《**数据挖掘技术**》实验考核表

实验名称： 实验二：数据挖掘算法（2）- 数据分类

2020级 专业： 计算机科学与技术

实验时间： 2023年4月20日 3-4节中午 实验地点： 7-212

表一

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学号 | 姓名 | 班级 | 实验摘要与总结（学生填写） | 成绩 |
| **2020** |  | 计算机8班 | 在本实验中，我进行了各种机器学习基础分类模型的使用。并通过生成一系列可视化图像来更好的理解其运作模式。 |  |

表二

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 考核标准 | 实验过程  30分 | 实验报告  30分 | 程序运行  20分 | 交流合作  10分 | 考勤纪律  10分 |
| 各项成绩 |  |  |  |  |  |
| 考核内容 | 评价实验课堂中的综合表现，包括实验态度、实验过程等内容。 | ○完整  ○较完整  ○一般  ○内容极少 | ○运行正常  ○功能不全  ○有小错  ○无法运行 | ○良好  ○一般  ○较差 | ○良好  ○一般  ○较差 |

表三

|  |
| --- |
| 其它批改意见 |
|  |

注：

* 表1除“成绩”栏外由学生填写，表2、表3由指导教师填写。
* 本页附在实验报告正文前装订，纸质版报告可单双面打印，**电子报告Word排版**。
* **实验报告如发现雷同，一律以0分计。**

**一、实验目的**

1. 理解机器学习中分类的基本概念；
2. 掌握支持向量机和集成学习进行分类的思想和计算流程；
3. 掌握使用工具包进行数据分类；
4. 能够对数据分类结果进行分析。
   1. **实验内容**
5. 数据来源：

行人：http://cbcl.mit.edu/software-datasets/PedestrianData.html

http://cbcl.mit.edu/projects/cbcl/software-datasets/pedestrians128x64.tar.gz

负样本：http://cvcl.mit.edu/database.htm

1. 数据预处理，可以选用各种技术，生成平衡的训练样本和测试样本数据集；
2. 支持向量机算法：（1）使用scikit中线性支持向量机函数对数据集进行训练并测试；（2）使用GridSearchCV, 调整核函数类型和参数，寻找较优化的模型和参数组合；（3）比较不同参数的准确率图；（4）绘制最优参数模型下的混淆矩阵。
3. 集成算法：使用sklearn包中以下方法进行分类（可以使用预处理技术）：（1）随机森林；（2）AdaBoost算法；（3）梯度提升算法；（4）选出较优的模型参数，分类问题打印该参数的混淆矩阵；（4）分析弱分类器的数目对准确率的影响：选择较优的算法，在一张图中按如下要求绘制：横坐标为弱分类器的数目，选择不同弱分类器数量（如10,50,100,500,1000,…），纵坐标为分类准确率。
4. 选择软间隔支持向量机或AdaBoost算法的使用流程图或伪代码方式描述计算过程；
5. **（选做）**Xgboost算法，参考 https://xgboost.readthedocs.io/en/latest/ 安装Xgboost包，计算结果，并对比结果；
6. **（选做）**LightGBM算法，参考 https://lightgbm.readthedocs.io/en/latest/ 安装LightGBM包，计算结果，并对比结果。

**三、实验要求**

实验报告中体现全部基本实验内容.，至少包括：

1. 支持向量机和集成学习中较优化的参数组合；
2. 集成学习中弱分类器的数目对准确率的影响分析图；
3. SVM或AdaBoost流程图或伪代码图；
4. 实验总结。

**四、实验记录**

（以下部分按实验要求填写，序号对应实验内容，只能粘贴核心程序语句。）

1. 数据预处理：

import numpy as np

import random

import cv2

import matplotlib.pyplot as plt

import os

from sklearn import model\_selection as ms

datadir = "lab2\_data"

dataset = "pedestrians128x64"

datafile = "%s/%s.tar.gz" % (datadir, dataset)

extractdir = "%s/%s" % (datadir, dataset)

def extract\_tar(datafile, extractdir):

try:

import tarfile

except ImportError:

raise ImportError("You do not have tarfile installed. "

"Try unzipping the file outside of Python.")

tar = tarfile.open(datafile)

tar.extractall(path=extractdir)

tar.close()

print("%s successfully extracted to %s" % (datafile, extractdir))

extract\_tar(datafile, datadir)

plt.figure(figsize=(10, 6))

for i in range(5):

filename = "%s/per0010%d.ppm" % (extractdir, i)

img = cv2.imread(filename)

plt.subplot(1, 5, i + 1)

plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()

win\_size = (48, 96)

block\_size = (16, 16)

block\_stride = (8, 8)

cell\_size = (8, 8)

num\_bins = 9

hog = cv2.HOGDescriptor(win\_size, block\_size, block\_stride, cell\_size, num\_bins)

random.seed(42)

X\_pos = []

for i in random.sample(range(900), 150):

filename = "%s/per%05d.ppm" % (extractdir, i)

img = cv2.imread(filename)

if img is None:

print('Could not find image %s' % filename)

continue

X\_pos.append(hog.compute(img, (64, 64)))

print(len(X\_pos))

X\_pos = np.array(X\_pos, dtype=np.float32)

y\_pos = np.ones(X\_pos.shape[0], dtype=np.int32)

print("X\_pos.shape:")

print(X\_pos.shape)

print("y\_pos.shape:")

print(y\_pos.shape)

# plt.show(X\_pos)

print("X\_pos[1, :]:")

print(X\_pos[1, :])

print("y\_pos[1]:")

print(y\_pos[1])

print("y\_pos:")

print(y\_pos)

negset = "pedestrians\_neg"

negfile = "%s/%s.tar.gz" % (datadir, negset)

negdir = "%s/%s" % (datadir, negset)

extract\_tar(negfile, datadir)

hroi = 128

wroi = 64

X\_neg = []

for negfile in os.listdir(negdir):

filename = '%s/%s' % (negdir, negfile)

img = cv2.imread(filename)

img = cv2.resize(img, (512, 512))

for j in range(5):

rand\_y = random.randint(0, img.shape[0] - hroi)

rand\_x = random.randint(0, img.shape[1] - wroi)

roi = img[rand\_y:rand\_y + hroi, rand\_x:rand\_x + wroi, :]

X\_neg.append(hog.compute(roi, (64, 64)))

