**智能信息系统综合实践**

**实验报告**

|  |  |
| --- | --- |
| **题 目：** | 集成学习 |
| **年 级：** | **2021** |
| **专 业：** | **软件工程** |
| **学 号：** | **2021117405** |
| **姓 名：** | **孙潇桐** |

目录

[1. 题目 3](#_Toc163908540)

[2. 解题步骤 3](#_Toc163908541)

[2.1. 题目一：采用AdaBoost算法对一个自选数据集进行分类实验 3](#_Toc163908542)

[2.1.1. 对数据的处理 3](#_Toc163908543)

[2.1.2. 关于AdaBoost 3](#_Toc163908544)

[2.2. 题目二：测试不同分类器和集成策略对性能的影响 5](#_Toc163908545)

[2.2.1. 关于集成策略 5](#_Toc163908546)

[2.2.2. 关于基分类器 5](#_Toc163908547)

[2.2.3. 总结不同分类器对性能的影响 8](#_Toc163908548)

[2.3. 题目三：讨论和图示基分类器的偏差和方差 9](#_Toc163908549)

[2.3.1. 关于偏差和方差 9](#_Toc163908550)

[2.3.2. 计算基分类器的偏差和方差 10](#_Toc163908551)

[3. 总结 10](#_Toc163908552)

[4. 附件 11](#_Toc163908553)

[4.1. 完整代码 11](#_Toc163908554)

[4.1.1. 测试SAMME和SAMME.R在深度为2的决策树上的效果ada\_boost\_DT.py 11](#_Toc163908555)

[4.1.2. 测试在SAMME.R下不同基分类器的表现并画图ada\_boost\_varies.py 12](#_Toc163908556)

[4.1.3. 工具集合utils.py 17](#_Toc163908557)

[4.2. 参考资料 17](#_Toc163908558)

# 题目

1. 采用AdaBoost算法对一个自选数据集进行分类实验。
2. 测试不同分类器和集成策略对性能的影响
3. 讨论和图示基分类器的偏差和方差

# 解题步骤

## 题目一：采用AdaBoost算法对一个自选数据集进行分类实验

### 对数据的处理

数据集我使用了前几次实验都用过的iris数据集，使用我写在utils.py中的load\_data函数读入特征值和标签，然后使用42为随机种子，将数据集划分为训练集和测试集。

X, y, title = utils.load\_data("data/iris.csv")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

### 关于AdaBoost

AdaBoost（Adaptive Boosting）是一种集成学习的算法，它通过迭代地训练一系列学习器并将它们组合起来，从而构成一个强大的分类器。为了能更加的凸显各个学习器的优势，这些学习器应该“好而不同”。为了保证“好而不同”，AdaBoost采用了如下的策略：

首先，对于AdaBoost来说，它使用了我们指定个数m的**相同小分类器**，并将每个模型的权重的初始化为。接下来，对数据进行m此迭代。每次迭代用当前带权重的数据集对新的小分类器进行分类，得到当前的分类器的错误率，如果的话就停止循环（结束训练）。如果没有跳出循环的话，计算当前的模型权重，在sklearn提供的库中还有另外一个参数learn\_rate，通过学习率控制每个小分类器对总体的贡献，也就是。接下来就需要通过更新样本权重，我认为这是AdaBoost最重要的步骤。通过更新权重，我们可以通过增加分类错误样本的权重，让后面的模型更加关注数据集中分类错误的部分，用后面的模型“补足”前面的模型的不足。这就是AdaBoost的整个训练过程，下面我将分步骤讲解我的实现：

首先，我们需要给AdaBoost定义一个基学习器，我这里使用了一个两层的决策树：

# 基学习器（深度为2的决策树）

base\_estimator = DecisionTreeClassifier(max\_depth=2)

为了展示出AdaBoost的全部实力，我设置了一些常用的参数，通过sklearn提供的GridSearchCV方法来对这些参数进行网格搜索。

# 建立AdaBoost分类器

clf = AdaBoostClassifier(estimator=base\_estimator, random\_state=42)

# 参数字典

param\_grid = {

    "n\_estimators": [10, 30, 50, 100, 150, 200],

    "learning\_rate": [0.05, 0.1, 0.5, 1.0, 1.5],

}

# 构建网格搜索器，使用五折交叉验证确定最佳参数

grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring="accuracy")

grid\_search.fit(X\_train, y\_train)

