西安电子科技大学

模式识别 课程实验报告

人工智能学院 1920032 班

姓名 付凯文 学号 19170100004

实验日期 2021 年 12 月 19 日

成绩:
实验报告内容基本要求及参考格式 一、实验目的 二、实验基本原理及步骤 三、实验仿真结果与分析 四、实验中遇到的问题及解决方法（至少 3 个，每人至少写 1 个，写清楚谁的问题和解决方法）

-

实验三　Bagging 和 Boosting 算法的性能比较

一、　实验目的

1. 掌握 Bagging 算法的原理
2. 掌握 Boosting 算法的原理
3. 编程实现 Bagging 算法与 Boosting 算法并比较两者的性能差异
4. 熟悉 Python 编程和相应的库

二、　实验基本原理及步骤

1. Bagging 算法：

Bagging 是指由多个并行的若分类器综合得到一个强分类器的方法，其全程为 Bootstrap Aggregating。它通过多次采样同一数据集的方法得到多组数据，然后分别进行训练得到若干弱分类分类器，最后通过对这些弱分类器投票的策略得到强分类器。

一般使用随机采样的方式对样本进行重复采样，即从训练集中采集固定数量的样本，每采集一个样本以后将其放回，该采样方法称为自助采样法。这样完成一轮的数据采样后，可计算出某个数据未被容器 m 的样本集选中的概率$(1-\frac{1}{m})^m$,而当 m 取值很大时，即

$$\lim_{m\to\infty}\left(1-\frac{1}{m}\right)^m=\frac{1}{e}=0.368$$

因此每一轮的采样过程中采样的数据大约覆盖了 63.2%的原始数据集，大约有 36.8%的数据集没有被抽中，而这些没有被抽中的数据则能够对算法的泛化能力进行验证。

我们假定使用的基分类器的复杂度为 $O(m)$,采样与投票的复杂度为 $O(s)$,用Ω表示整体算法的复杂度函数，则 Bagging 整体算法的复杂度约为$\Omega(O(m)+O(s))$,整个训练的过程由于采样与投票的计算复杂度低，训练轮次Ω又是一个固定的不太大的常数，因此 Bagging 算法的复杂度基本与单个基分类器的算法复杂度同阶。

Bagging 算法的主要流程如图表 1 所示：

输入：训练集 $D=\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$
分类算法 H；重采样的样本分布 D_b
训练轮数 T
训练过程：
(1) **for** $t=1, 2, \cdots\cdots, T$ **do**
(2)　　$h_t=H(D, D_b)$
(3) **end for**
输出：$H(\boldsymbol{x})=\arg\max_{y\in Y}\sum_{t=1}^{T}\Pi(h_t(\boldsymbol{x})=y)$

图表 1　Bagging 算法流程图

值得注意的是，由于对样本进行多次采样，有些数据会在采样分布中出现多次而有的可能不出现。当采样规模较大时，会有多个相同的数据多次出现在采样的各个数据集中，从统计学角度来看，这些采样的数据之间并不相互独立。为了确保集成个体的多样性，我们应该采用不稳定的分类器，通过对分类器添加扰动而获得差异较大的个体分类器，而如果使用稳定的弱分类器，那么集成的分类器将会出现相近的分类精度，导致最终的组合分类器精度有限。通常对数据样本扰动较为敏感的“不稳定基分类器”主要有决策树、神经网络等，对于数据扰动不敏感的“稳定基分类器”主要有支撑向量机、朴素贝叶斯、k 近邻等方法。

2. Boosting 算法：

Boosting 算法是一族提升分类器性能的方法。该类算法与 Bagging 算法都是通过多次对原始数据集进行重采样产生的，并且对分类器进行优化得到强分类器。与 Bagging 算法所不同的是，Boosting 算法对重采样的机制进行改进，在迭代过程中更多地关注对部分样本进行重采样。这类算法的工作机制大致为：从初始训练集训练出一个弱分类器，根据弱分类器学习出的性能调整训练集的分布，以便再次使用分类器时能够更多地关注到一些之前分类错误的学习样本，再用调整过后的样本分布进行下一个弱分类器的学习。如此反复进行多次迭代，直至组合后的弱分类器的错误率达到某一阈值，再将这若干个弱分类器进行加权结合。