X\_neg = np.array(X\_neg, dtype=np.float32)

y\_neg = -np.ones(X\_neg.shape[0], dtype=np.int32)

print("X\_neg.shape:")

print(X\_neg.shape)

print("y\_neg.shape:")

print(y\_neg.shape)

X = np.concatenate((X\_pos, X\_neg))

y = np.concatenate((y\_pos, y\_neg))

print("X.shape:")

print(X.shape)

X\_train, X\_test, y\_train, y\_test = ms.train\_test\_split(X, y, test\_size=0.2, random\_state=42)

1. 支持向量机算法：

下列代码为了确保展示简洁，仅仅截取了核心代码，去除了一些如输出准确率矩阵、热力图等部分。其中训练了支持向量机中，线性linear、多项式poly、径向基RBF三种核函数，找到了其在一定区间内的最佳参数，并输出了由最佳参数训练后的模型的准确率。

# 绘制热力图

def heatmap(modelname, scores, param\_grid):

# 使用 matplotlib 绘制准确率热力图

plt.figure(figsize=(8, 6))

plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot)

plt.xlabel('gamma')

plt.ylabel('C')

plt.colorbar()

plt.xticks(np.arange(len(param\_grid['gamma'])), param\_grid['gamma'], rotation=45)

plt.yticks(np.arange(len(param\_grid['C'])), param\_grid['C'])

plt.title(modelname + 'poly accuracy')

plt.show()

print('ok')

# 绘制混淆矩阵

def con\_matrix(name, model):

# 使用模型对测试数据进行预测

y\_pred = model.predict(X\_test)

# 计算混淆矩阵

cm = confusion\_matrix(y\_test, y\_pred)

# 使用 seaborn 绘制混淆矩阵

plt.figure(figsize=(7, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('predict label')

plt.ylabel('true label')

plt.title('Confusion Matrix of ' + name)

plt.show()

# 训练线性核模型

param\_grid\_linear = {'C': np.logspace(-3, 3, 7),

'gamma': np.logspace(-3, 3, 7)}

grid\_linear = GridSearchCV(svm.SVC(kernel="linear"),param\_grid\_linear, cv=5)

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

scores\_linear = grid\_linear.cv\_results\_['mean\_test\_score'].reshape(len(param\_grid\_linear['C']), len(param\_grid\_linear['gamma']))

# 使用最佳准确率训练模型

best\_linear = svm.SVC(kernel='linear', C=grid\_linear.best\_params\_['C'], gamma=grid\_linear.best\_params\_['gamma'])

best\_linear.fit(X\_train, y\_train)

print("最佳线性核模型准确率为: %f" % best\_linear.score(X\_test, y\_test))

# 训练多项式核模型

param\_grid\_poly = {'C': np.logspace(-3, 3, 7),

'gamma': np.logspace(-3, 3, 7)}

grid\_poly = GridSearchCV(svm.SVC(kernel="poly"), param\_grid\_poly, cv=5)

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

scores\_poly = grid\_poly.cv\_results\_['mean\_test\_score'].reshape(len(param\_grid\_poly['C']), len(param\_grid\_poly['gamma']))

# 使用最佳准确率训练模型

best\_poly = svm.SVC(kernel='poly', C=grid\_poly.best\_params\_['C'], gamma=grid\_poly.best\_params\_['gamma'])

best\_poly.fit(X\_train, y\_train)

print("最佳poly核模型准确率为: %f" % best\_poly.score(X\_test, y\_test))

# 下面是RBF核函数的训练

param\_grid = {'C': np.logspace(-1, 5, 7),

'gamma': np.logspace(-5, 1, 7)}

# 使用 GridSearchCV 训练模型,其中param\_grid中存储了各种参数的组合。其中cv5是交叉验证的折数

grid = GridSearchCV(svm.SVC(kernel='rbf'), param\_grid, cv=5)

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

# 这里得到了最佳的准确率

scores = grid.cv\_results\_['mean\_test\_score'].reshape(len(param\_grid['C']), len(param\_grid['gamma']))

# 使用最佳准确率训练模型

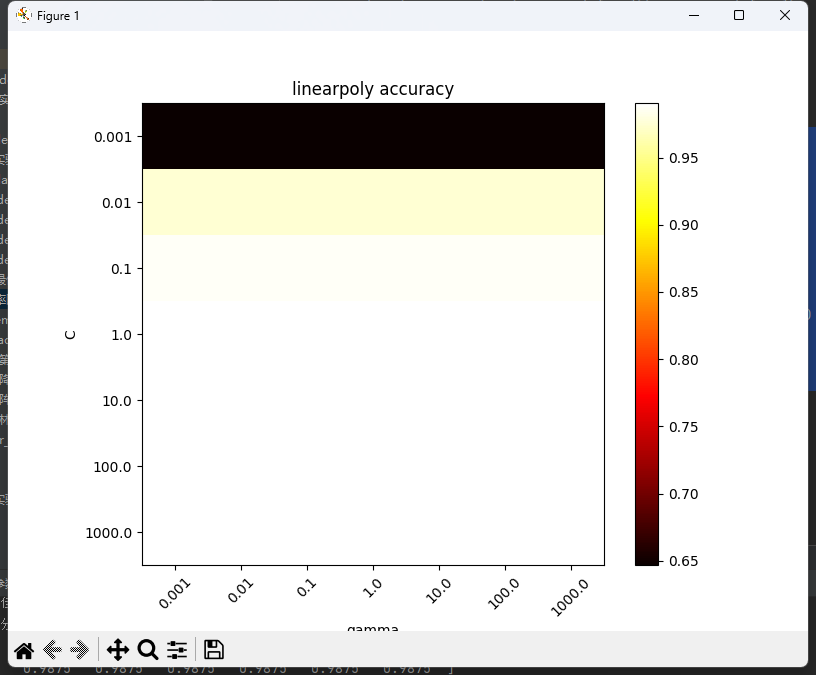
best\_clf = svm.SVC(kernel='rbf', C=grid.best\_params\_['C'], gamma=grid.best\_params\_['gamma'])

best\_clf.fit(X\_train, y\_train)

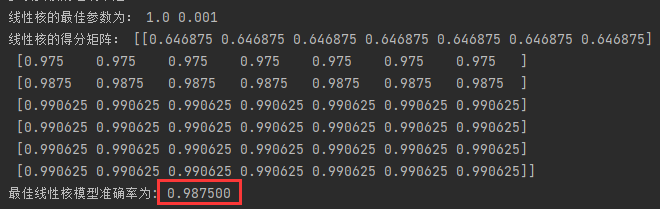
score\_new = best\_clf.score( X\_test, y\_test)

print("最佳RBF核模型准确率为: %f" % score\_new)