我在训练完成之后，画出了所有参数和5折交叉验证的准确率的热力图：

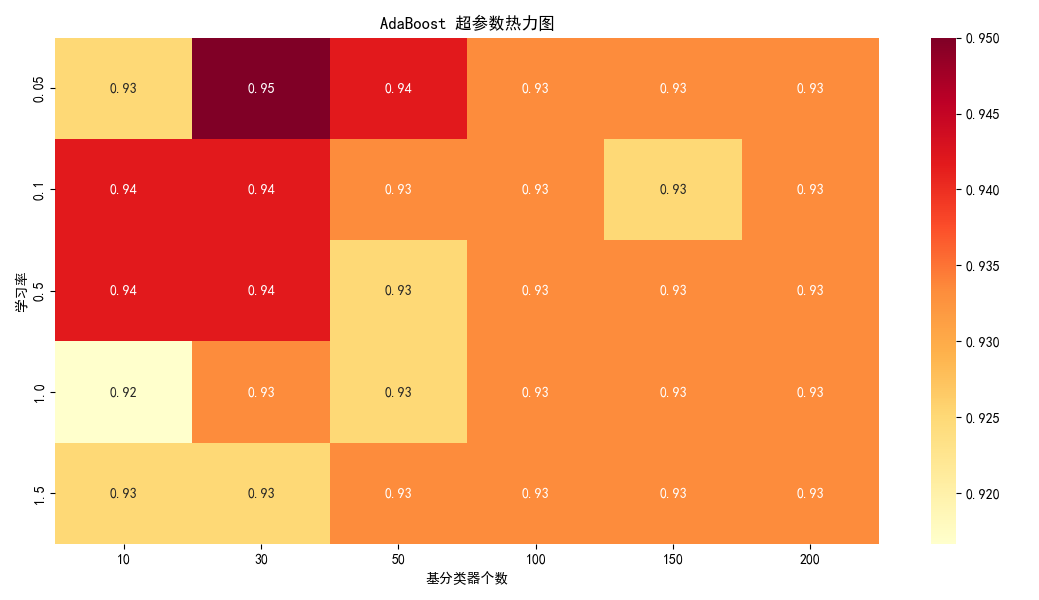


Figure 1 AdaBoost(SAMME.R) 超参数热力图

从上面的结果可以很轻易的看出，基分类器并不是越多越好。因为这个数据集比较小，所以过多的基分类器可能造成过拟合的问题，从而导致准确率的下降。同时学习率和分类器个数也有关系，当学习率较小的时候，由于每个分类器对总体的贡献较小，可能需要更多的基分类器来保证后面的分类器能够学习到前面分类器出错的部分。在得到了最佳参数之后，我在测试集上进行了测试，准确率达到了**96.667%**

## 题目二：测试不同分类器和集成策略对性能的影响

### 关于集成策略

sklearn中的AdaBoostClassifier提供了两种集成策略，分别是SAMME 和SAMME.R（默认），下面我来解释一下这两个集成策略[1]：

* SAMME：这个算法和传统的AdaBoost算法比较相似，传统算法的具体的步骤在上文（[2.1.2节](#_关于AdaBoost)）有提到。不过，为了支持多分类，这个算法使用one-hot编码，也就是将标签编码为长度为标签个数且只有当前标签下标为1其余元素为0的一维向量。每个基分类器的输出也为长度为标签个数的向量，通过计算错误率来更新样本权重。
* SAMME.R: 这个算法是SAMME 算法的一种变体也是AdaBoostClassifier的默认集成策略，它采用加权概率估计的方法更新样本权重。每个基分类器的输出从标签变成了每个标签的概率，再通过得到的概率向量更新样本的权重。

在上文（[2.1.2节](#_关于AdaBoost)）中，我使用了默认的集成策略（SAMME.R），下面我将不改变其他参数，并尝试使用SAMME的集成策略，代码非常相似，我就直接贴结果了。

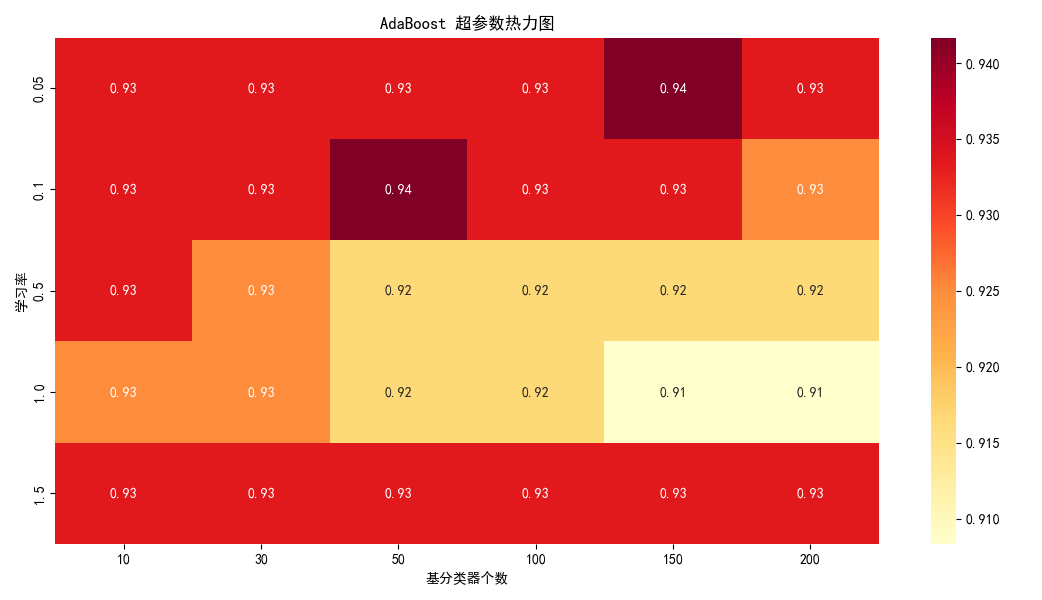


Figure 2 AdaBoost(SAMME) 超参数热力图

可以看出，SAMME的最佳准确率（0.94）比SAMME.R的最佳准确率（0.95）要低，而且我在测试集上测试的准确率为**90%**，低于SAMME.R的**96.667%**，说明SAMME.R是一个更优的集成策略。我后面的实验也将基于SAMME.R这个集成策略进行。

### 关于基分类器

在sklearn提供的AdaBoostClassifier中，默认的分类器是深度为1的决策树。我在上面使用的都是深度为2的决策树，下面我将使用SAMME.R集成策略，测试几个不同的基分类器：