下面以 Adaboost 为例进行介绍。

Adaboost [Freund and Schapire, 1997] 推导方式较多，比较常用的一种是“线性加性模型”，其步骤大致为：首先初始化数据权值和弱分类器的权值，然后开始迭代计算，每一轮数据的权值根据上一轮的计算结果更新，如果上一轮的计算过程中某一数据分类正确，则其对应权重相应减少，反之则增加，这么做的目的主要是为了让分类出错的样本受到更多的关注，再根据数据权值计算该轮弱分类器的分类正确率。若分类正确率提高，则增大该分类器的权重，反之则减少，经过若干轮迭代计算后，最终得到一个强分类器。

(1) 参数说明

$D_t(i)$: 迭代过程中第 t 轮，第 i 个样本的参数分布。

W_{ti}: 第 t 轮中第 i 个数据的权重值。

a_t: 第 t 轮弱分类器的权重值。

e_t: 第 t 轮弱分类器的错误率。

y_i: 第 i 个数据的类标值。

m: 样本总数。

(2) 具体步骤

A. 初始化第一轮的权重值，将所有数据平均取样

$$D_1(i) = (w_1, w_2, \cdots, w_m) = \left(\frac{1}{m}, \frac{1}{m}, \cdots, \frac{1}{m}\right)$$

B. 进行迭代，$t = 1,2,3,\ldots T$ 开始先选择当前数据误差率e_t最低的一个弱分类器，并计算该弱分类器$H_t(x)\epsilon\{-1,1\}$在分布D_t上的误差：

$$e_t = P(H_t(\boldsymbol{x}_i) \neq y_i) = \sum_{i=1}^{m} w_{ti} I(H_t(\boldsymbol{x}_i) \neq y_i)$$

上式给出了第 t 轮的误差计算公式，它与当前弱分类器的数据权值和分错的样本数有关，随后根据该错误率确定该分类器在改进的分类器模型中所占的比重

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

式中的a_t的值表示该分类器所占的权重大小，下一步就是根据该权重值更新样本数据的分

布D_t，可表示为

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i H_t(x_i))}{Z_t}$$

其中Z_t为归一化常数，它的值为$Z_t = 2\sqrt{e_t(1-e_t)}$。

C. 根据 t 轮所求的数据权值和各个弱分类器的权值求出最后的强分类器

$$H(\boldsymbol{x}) = \operatorname{sign}(f(\boldsymbol{x})) = \operatorname{sign}\Big(\sum_{t=1}^{T}\alpha_t H_t(\boldsymbol{x})\Big)$$

值得注意的是，由数据分布更新计算公式可以得到当前数据的预测结果对下一轮的影响，例如：当样本预测正确时，有$H_t(x_i) = y_i$。将$H_t(x_i) * y_i = 1$带入得

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t}\exp(-\alpha_t) = \frac{D_t(i)}{Z_t}\exp\Big[-\frac{1}{2}\ln\Big(\frac{1-e_t}{e_t}\Big)\Big] = \frac{D_t(i)}{2(1-e_t)}$$

需要注意的是，错误率e_t一般要求小于 0.5，因此$D_{t+1}(i) < D_t(i)$，所以在下一轮对于正确预测的样本减少关注。此外，如果算法的迭代次数过多，那么算法容易出现过拟合现象，因此要选用较小的迭代次数。

当样本预测错误时，有$H_t(x_i) \neq y_i$且$H_t(x_i) * y_i = -1$，同样的有

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t}\exp(\alpha_t) = \frac{D_t(i)}{Z_t}\exp\Big[\frac{1}{2}\ln\Big(\frac{1-e_t}{e_t}\Big)\Big] = \frac{D_t(i)}{2e_t}$$