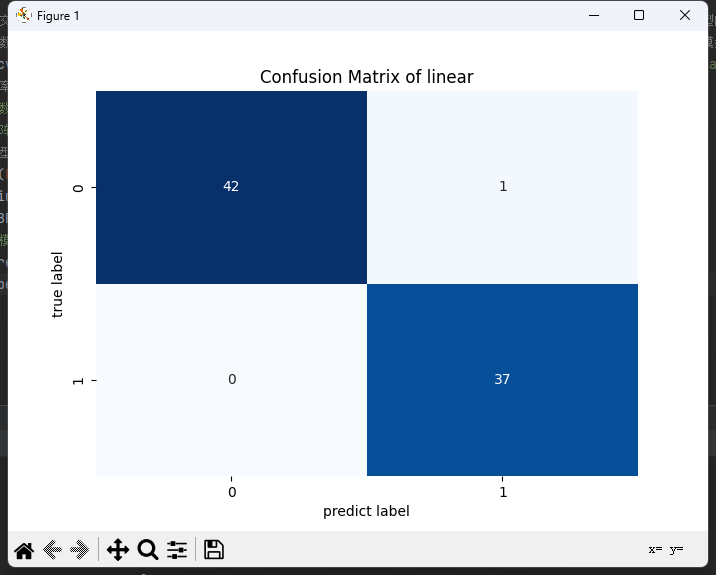
线性核心的热力图：



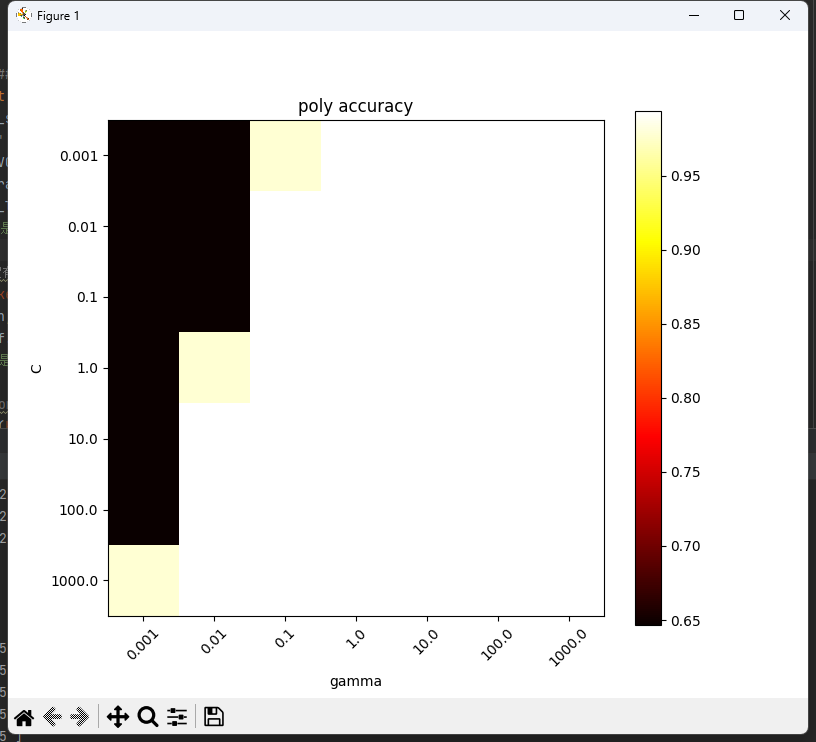
最佳参数搜索结果：



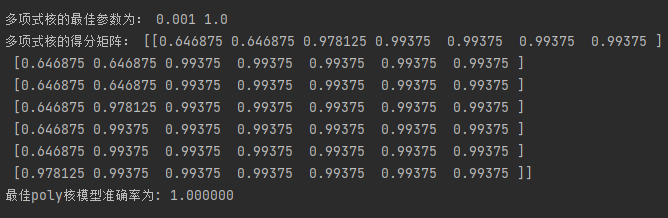
混淆矩阵：



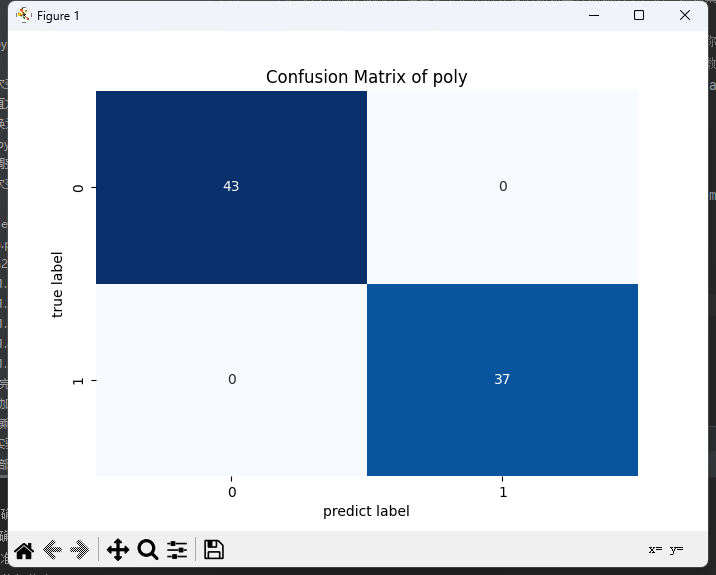
Poly核模型的热力图如下：



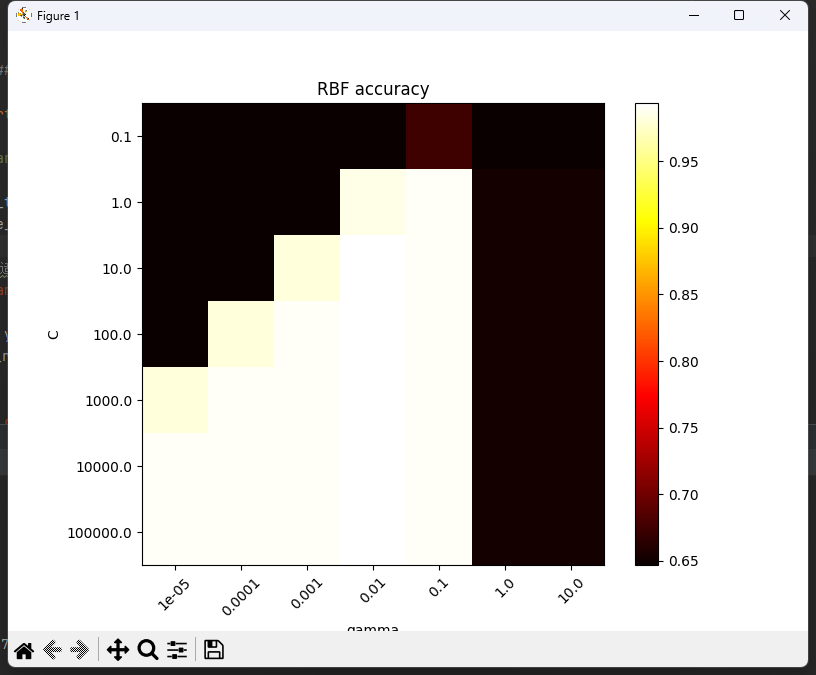
经过最佳参数搜索后，多项式核的模型在测试集上的准确率从最初的0.5375上升到了1。



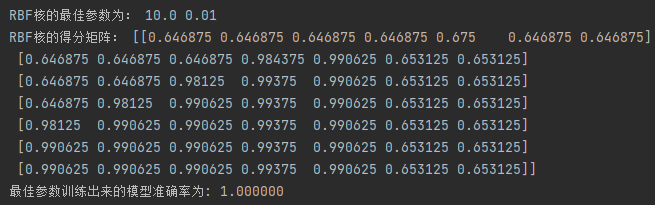
混淆矩阵：



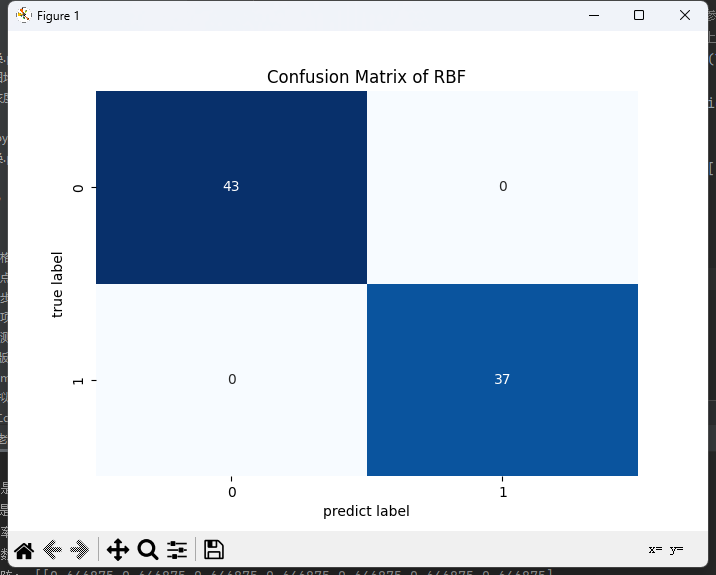
RBF核的热力图如下：



经过最佳参数搜索后，RBF核分类器的训练集准确率从0.5375上升到了1。



混淆矩阵：



1. 集成算法：

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

from sklearn.model\_selection import GridSearchCV

# 随机森林

# 参数'n\_estimators'代表着弱分类器的数量

param\_grid\_rf = {

'n\_estimators': [50, 100],

'max\_depth': [None, 10],

'min\_samples\_split': [2, 5],

'min\_samples\_leaf': [1, 2]

}

rf = RandomForestClassifier(random\_state=42)

grid\_rf = GridSearchCV(rf, param\_grid\_rf, cv=5)

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

best\_rf = RandomForestClassifier(n\_estimators=grid\_rf.best\_params\_['n\_estimators'],

max\_depth=grid\_rf.best\_params\_['max\_depth'],

min\_samples\_split=grid\_rf.best\_params\_['min\_samples\_split'],

min\_samples\_leaf=grid\_rf.best\_params\_['min\_samples\_leaf'],