**深度为1的决策树（决策桩）：**

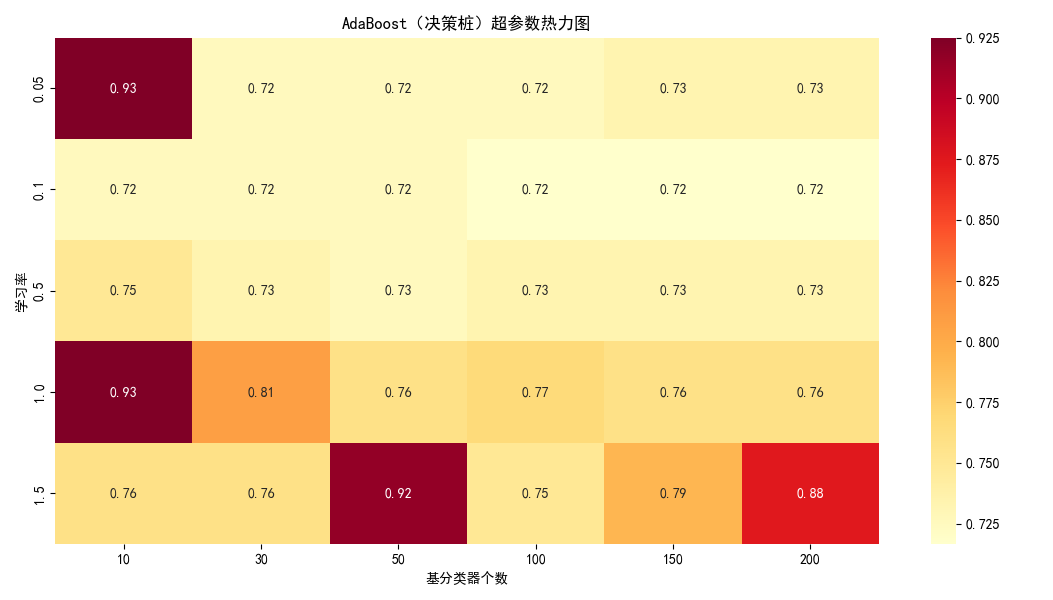


Figure 3 AdaBoost（决策桩）超参数热力图

深度为1的决策桩虽然是AdaBoostClassifier的默认参数，但是效果大部分不太行，但是在最优的参数下只比起前面的深度为2的决策树效果差了一些。5折交叉验证最高准确率达到了93%，但是在测试集上准确率和深度为2的决策树一样为96.67%，整体来说效果还是可以接受的。

**深度为5的决策树：**

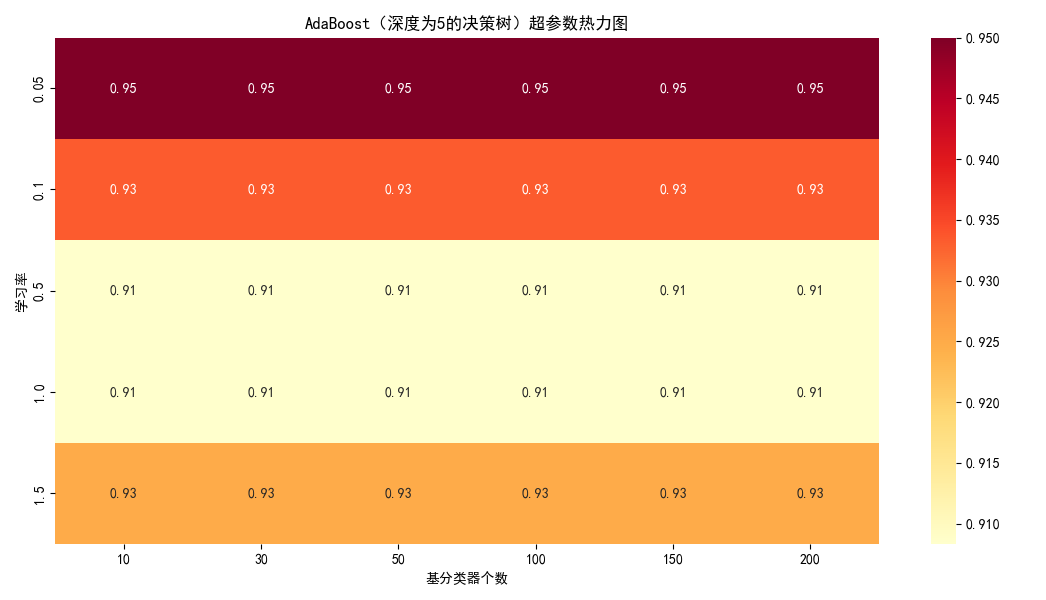


Figure 4 AdaBoost（深度为5的决策树）超参数热力图

深度为5的决策树得益于更强的数据区分能力，结果比前面深度为1和2的决策树都要稳定，但是在最优的参数下也只打平了前面的深度为2的决策树。5折交叉验证最高准确率达到了95%，但是在测试集上准确率和深度为2的决策树一样为96.67%，整体来说效果还可以。