其变化规律与预测正确时相反。

且集体算法流程图如图表 2 所示：

输入：训练集 $D=\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$

分类器 H

训练轮数 T：第七轮样本分布 D_t

训练过程：

(1) 初始化 $D_1(\boldsymbol{x}) = \frac{1}{m}$

(2) **for** $t=1, 2, \cdots\cdots, T$ **do**

(3) $h_t = H(D, D_t)$

(4) $e_t = P_{x\sim D_t}(h_t(\boldsymbol{x}_i) \neq y_i)$

(5) if $e_t > 0.5$ then break

(6) $\alpha_t = \frac{1}{2}\ln\Big(\frac{1-e_t}{e_t}\Big)$

(7) $$D_{t+1}(\boldsymbol{x}_i) = \begin{cases} \frac{D_t(\boldsymbol{x}_i)}{Z_t}\exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \frac{D_t(\boldsymbol{x}_i)}{Z_t}\exp(\alpha_t), & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$$

(8) **end for**

输出：$H(x) = \arg\max_{y\in Y}\sum_{t=1}^{T} \Pi(h_t(\boldsymbol{x}) = y)$

图表 2 AdaBoost 算法流程图

3. 实验步骤：

A. 首先导入实验所需的库，并从 sklearn 中载入 iris 数据集。

B. 对 iris 数据集中的数据进行归一化处理，利用线性变换将特征值映射到[0,1]的范围。

C. 对 iris 数据集进行可视化，观察数据分布。

D. 按 train：test=7:3 的比例从数据集中随机划分训练集和测试集。

E. 实现 RandomForest 算法，并求解出最优超参数。设置 RandomForest 的超参数为求解出的最优值，对 RandomForest 进行训练并利用测试集求解算法分类的正确率。

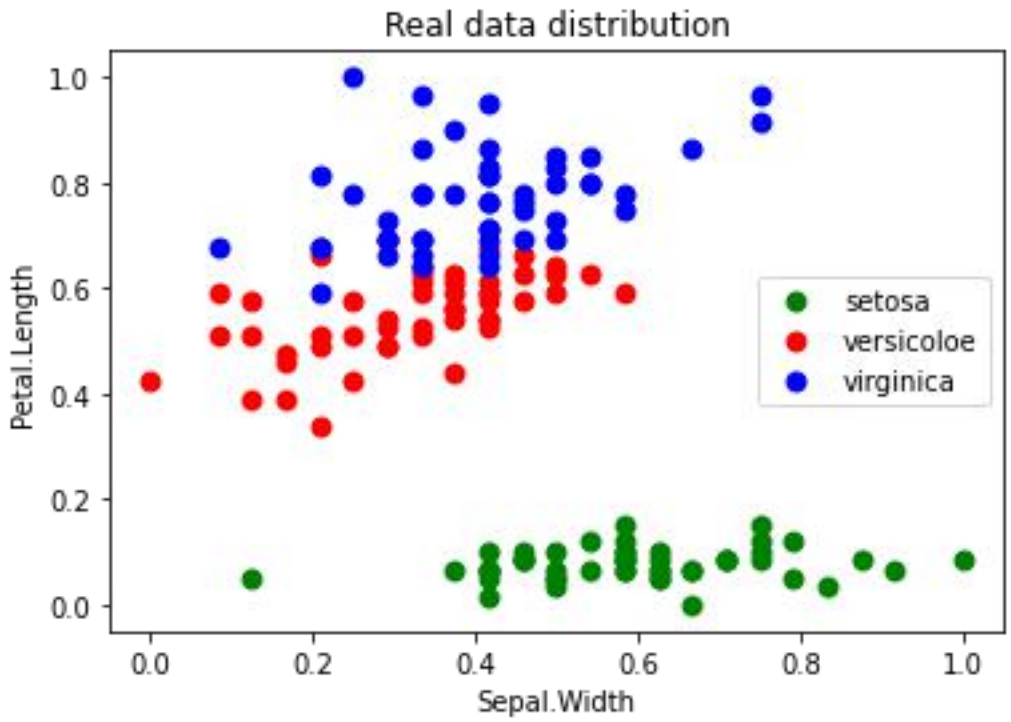
F. 实现 AdaBoost 算法，并求解出最优超参数。设置 AdaBoost 的超参数为求解出的最优值，对 AdaBoost 进行训练并利用测试集求解算法分类的正确率。