random\_state=42)

best\_rf.fit(X\_train, y\_train)

# AdaBoost

param\_grid\_ada = {

'n\_estimators': [50, 100, 200],

'learning\_rate': [0.01, 0.1, 1]

}

ada = AdaBoostClassifier(random\_state=42)

grid\_ada = GridSearchCV(ada, param\_grid\_ada, cv=5)

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

best\_ada = AdaBoostClassifier(n\_estimators=grid\_ada.best\_params\_['n\_estimators'],

learning\_rate=grid\_ada.best\_params\_['learning\_rate'],

random\_state=42)

best\_ada.fit(X\_train, y\_train)

# 梯度提升

param\_grid\_gb = {

'n\_estimators': [50, 100],

'learning\_rate': [0.1, 0.5, 1.0],

'max\_depth': [3, 5, 10]

}

gb = GradientBoostingClassifier(random\_state=42)

grid\_gb = GridSearchCV(gb, param\_grid\_gb, cv=5)

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

best\_gb = GradientBoostingClassifier(n\_estimators=grid\_gb.best\_params\_['n\_estimators'],

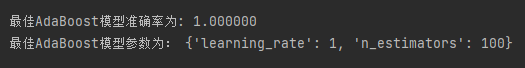
learning\_rate=grid\_gb.best\_params\_['learning\_rate'],

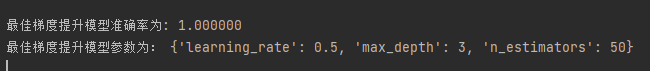
max\_depth=grid\_gb.best\_params\_['max\_depth'],

random\_state=42)

best\_gb.fit(X\_train, y\_train)