**逻辑回归：**

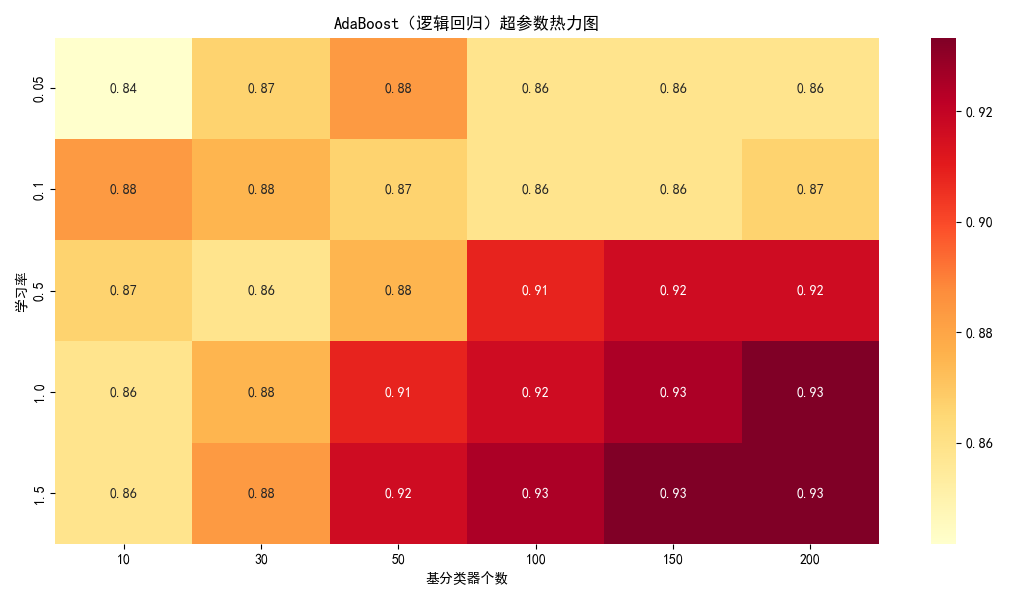


Figure 5 AdaBoost（逻辑回归）超参数热力图

使用逻辑回归的话很明显使用的基分类器越多，准确率越高。说明单个逻辑回归分类器的表达能力有限，需要多个逻辑回归来对数据进行更细致的拟合。最后在测试集的测试结果也是93%，并不如深度分别为1，2和5的决策树。

**逻辑回归：**

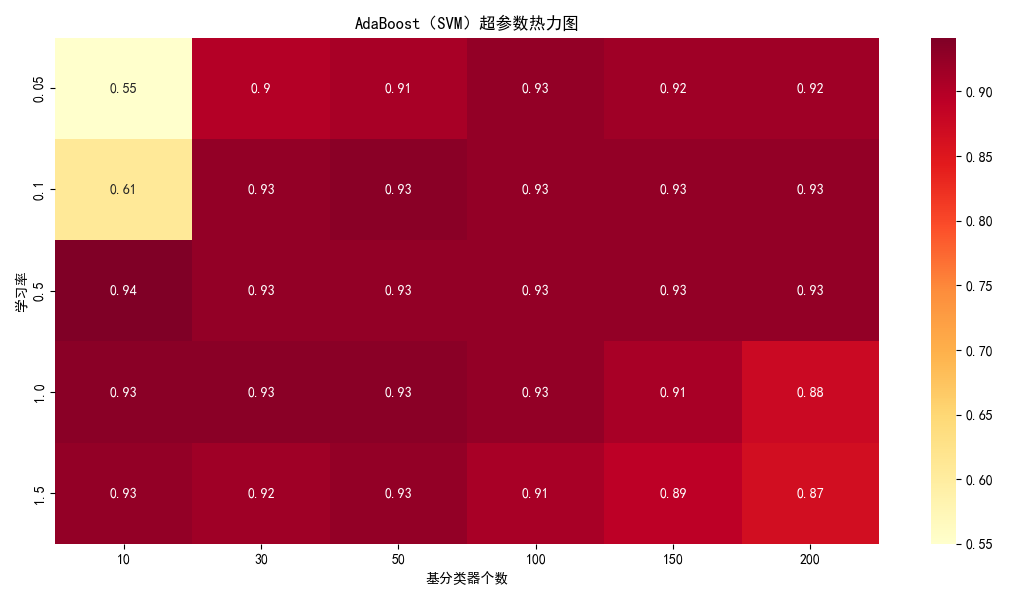


Figure 6 AdaBoost（SVM）超参数热力图

SVM在各个超参数的表现下都不错，但是在分类器个数非常多的时候还是出现了过拟合的问题。最优的参数出现在10个分类器的时候，虽然在5折交叉验证的时候最高正确率只有**94%**，但是最后在测试集上拿下了**100%** 的正确率。