G. 对比 RandomForest 与 AdaBoost 的分类正确率以及训练时间，比较两者的性能差异。

三、 实验仿真结果及分析

实验仿真结果：

首先对 iris 数据集进行归一化和数据可视化，数据可视化的结果如图表 3 所示：



图表 3 iris 数据分布

实现 RandomForest 算法，并求解出最优超参数：

决策树最大深度	max_depth = 5
单个决策树使用特征的最大数量	max_feature = 1
最小样本叶片大小	min_samples_leaf = 7
决策树数量	n_estimators = 50

设置 RandomForest 的超参数为求解出的参数，对模型进行训练并在测试集上进行测试，得到模型分类的正确率为：0.9777

实现 AdaBoost 算法，并求解出最优超参数：

学习率	Learing_rate = 2
迭代次数	n_estimators = 60

设置 AdaBoost 的超参数为求解出的参数，对模型进行训练并在测试集上进行测试，得到模型分类的正确率为：0.9555

实验仿真结果分析：

Iris 数据集下，在测试集与训练集相同的情况下，RandomForet 的预测正确率高于 AdaBoost。RandomForest 的模型训练时间明显小于 AdaBoost，这是因为 RandomForest 可以进行并行计算而 AdaBoost 不能。因此可以得出在 iris 数据集上，相比于 AdaBoost，RandomForest 有着更优良的效果。但是并不能说明在所有情况下 RandomForest 均有更好的效果。

Bagging 与 Boosting 算法的主要区别如下：

1）样本选择

Bagging：训练集是在原始集中有放回选取的，从原始集中选出的各轮训练集之间是独立的。

Boosting：每一轮的训练集不变，只是训练集中每个样例在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整。

2）样例权重

Bagging：使用均匀取样，每个样例的权重相等

Boosting：根据错误率不断调整样例的权值，错误率越大则权重越大。

3）预测函数

Bagging：所有预测函数的权重相等。

Boosting：每个弱分类器都有相应的权重，对于分类误差小的分类器会有更大的权重。

4）并行计算

Bagging：各个预测函数可以并行生成

Boosting：各个预测函数只能顺序生成，因为后一个模型参数需要前一轮模型的结果。

Bagging 对样本重采样，对每一重采样得到的子样本训练一个模型，最后取平均。由于各模型有近似相等的 bias 和 variance，由于：

$$E[\frac{\sum X_i}{n}] = E[X_i]$$

所以 Bagging 后的 bias 和单个子模型接近，一般来说不能显著降低 bias。另一方面，如各个子模型独立，则有：

$$Var(\frac{\sum X_i}{n}) = \frac{Var(X_i)}{n}$$

此时可以显著降低 variance。若各个子模型完全相同，则：

$$Var(\frac{\sum X_i}{n}) = Var(X_i)$$

此时不会降低 variance。Bagging 方法得到的各个子模型有一定相关性，属于以上两个极端

状态的中间态，因此可以一定程度上的减低 variance。为了进一步境地 variance，RandomForest 通过随机选取变量子集做拟合的方式 de-correlated 了各子模型，使得 variance 进一步降低。

boosting 从优化角度来看，是用 forward-stagewise 这种贪心法去最小化损失函数：

$$L(y, \sum_i a_i f_i(x))$$

例如，常见的 AdaBoost 即等价于用这种方法最小化 exponential loss：

$$L(y, f(x)) = exp(-yf(x))$$

所谓 forward-stagewise，就是在迭代的第 n 步，求解新的子模型 f(x)及步长 a（或者叫组合系数），来最小化：

$$L(y, f_{n-1}(x) + af(x))$$

其中$f_{n-1}(x)$是前 n-1 步得到的子模型的和。因此 Boosting 是在 sequential 地最小化损失函数，其 bias 自然逐步下降。但由于是采取这种 sequential、adaptive 的策略，各子模型之间是强相关的，于是子模型之和并不能显著降低 variance。所以说 boosting 主要还是靠降低 bias 来提升预测精度。