最佳参数搜寻训练结果：

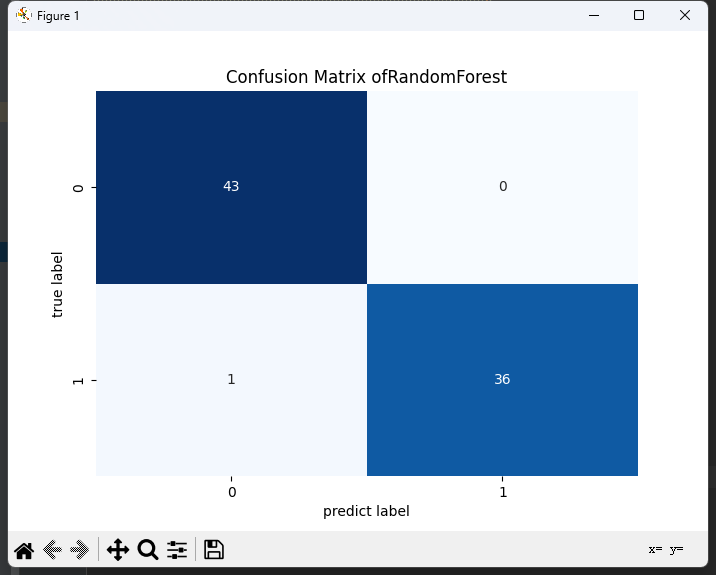




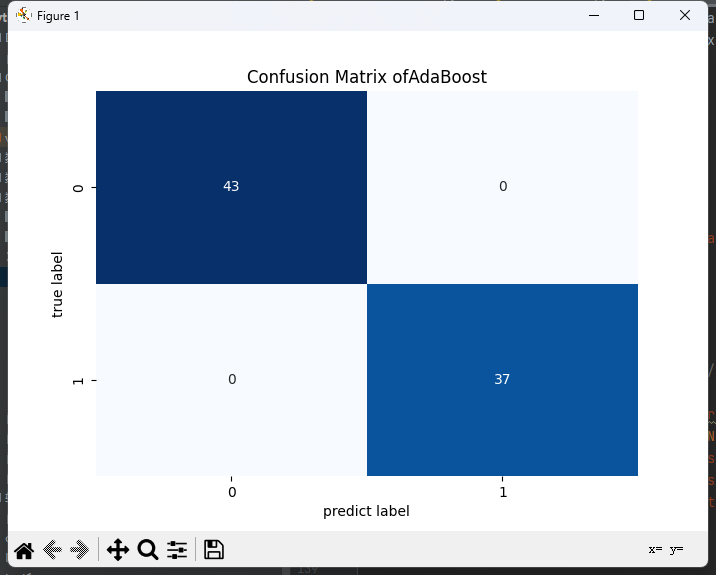


使用其最佳参数训练模型，并输出其混淆矩阵：

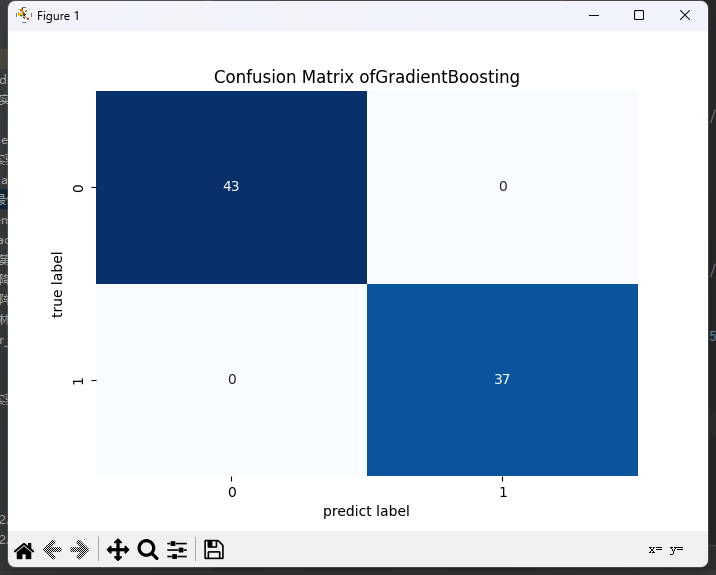
随机森林：



Adaboost：

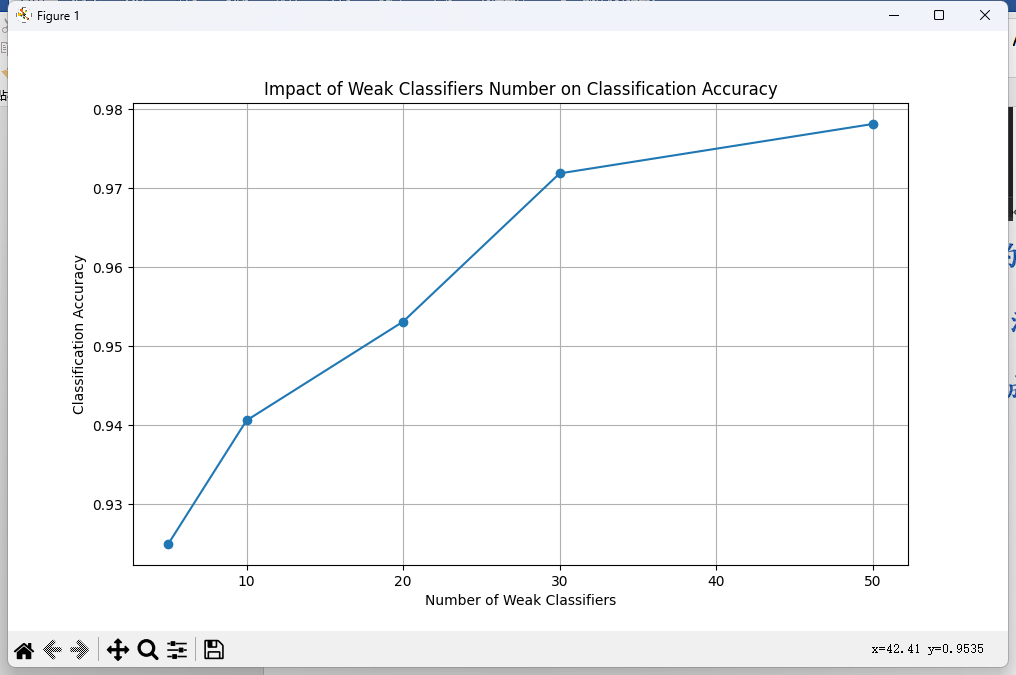


梯度提升：

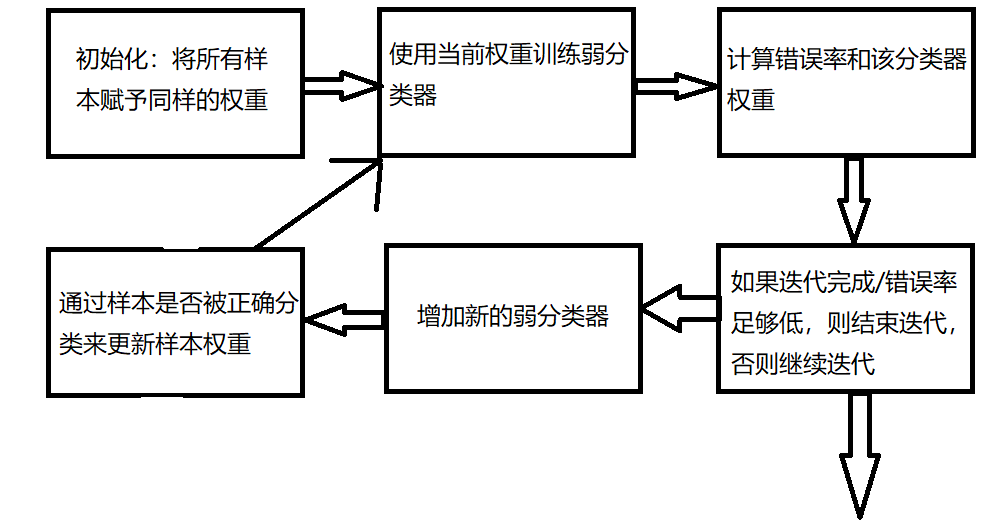


根据不同的弱分类器的数量下的准确率绘制准确率变化图，在这里我选取了实际表现最好的梯度提升算法。由于数据集只有400，因此弱分类器数量不宜过多，否则很容易造成过拟合。因此此处我选择了从[5, 10, 20, 30, 50]的弱分类器数量进行绘制。

可以看到，随着弱分类器的数量提升，预测准确率也在上升，在弱分类器数量达到50时，准确率已经上升至接近0.98。



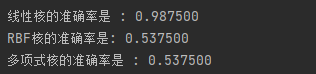
1. 集成学习方法AdaBoost的计算过程的核心在于反复迭代，每一次迭代都在训练一个弱分类器，然后根据这个弱分类器的错误率来更新数据的权重，并用这个更新后的数据训练下一个弱分类器。流程图如下：



1. (选做)
2. (选做)

**五、实验总结**

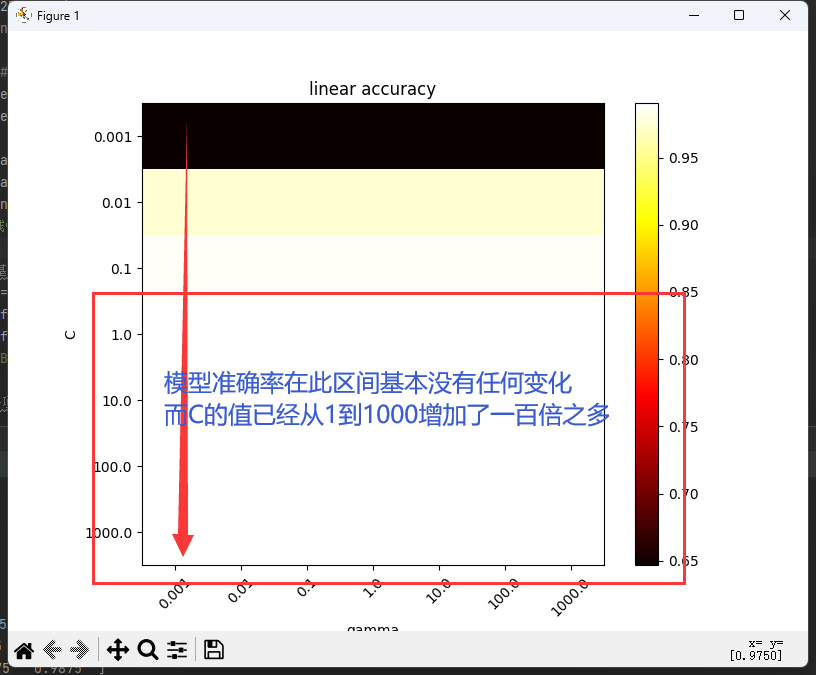
在本实验中，我首先不使用GridSearchCV进行最佳参数的搜寻。而仅是让每个核函数以如“clf\_rbf = svm.SVC(kernel='rbf', gamma='auto')”的格式自动选择其gamma值，最终得到的训练效果如下，可以看到不甚理想：