**朴素贝叶斯：**

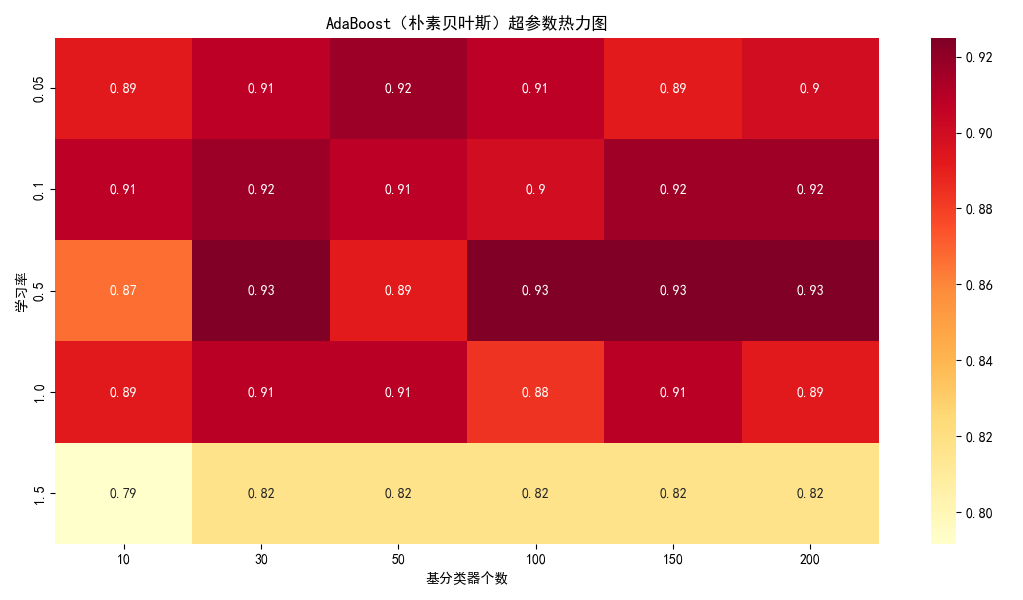


Figure 7 AdaBoost（朴素贝叶斯）超参数热力图

朴素贝叶斯在学习率小于等于1的时候表现都还行，但是都差强人意。最优的参数出现在100个分类器的时候，说明朴素贝叶斯的表现能力有限，需要比较多的分类器才能比较好的拟合。

### 总结不同分类器对性能的影响

下面是我的整体结果：

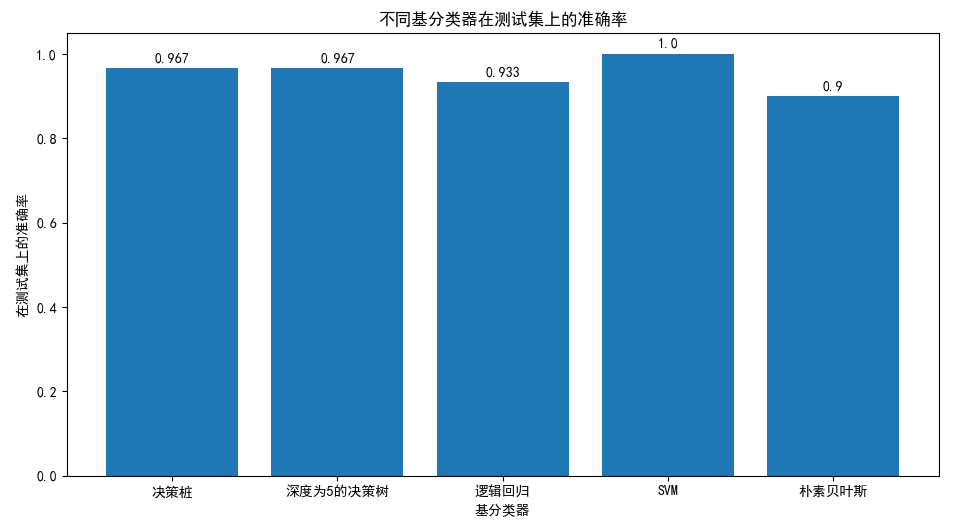


Figure 8 不同基分类器在测试集上的表现

可以看到，在AdaBoost的SAMME.R集成策略的加持下，不论使用哪个基分类器的效果都不错，都能达到90%以上的准确率。特别是SVM作为基分类器的时候，达成了100% 的准确率。当然，我也很好奇这些基分类器不集成在测试集上的表现如何，下面是我尝试的结果：

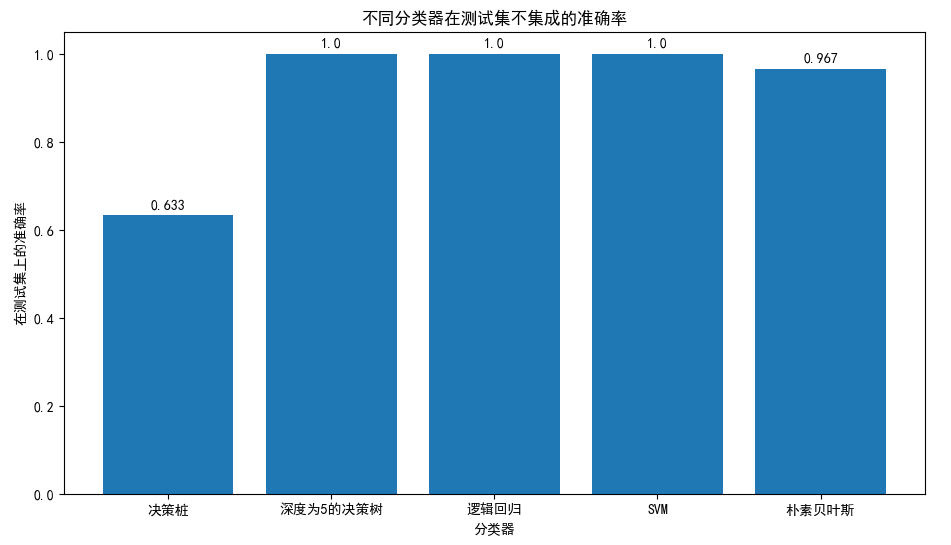


Figure 9 不同分类器在测试集不集成的准确率

可以看到，在仅使用一个分类器的时候，只有决策桩因为表达能力的问题准确率非常低，其他的分类器都超过了经过AdaBoost集成的版本。我认为应该是由于在使用AdaBoost的时候由于迭代次数过多，以及使用的分类器过多而产生的过拟合导致准确率的下降。

## 题目三：讨论和图示基分类器的偏差和方差

### 关于偏差和方差

* 偏差（bias）：模型预测的期望值（或平均值）与正确值之间的差异。也就是
* 方差(Variance)：模型对某个数据点的预测结果的可变性，使用不同模型对同一数据点的预测值可变性。

