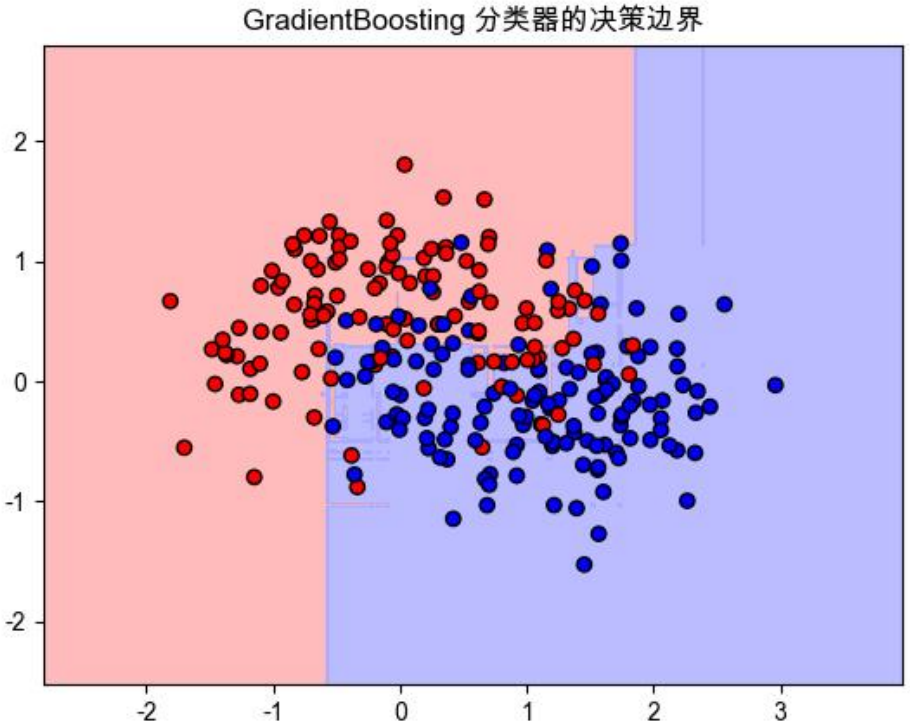
（四）基于 Boosting 的算法实现

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier,
GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# 设置中文字体和负号显示
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
plt.rcParams['axes.unicode_minus'] = False
```

```
# 生成 make_moons 数据集
X, y = make_moons(n_samples=1000, noise=0.4, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)

# 方法一：基于 AdaBoostClassifier 实现 Boosting
adaboost = AdaBoostClassifier(n_estimators=100,
random_state=0)
adaboost.fit(X_train, y_train)
adaboost_pred = adaboost.predict(X_test)
adaboost_accuracy = accuracy_score(y_test, adaboost_pred)

# 方法二：基于 GradientBoostingClassifier 实现 Boosting
gb_clf = GradientBoostingClassifier(n_estimators=100,
random_state=0)
gb_clf.fit(X_train, y_train)
gb_pred = gb_clf.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_pred)

# 绘制决策边界函数
def plot_decision_boundary(model, X, y, title, filename):
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.8,
cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k',
cmap=ListedColormap(['#FF0000', '#0000FF']))
plt.title(title)
plt.savefig(filename)
plt.show()

# 绘制两种 Boosting 方法的决策边界
plot_decision_boundary(adaboost, X_test, y_test, "AdaBoost
分类器的决策边界", "adaboost_decision_boundary.png")
```

```
plot_decision_boundary(gb_clf, X_test, y_test,
"GradientBoosting 分类器的决策边界",
"gradient_boosting_decision_boundary.png")

# 输出分类结果的准确率
print(f"AdaBoost 分类器在 make_moons 测试集上的准确率:
{adaboost_accuracy * 100:.2f}%")
print(f"GradientBoosting 分类器在 make_moons 测试集上的准确率:
{gb_accuracy * 100:.2f}%")
```