之后，我采用GridSearchCV来进行最佳参数的搜寻。首先，我利用numpy的logspace方法创建了一个参数字典，这个方法构造了一个等比数列，它生成了一个在对数尺度上均匀分布的数值。例如在代码段“param\_grid = {'C': np.logspace(-1, 5, 7), 'gamma': np.logspace(-5, 1, 7)}”中，它实际上生成了一个从从10^(-1) 和 10^(5) 之间分布的C值序列。和一个从10^(-5) 和 10^(1) 之间分布的gamma值。

这里之所以使用等比数列的生成方式，而非是在一定范围内的等差数列。是因为这样可以更迅速的测试模型在不同数量级的参数上的表现。实际上，在我之后的实验过程中，我也发现了这样一个事实：很多时候参数的细微变化并不能明显影响模型的性能表现。而在参数的范围超过一定限度后，甚至已经完全不能影响模型的性能。

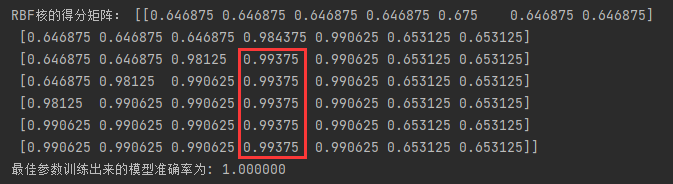
下图是以线性模型生成的热力图举例，从图中可以明显看到，C值超过1后，模型的准确率已经稳定趋于1.0，直到1000都没有太大变化。如果用线性搜索则要更多的计算量才能发现这种规律，而用等比数列的搜索可以更迅速地发现这种规律。



经过最佳参数的搜索后，线性核模型的准确率似乎和最初没有太大变化。这可能是线性核是一种比较简单的模型导致的，也有可能是数据集规模不够大导致的。

这里产生了一个疑问：最佳参数训练出来的模型准确率为: 1.00，而问题在于，它在训练过程中始终**没有一次**准确率是达到1的，但是在我测试的时候准确率却是1。

经过查询相关资料，我得到了一种推论：首先，GridSearchCV 的评分是通过交叉验证（在这个例子中是5折交叉验证）获得的，这就意味着接下来打印出的所有分数，都只是在一些独立子集上进行训练获得的，而这些子集每一个并不足以完全反映训练集的全貌，我们可以看到这里打印出来的平均准确率最高只有0.99375。