下面是我看到的一张解释很清楚的图标：

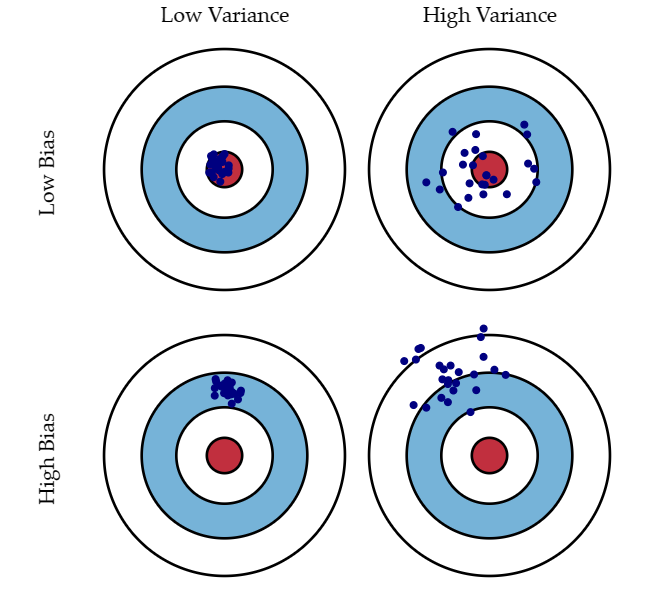


Figure 10 图示偏差和方差[2]

### 计算基分类器的偏差和方差

在计算偏差和方差的时候，我选择了mlxtend库提供的bias\_variance\_decomp函数。这个函数通过从训练集中随机多次（默认200次）随机选择数据并训练模型，然后在测试集上进行测试，再按照上面提到的两个公式计算出偏差和方差。于是我得到了下面的结果：

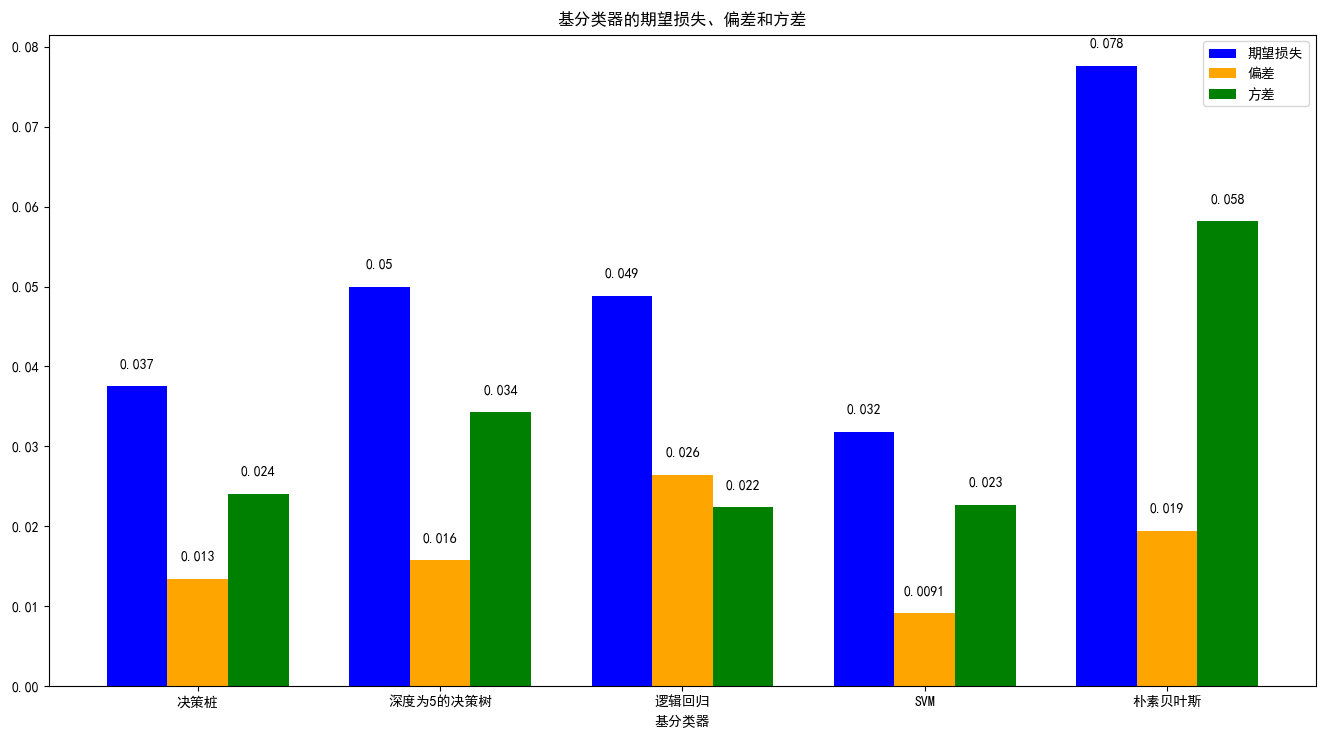


Figure 11 基分类器的期望损失、偏差和方差

除了朴素贝叶斯，感觉其他的基分类器表现都比较相似。比如SVM作为在上文的测试中表现最好的模型，偏差和方差都是最低的，但是和其他模型也没有拉开特别大的差距。

# 总结

综上所述，对于表现能力不佳的模型，比如决策桩，使用AdaBoost能够大幅提升整体的准确度。但是如果使用表现力比较强的模型进行集成的话，有时候可能会因为过拟合等问题导致反而不如不集成而仅使用单个分类器的情况。并且经过一系列的测试，决策桩的整体效果是很不错的，我认为达到了一个正确率和计算效率的平衡点。