两种算法各有优劣，在使用过程中应根据实际情况选取二者中更为适宜的算法。

四、 实验中遇到的问题及解决方法

遇到的问题：RandomForest 和 AdaBoost 有大量的超参数，在使用时不知道如何选取超参数的值。

解决方法：在查阅相关资料和博客后，发现 RandomForest 算法中较为重要的超参数为 max_depth,max_feature,min_samples_leaf,n_estimators。而 AdaBoost 算法中较为重要的超参数为 learning_rate 和 n_estimators。只需对这些参数进行调优即可。

另外，可以利用 sklearn 中的 GridSearchCV 对 RandomForest 和 AdaBoost 的超参数进行选择。首先设置训练集合需要调优的算法，设置超参数的取值范围，即可得到在当前训练数据上最优的超参数组合。

在本实验中，求解出的最优超参数为

RandomForest：max_depth=5、max_feature=1、min_samples_leaf=7、n_estimators=50

AdaBoost：learning_rate=2、n_estimators=60

五、 附录

```
#author:fkw
#utf-8
#creat time:2021/12/19 19：15
#bagging:random forest&&boosting:Adaboost
#dataset=iris
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from  sklearn.model_selection import GridSearchCV
#load the dataset
iris = datasets.load_iris()
# 特征值
diris = iris.data
# 标签
tiris = iris.target
labels = ['setosa', 'versicolor', 'virginica']
"""数据可视化"""
# 数组归一化 x=(x-min)/(max-min)
for i in range(4):
    diris[:, i] = (diris[:, i]-np.min(diris[:, i]))/(np.max(diris[:, i])-np.min(diris[:, i]))
#绘制 dataset 散点图,选取 2,3 列作为 Xlable,ylabel
x_axis=diris[:,1]
y_axis=diris[:,2]
plt.scatter(x_axis[tiris==0],y_axis[tiris==0],c='green',marker='.',linewidths=5)
plt.scatter(x_axis[tiris==1],y_axis[tiris==1],c='red',marker='.',linewidths=5)
plt.scatter(x_axis[tiris==2],y_axis[tiris==2],c='blue',marker='.',linewidths=5)
plt.legend(["setosa","versicoloe","virginica"], loc ='upper right')
plt.title("Real data distribution")
plt.xlabel('Sepal.Width')
plt.ylabel('Petal.Length')
plt.show()
'''''划分训练集和测试集'''
# 样本总数
num = diris.shape[0]
# 划分数据集 train/test=7:3
ratio = 7/3
# 测试集样本数目
num_test = int(num/(1+ratio))
# 训练集样本数目
num_train = num-num_test
# 产生样本标号并打乱
index = np.arange(num)
np.random.shuffle(index)
#前 30%做测试集
diris_test = diris[index[:num_test],:]
tiris_test = tiris[index[:num_test]]
```

```
#剩余的做训练集
diris_train = diris[index[num_test:],:]
tiris_train = tiris[index[num_test:]]
# RandomForest
#调参
clf = RandomForestClassifier()
parameters = {'n_estimators': range(30,100,10),'max_depth':range(3,10,2),
              'min_samples_leaf':[5,6,7],'max_features':[1,2,3]}
grid_clf = GridSearchCV(clf,parameters,scoring='f1_macro')
grid_clf.fit(diris_train, tiris_train)
grid_clf.best_params_,grid_clf.best_score_
#train
clf = RandomForestClassifier(max_depth=5,max_features=1,min_samples_leaf=7,n_es
timators=50)
# 训练模型
clf.fit(diris_train, tiris_train)
# 评价模型
score = clf.score(diris_test, tiris_test)
print("\n 模型测试集准确率为: ", score)

# AdaBoost
#调参
clf = AdaBoostClassifier()
parameters = {'n_estimators': range(10,100,10),'learning_rate':range(1,3,1)}

grid_clf = GridSearchCV(clf,parameters,scoring='f1_macro')
grid_clf.fit(diris_train, tiris_train)
grid_clf.best_params_,grid_clf.best_score_
#train
clf = AdaBoostClassifier(learning_rate=2, n_estimators=60)
# 训练模型
clf.fit(diris_train, tiris_train)
# 评价模型
score = clf.score(diris_test, tiris_test)
print("\n 模型测试集准确率为: ", score)
```