而在最后我们使用最佳参数训练模型的时候，模型终于是使用完整的训练集去训练的了，这些训练集得以反映数据分布的全貌，因此最后获得比单个子集更好的训练效果也是理所当然的。

而在得出这个结论的过程中，我发现了我之前一直存在的一个认知错误：我在之前的实验中一直以为交叉验证是用来为我已经训练好的模型进行打分的。但实际上，交叉验证所评判的**并非**是模型的性能高低，而是模型的**参数**选的好不好。**正确用法应该是：通过交叉验证来找到最优参数，再使用最优参数来训练模型。**

而交叉验证之所以要把数据分成多份进行交叉的训练和测试。就是为了验证模型在陌生数据上的表现，这样得出的分数才能够反映当前的参数能否让模型取得最好的泛化能力。

实际上在测试集的结果上我还能发现另一个问题：**数据集的规模可能太小了**，这从测试集的大小只有80就可以看出来。随之而来的结果是，poly核和RBF核经过最优参数搜索后，可以轻易的达到1的准确率。而linear核的模型即使使用默认参数，其准确率也能达到0.987之高。过小的数据集有可能导致模型对于参数的敏感度降低。这也能从得分矩阵中，随着参数的变化，模型准确率常常毫无变化的现象推测出来。

集成算法的模型拥有比支持向量机更大的复杂度，同时其参数的选择也更会显著影响它们的训练耗时。比如随机森林的深度、弱分类器的数量等。实验报告要求选择不同弱分类器数量（如10,50,100,500,1000,…）。但是根据在实验中的实际体验，400的训练集可能并不足以支撑上百的弱分类器的训练。

在本实验中，我学习到了各种机器学习基础分类模型的使用方法。这些分类模型仍然是机器学习中非常基础的部分，但是通过对它们的学习我也对机器学习整个体系的构筑模式有了更深刻的感受。

**六、代码附录**

（如有必要，此处可以粘贴完整的代码。）

以下是支持向量机相关的代码，去除了数据的预处理部分：

from sklearn import svm

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import confusion\_matrix

import seaborn as sns

# 绘制热力图

def heatmap(modelname, scores, param\_grid):

# 使用 matplotlib 绘制准确率热力图

plt.figure(figsize=(8, 6))

plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot)

plt.xlabel('gamma')

plt.ylabel('C')

plt.colorbar()

plt.xticks(np.arange(len(param\_grid['gamma'])), param\_grid['gamma'], rotation=45)

plt.yticks(np.arange(len(param\_grid['C'])), param\_grid['C'])

plt.title(modelname + 'poly accuracy')

plt.show()

print('ok')

# 绘制混淆矩阵

def con\_matrix(name, model):

# 使用模型对测试数据进行预测

y\_pred = model.predict(X\_test)

# 计算混淆矩阵

cm = confusion\_matrix(y\_test, y\_pred)

# 使用 seaborn 绘制混淆矩阵

plt.figure(figsize=(7, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('predict label')

plt.ylabel('true label')

plt.title('Confusion Matrix of ' + name)

plt.show()

# kernel = 'linear'

clf\_linear = svm.SVC(kernel='linear', gamma='auto')

clf\_linear.fit(X\_train, y\_train)

score\_linear = clf\_linear.score(X\_test, y\_test)

print("线性核的准确率是 : %f" % score\_linear)

# RBF径向基核函数，仅仅有gamma一个参数，适用于各种类型的数据，也是默认的核函数？

clf\_rbf = svm.SVC(kernel='rbf', gamma='auto')

clf\_rbf.fit(X\_train, y\_train)

score\_rbf = clf\_rbf.score(X\_test, y\_test)

print("RBF核的准确率是: %f" % score\_rbf)

# poly指多项式（Polynomial）核函数。

clf\_poly = svm.SVC(kernel='poly', gamma='auto')

clf\_poly.fit(X\_train, y\_train)

score\_poly = clf\_poly.score(X\_test, y\_test)

print("多项式核的准确率是 : %f" % score\_poly)

# 训练线性核模型

param\_grid\_linear = {'C': np.logspace(-3, 3, 7),

'gamma': np.logspace(-3, 3, 7)}

grid\_linear = GridSearchCV(svm.SVC(kernel="linear"), param\_grid\_linear, cv=5)

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

scores\_linear = grid\_linear.cv\_results\_['mean\_test\_score'].reshape(len(param\_grid\_linear['C']),

len(param\_grid\_linear['gamma']))

print('线性核的最佳参数为：', grid\_linear.best\_params\_['C'], grid\_linear.best\_params\_['gamma'])

print('线性核的得分矩阵：', scores\_linear)

# 使用最佳准确率训练模型,

best\_linear = svm.SVC(kernel='linear', C=grid\_linear.best\_params\_['C'], gamma=grid\_linear.best\_params\_['gamma'])

best\_linear.fit(X\_train, y\_train)

print("最佳线性核模型准确率为: %f" % best\_linear.score(X\_test, y\_test))

print('')

heatmap('linear', scores\_linear, param\_grid\_linear)

con\_matrix('linear', best\_linear)

# 训练多项式核模型

param\_grid\_poly = {'C': np.logspace(-3, 3, 7),

'gamma': np.logspace(-3, 3, 7)}

grid\_poly = GridSearchCV(svm.SVC(kernel="poly"), param\_grid\_poly, cv=5)

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

scores\_poly = grid\_poly.cv\_results\_['mean\_test\_score'].reshape(len(param\_grid\_poly['C']), len(param\_grid\_poly['gamma']))

print('多项式核的最佳参数为：', grid\_poly.best\_params\_['C'], grid\_poly.best\_params\_['gamma'])

print('多项式核的得分矩阵：', scores\_poly)

# 使用最佳准确率训练模型

best\_poly = svm.SVC(kernel='poly', C=grid\_poly.best\_params\_['C'], gamma=grid\_poly.best\_params\_['gamma'])

best\_poly.fit(X\_train, y\_train)

print("最佳poly核模型准确率为: %f" % best\_poly.score(X\_test, y\_test))

print('')

# 使用 matplotlib 绘制准确率热力图

heatmap('poly', scores\_poly, param\_grid\_poly)

con\_matrix('poly', best\_poly)

param\_grid\_RBF = {'C': np.logspace(-1, 5, 7),

'gamma': np.logspace(-5, 1, 7)}

# 使用 GridSearchCV 训练模型,其中param\_grid中存储了各种参数的组合。其中cv5是交叉验证的折数

grid\_RBF = GridSearchCV(svm.SVC(kernel='rbf'), param\_grid\_RBF, cv=5)