这次实验花费了我非常多的时间和精力，从复习上个学习学过的各种集成方式的原理到可视化模型的方法，到思考模型的测试策略。每一个环节都十分重要，我也都遇到了一些困难：

这次我花费最多时间解决的问题就是研究SAMME和SAMME.R的实现方式[1]，以及它们的不同。之前学的AdaBoost都是二分类的，SAMME使用了一种很巧妙的方式解决了多分类的问题，让我醍醐灌顶。

这次的实验让我受益匪浅，使我对sklearn库的理解更进一步的同时也锻炼了我解决问题的能力。上面提到的两个库都是在机器学习领域非常重要的工具，只有用好他们才能更好的完成实验，并理解这些模型的底层原理。我将继续保持学习，争取有朝一日能成为机器学习大师。

# 附件

## 完整代码

### 测试SAMME和SAMME.R在深度为2的决策树上的效果ada\_boost\_DT.py

import utils

# import numpy as np

from sklearn.ensemble import AdaBoostClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import rcParams

# 设置中文字体

rcParams["font.family"] = "SimHei"

# 解决负号问题

rcParams["axes.unicode\_minus"] = False

X, y, title = utils.load\_data("data/iris.csv")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

algorithms = ["SAMME", "SAMME.R"]

for algorithm in algorithms:

    # 基学习器（深度为2的决策树）

    base\_estimator = DecisionTreeClassifier(max\_depth=2)

    # 建立AdaBoost分类器

    clf = AdaBoostClassifier(

        estimator=base\_estimator, random\_state=42, algorithm=algorithm

    )

    # 参数字典

    param\_grid = {

        "n\_estimators": [10, 30, 50, 100, 150, 200],

        "learning\_rate": [0.05, 0.1, 0.5, 1.0, 1.5],

    }

    # 构建网格搜索器

    grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring="accuracy", n\_jobs=-1)

    grid\_search.fit(X\_train, y\_train)

    # 使用最佳参数进行预测

    best\_clf = grid\_search.best\_estimator\_

    y\_pred = best\_clf.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    print("Accuracy:", accuracy)

    # 输出最佳模型的参数和性能指标

    print("Best Parameters:", grid\_search.best\_params\_)

    print("Best Score:", grid\_search.best\_score\_)

    # 绘制热力图

    param\_scores = grid\_search.cv\_results\_["mean\_test\_score"].reshape(

        len(param\_grid["learning\_rate"]), len(param\_grid["n\_estimators"])

    )

    # param\_grid = np.round(np.array(param\_grid), 2)

    # print(grid\_search.cv\_results\_["mean\_test\_score"])

    fig, ax = plt.subplots(figsize=(11, 6))

    sns.heatmap(

        param\_scores,

        annot=True,

        cmap="YlOrRd",

        vmin=0,

        vmax=1,

        yticklabels=param\_grid["learning\_rate"],

        xticklabels=param\_grid["n\_estimators"],

    )

    ax.set\_title("AdaBoost 超参数热力图")

    ax.set\_xlabel("基分类器个数")

    ax.set\_ylabel("学习率")

    plt.tight\_layout()

    plt.show()

### 测试在SAMME.R下不同基分类器的表现并画图ada\_boost\_varies.py

import utils

import numpy as np

from sklearn.ensemble import AdaBoostClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

import os

from matplotlib import rcParams

from mlxtend.evaluate import bias\_variance\_decomp

# from sklearn.model\_selection import cross\_val\_score

# 设置中文字体

rcParams["font.family"] = "SimHei"

# 解决负号问题

rcParams["axes.unicode\_minus"] = False

os.environ["PYTHONWARNINGS"] = "ignore"

X, y, title = utils.load\_data("data/iris.csv")

labels = np.unique(y)

label\_to\_int = {label: i for i, label in enumerate(labels)}

y = np.array([label\_to\_int[label] for label in y])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

# 基学习器数组

base\_estimators = [

    DecisionTreeClassifier(max\_depth=1),  # 决策桩

    DecisionTreeClassifier(max\_depth=5),  # 深度为5的决策树

    LogisticRegression(),  # 逻辑回归

    SVC(probability=True),  # SVM

    GaussianNB(),  # 朴素贝叶斯

]

base\_estimators\_name = ["决策桩", "深度为5的决策树", "逻辑回归", "SVM", "朴素贝叶斯"]

# 作为基分类器的准确率

base\_estimators\_acc = []

# 直接分类的准确率

stand\_alone\_acc = []

# 预期损失，偏差和方差

list\_expected\_loss, list\_bias, list\_var = [], [], []

for base\_estimator, name in zip(base\_estimators, base\_estimators\_name):

    # 建立AdaBoost分类器

    clf = AdaBoostClassifier(estimator=base\_estimator, random\_state=42)

    # 参数字典

    param\_grid = {

        "n\_estimators": [10, 30, 50, 100, 150, 200],

        "learning\_rate": [0.05, 0.1, 0.5, 1.0, 1.5],

    }

    # 构建网格搜索器

    grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring="accuracy", n\_jobs=-1)

    grid\_search.fit(X\_train, y\_train)