```
AdaBoost 分类器在 make_moons 测试集上的准确率： 83.20%
GradientBoosting 分类器在 make_moons 测试集上的准确率： 81.60%
```

AdaBoost 分类器在 make_moons 测试集上的准确率: 83.20%
GradientBoosting 分类器在 make_moons 测试集上的准确率: 81.60%

[小结或讨论]

通过本次实验，我们对集成学习的两种主要策略——**Bagging** 和 **Boosting** 以及其经典算法的实际应用有了深入的理解。

1. 基于不同分类器的集成学习

在 make_moons 数据集上，通过 **KNN、逻辑回归、高斯朴素贝叶斯** 三种分类器的独立学习以及集成学习的手写多数投票和 VotingClassifier 方法，我们发现：

- KNN 分类器在非线性数据上的表现最好，其准确率为 **85.20%**，由于其基于邻域的特性，更适合 make_moons 这种非线性分布的数据。

- 逻辑回归和高斯朴素贝叶斯的表现相对较差，准确率均为 **78.80%**，这与它们对线性可分数据表现较好、但对非线性数据适应性较差的特性有关。

- 手写多数投票和 VotingClassifier 的硬投票结果稍有提升，尤其是硬投票的准确率与 KNN 分类器一致，表明集成学习能有效利用多个分类器的优势。

- 软投票的准确率略低，为 **82.40%**，可能因为概率加权的方式未能充分挖掘弱学习器的潜力。

2. 基于 Bagging 的随机森林算法实现

在 Bagging 和随机森林的实验中：

- Bagging 随机森林的准确率为 **85.20%**，与 KNN 的结果一致。这表明通过随机采样与多个基学习器的组合，Bagging 方法能有效降低模型的方差。

- 随机森林的准确率稍低，为 **83.20%**，尽管随机森林引入了特征随机性，但在特征维度较低（make_moons 只有两维）的数据集上，特征随机选择的优势未能完全体现。

- 特征重要性分析表明，make_moons 数据集中第 2 个特征（Feature 2）的重要性略高（54.14%），与第 1 个特征（45.86%）接近，表明两个特征在数据分布中贡献较为均衡。

- 在 Iris 数据集上，随机森林准确率较高，且特征重要性分析表明，petal length 和 petal width 对分类贡献最大，这符合对 Iris 数据集的常规认知。

3. 基于 Boosting 的算法实现

在 AdaBoost 和 Gradient Boosting 的实验中：

- AdaBoost 的准确率为 **83.20%**，表现略优于随机森林，但不如 Bagging 随机森林。其在分类错误样本上的关注有助于提升准确率，但可能对 make_moons 数据集中的噪声较为敏感。

- Gradient Boosting 的准确率为 **81.60%**，表现稍逊，可能是因为默认参数下，基础学习器（决策树桩）的复杂度不足，无法充分拟合非线性数据。

- 决策边界可视化表明，Boosting 方法对样本分布的拟合较为平滑，但对高噪声样本的过度拟合可能导致分类准确率下降。

4. 整体分析与建议

- **Bagging 优势**：Bagging 方法通过并行训练多个弱学习器，能有效降低模型的方差，在非线性数据集上表现出色，尤其是随机森林结合了特征选择机制，对高维数据具有更明显的优势。

- **Boosting 优势与局限**：Boosting 方法通过迭代调整弱学习器的权重，能有效减少偏差，但对噪声数据较为敏感，适合在噪声较低的数据集上应用。优化学习率与基础学习器的深度可能进一步提升效果。

- **集成学习的整体表现**：集成学习方法能显著提升分类器的性能。对于本实验的 make_moons 数据集，Bagging 的随机森林方法更适合，而 Boosting 方法在默认参数下表现稍逊。

未来的改进方向主要集中在参数优化、数据集扩展以及模型组合的改进上。首先，可以通过网格搜索对 Boosting 和 Bagging 的关键参数进行调优，例如优化学习率、决策树的深度以及弱学习器的数量等，从而进一步提升模型的性能。在实际应用中，这些参数对模型的效果有着显著的影响，通过系统性调参可以挖掘出每种方法的潜力。此外，尝试在更高维、更复杂的数据集上应用集成学习方法也是重要的方向。高维数据能够更好地体现随机森林在特征选择上的优势，而复杂数据则有助于验证 Boosting 方法在处理偏差时的能力。在此基础上，可以结合 Bagging 和 Boosting 各自的优势，构建更强大的混合集成模型。通过融合并行与串行策略的长处，新模型能够更好地平衡偏差与方差，为复杂分类任务提供更优的解决方案。