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

scores = grid\_RBF.cv\_results\_['mean\_test\_score'].reshape(len(param\_grid\_RBF['C']), len(param\_grid\_RBF['gamma']))

# 这里得到了最佳的准确率

print('RBF核的最佳参数为：', grid\_RBF.best\_params\_['C'], grid\_RBF.best\_params\_['gamma'])

print('RBF核的得分矩阵：', scores)

# 使用最佳准确率训练模型,

best\_RBF = svm.SVC(kernel='rbf', C=grid\_RBF.best\_params\_['C'], gamma=grid\_RBF.best\_params\_['gamma'])

best\_RBF.fit(X\_train, y\_train)

score\_RBF = best\_RBF.score(X\_test, y\_test)

print("最佳参数RBF核模型准确率为: %f" % score\_RBF)

heatmap('RBF', scores, param\_grid\_RBF)

con\_matrix('RBF', best\_RBF)

以下是集成学习搜寻最佳参数相关的代码，去除了数据的预处理部分：

# 参数'n\_estimators'代表着弱分类器的数量

param\_grid\_rf = {

'n\_estimators': [50, 100],

'max\_depth': [None, 10],

'min\_samples\_split': [2, 5],

'min\_samples\_leaf': [1, 2]

}

rf = RandomForestClassifier(random\_state=42)

grid\_rf = GridSearchCV(rf, param\_grid\_rf, cv=5)

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

best\_rf = RandomForestClassifier(n\_estimators=grid\_rf.best\_params\_['n\_estimators'],

max\_depth=grid\_rf.best\_params\_['max\_depth'],

min\_samples\_split=grid\_rf.best\_params\_['min\_samples\_split'],

min\_samples\_leaf=grid\_rf.best\_params\_['min\_samples\_leaf'],

random\_state=42)

best\_rf.fit(X\_train, y\_train)

print("最佳随机森林模型准确率为: %f" % best\_rf.score(X\_test, y\_test))

print("最佳随机森林模型参数为：", grid\_rf.best\_params\_)

# # AdaBoost

param\_grid\_ada = {

'n\_estimators': [50, 100, 200],

'learning\_rate': [0.01, 0.1, 1]

}

ada = AdaBoostClassifier(random\_state=42)

grid\_ada = GridSearchCV(ada, param\_grid\_ada, cv=5)

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

best\_ada = AdaBoostClassifier(n\_estimators=grid\_ada.best\_params\_['n\_estimators'],

learning\_rate=grid\_ada.best\_params\_['learning\_rate'],

random\_state=42)

best\_ada.fit(X\_train, y\_train)

print("\n最佳AdaBoost模型准确率为: %f" % best\_ada.score(X\_test, y\_test))

print("最佳AdaBoost模型参数为：", grid\_ada.best\_params\_)

#

# 梯度提升

param\_grid\_gb = {

'n\_estimators': [50, 100],

'learning\_rate': [0.1, 0.5, 1.0],

'max\_depth': [3, 5, 10]

}

gb = GradientBoostingClassifier(random\_state=42)

grid\_gb = GridSearchCV(gb, param\_grid\_gb, cv=5)

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

best\_gb = GradientBoostingClassifier(n\_estimators=grid\_gb.best\_params\_['n\_estimators'],

learning\_rate=grid\_gb.best\_params\_['learning\_rate'],

max\_depth=grid\_gb.best\_params\_['max\_depth'],

random\_state=42)

best\_gb.fit(X\_train, y\_train)

print("\n最佳梯度提升模型准确率为: %f" % best\_gb.score(X\_test, y\_test))

print("最佳梯度提升模型参数为：", grid\_gb.best\_params\_)

以下是集成学习中输出最优参数下混淆矩阵相关代码：

def con\_matrix(name, model):

# 使用模型对测试数据进行预测

y\_pred = model.predict(X\_test)

# 计算混淆矩阵

cm = confusion\_matrix(y\_test, y\_pred)

# 使用 seaborn 绘制混淆矩阵

plt.figure(figsize=(7, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('predict label')

plt.ylabel('true label')

plt.title('Confusion Matrix of' + name)

plt.show()

# ///////////////////////////////////////////////////////////////////////

# 随机森林

best\_rf = RandomForestClassifier(n\_estimators=100,

max\_depth=None,

min\_samples\_split=2,

min\_samples\_leaf=1,

random\_state=42)

best\_rf.fit(X\_train, y\_train)

con\_matrix("RandomForest", best\_rf)

# ////////////////////////////////////////////////////////////////////////

best\_ada = AdaBoostClassifier(n\_estimators=100,

learning\_rate=1,

random\_state=42)

best\_ada.fit(X\_train, y\_train)

con\_matrix("AdaBoost", best\_ada)

# /////////////////////////////////////////////////////////////////////////

# 梯度提升

best\_gb = GradientBoostingClassifier(n\_estimators=50,

learning\_rate=0.5,

max\_depth=3,

random\_state=42)

best\_gb.fit(X\_train, y\_train)

con\_matrix("GradientBoosting", best\_gb)

以下是集成学习中，使用Adaboost模型构筑不同弱分类器数目下的准确率图像代码：

from sklearn.ensemble import GradientBoostingClassifier

import numpy as np

import matplotlib.pyplot as plt

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

# 弱分类器的数量

n\_estimators\_list = [5, 10, 20, 30, 50]

accuracy\_scores = []

# 遍历不同数量的弱分类器

for n\_estimators in n\_estimators\_list:

# 创建一个 GradientBoostingClassifier 模型

best\_gb = GradientBoostingClassifier(n\_estimators=n\_estimators,

learning\_rate=0.5,

max\_depth=3,

random\_state=42)

# 使用交叉验证计算分类准确率

scores = cross\_val\_score(best\_gb, X\_train, y\_train, cv=5, scoring='accuracy')

mean\_accuracy = np.mean(scores)

accuracy\_scores.append(mean\_accuracy)

# 绘制弱分类器数量与分类准确率之间的关系图

plt.figure(figsize=(10, 6))

plt.plot(n\_estimators\_list, accuracy\_scores, marker='o', linestyle='-')

plt.xlabel('Number of Weak Classifiers')

plt.ylabel('Classification Accuracy')

plt.title('Impact of Weak Classifiers Number on Classification Accuracy')

plt.grid()

plt.show()