    # 使用最佳参数进行预测

    best\_clf = grid\_search.best\_estimator\_

    y\_pred = best\_clf.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    # 储存当前模型的准确率

    base\_estimators\_acc.append(accuracy)

    print("Accuracy:", accuracy)

    # 输出最佳模型的参数和性能指标

    print("Best Parameters:", grid\_search.best\_params\_)

    print("Best Score:", grid\_search.best\_score\_)

    # 绘制热力图

    param\_scores = grid\_search.cv\_results\_["mean\_test\_score"].reshape(

        len(param\_grid["learning\_rate"]), len(param\_grid["n\_estimators"])

    )

    # 计算预期损失，偏差和方差

    avg\_expected\_loss, avg\_bias, avg\_var = bias\_variance\_decomp(

        best\_clf, X\_train, y\_train, X\_test, y\_test, random\_seed=42, loss="mse"

    )

    list\_expected\_loss.append(avg\_expected\_loss)

    list\_bias.append(avg\_bias)

    list\_var.append(avg\_var)

    # print(grid\_search.cv\_results\_["mean\_test\_score"])

    fig, ax = plt.subplots(figsize=(11, 6))

    sns.heatmap(

        param\_scores,

        annot=True,

        cmap="YlOrRd",

        # vmin=0,

        # vmax=1,

        yticklabels=param\_grid["learning\_rate"],

        xticklabels=param\_grid["n\_estimators"],

    )

    ax.set\_title(f"AdaBoost（{name}）超参数热力图")

    print(f"AdaBoost（{name}）超参数热力图")

    ax.set\_xlabel("基分类器个数")

    ax.set\_ylabel("学习率")

    plt.tight\_layout()

    # 测试不集成直接分类

    base\_estimator.fit(X\_train, y\_train)

    y\_pred = base\_estimator.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    stand\_alone\_acc.append(accuracy)

# plt.show()

# 绘制AdaBoost柱状图

fig, ax = plt.subplots(figsize=(8, 6))

# 绘制柱状图

bars = ax.bar(base\_estimators\_name, base\_estimators\_acc)

# 在柱子上标注数据

for bar, value in zip(bars, base\_estimators\_acc):

    height = bar.get\_height()

    ax.annotate(

        f"{value:.3}",

        xy=(bar.get\_x() + bar.get\_width() / 2, height),

        xytext=(0, 3),  # 3 points vertical offset

        textcoords="offset points",

        ha="center",

        va="bottom",

    )

# 设置标题和轴标签

ax.set\_title("不同基分类器在测试集使用AdaBoost(SAMME.R)的准确率")

ax.set\_xlabel("基分类器")

ax.set\_ylabel("在测试集上的准确率")

# 绘制使用单个分类器柱状图

fig, ax = plt.subplots(figsize=(8, 6))

# 绘制柱状图

bars = ax.bar(base\_estimators\_name, stand\_alone\_acc)

# 在柱子上标注数据

for bar, value in zip(bars, stand\_alone\_acc):

    height = bar.get\_height()

    ax.annotate(

        f"{value:.3}",

        xy=(bar.get\_x() + bar.get\_width() / 2, height),

        xytext=(0, 3),  # 3 points vertical offset

        textcoords="offset points",

        ha="center",

        va="bottom",

    )

# 设置标题和轴标签

ax.set\_title("不同分类器在测试集不集成的准确率")

ax.set\_xlabel("分类器")

ax.set\_ylabel("在测试集上的准确率")

# 创建图像和坐标轴

fig, ax = plt.subplots(figsize=(15, 8))

# 绘制三组柱状图

x = range(len(base\_estimators\_name))

width = 0.25

ax.bar(x, list\_expected\_loss, width, color="blue", label="期望损失")

ax.bar([i + width for i in x], list\_bias, width, color="orange", label="偏差")

ax.bar([i + 2 \* width for i in x], list\_var, width, color="green", label="方差")

# 在柱子上标注数据

for i, (v1, v2, v3) in enumerate(zip(list\_expected\_loss, list\_bias, list\_var)):

    ax.text(i, v1 + 0.002, f"{v1:.2}", ha="center", va="bottom")

    ax.text(i + width, v2 + 0.002, f"{v2:.2}", ha="center", va="bottom")

    ax.text(i + 2 \* width, v3 + 0.002, f"{v3:.2}", ha="center", va="bottom")

# 设置标题和坐标轴标签

ax.set\_title("基分类器的期望损失、偏差和方差")

ax.set\_xlabel("基分类器")

ax.set\_xticks([i + width for i in x])

ax.set\_xticklabels(base\_estimators\_name)

ax.legend()

# 显示图像

plt.show()

### 工具集合utils.py

import csv

import numpy as np

def load\_data(path):

    # 使用的特征数

    ATTR\_NUM = 3

    # 标签的列

    TAG\_COL = 4

    x, y = [], []

    with open(path, "r") as file:

        reader = csv.reader(file)

        title = next(reader)

        for row in reader:

            x.append(np.array(row[:ATTR\_NUM], dtype=float))

            y.append(row[TAG\_COL])

    x = np.array(x)

    y = np.array(y)

    return x, y, title[:ATTR\_NUM]

## 参考资料

[1] HASTIE T, ROSSET S, ZHU J, 等. Multi-class AdaBoost[J/OL]. Statistics and Its Interface, 2009, 2(3): 349-360. DOI:10.4310/SII.2009.v2.n3.a8.

[2] Understanding the Bias-Variance Tradeoff[EB/OL]. [2024-04-13]. https://scott.fortmann-roe.com/docs/BiasVariance.html.