**机器学习应用实践（实验六） —集成学习**

**工业智能2201班 刘天行 20225354**

1. **实验目的**

1、熟悉 AdaBoost 算法的原理。

2、掌握 scikit-learn 中 AdaBoost 算法的使用。

1. 具备 AdaBoost 算法的编程实现能力。

4、理解 GBDT 算法的原理。

5、掌握 scikit-learn 中 GBDT 算法的使用。

1. **开发环境：Visual Studio Code + Jupyter notebook拓展 + python 3.12.2**

文本

描述已自动生成

1. **实验内容**

1、题目一： 采用 scikit-learn 中的 AdaBoostClassifier 对葡萄酒数据集进行预测。

【讨论一】模型在葡萄酒数据集上的表现如何？是过拟合还是欠拟合？【讨论二】如何提升模型性能？模型超参数对性能有何影响？

（1） 观察不同求解算法（"SAMME"和"SAMME.R"） 的性能差异。

（2） 通过学习曲线分析 AdaBoost 算法中的 n\_estimators 参数对模型性能的影响（其他参数不变）

（3）通过学习曲线分析 AdaBoost 算法中的 learning\_rate 参数对模型性能的影响。

2、 题目二： 编写 AdaBoost-SAMME 算法，并对乳腺癌数据集进行预测，并展示模型评分， 与sklearn 自带的评估器建模结果进行对比， 或与之前学习过的 SVM 算法进行对比。

3、题目三（选做） ： 采用 scikit-learn 中的 GradientBoostingRegressor 对加州房价数据集进行预测

【讨论四】 自行选择超参数寻优的方法， 确定模型最优超参数，并将建模结果与之前学过的调参后的模型(如随机森林、 Adaboost)进行比较（可从建模时间， 模型预测效果等方面进行比较） 。

1. **实验情况**

题目一：采用 scikit-learn中的AdaBoostClassifier对葡萄酒数据集进行预测

from sklearn.datasets import load\_iris

from sklearn import tree

import matplotlib.pyplot as plt

import matplotlib as mpl

import numpy as np

from sklearn.datasets import fetch\_california\_housing

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

import graphviz

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.datasets import make\_classification,load\_wine

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

from sklearn.model\_selection import validation\_curve

from sklearn.model\_selection import GridSearchCV

import time

from sklearn.metrics import mean\_squared\_error,r2\_score

from sklearn.model\_selection import cross\_validate

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import AdaBoostClassifier

from sklearn.datasets import make\_classification

plt.rcParams['font.family'] = 'SimHei'      # 中文正常显示

plt.rcParams['axes.unicode\_minus'] = False  # 符号正常显示

# 加载数据集

data = load\_wine()

frame = data.frame # 传入参数 as\_frame=True 时才会返回

X = data.data

y = data.target

# 划分数据集，训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

feature\_names = data.feature\_names

target\_names = data.target\_names

clf\_model = AdaBoostClassifier(random\_state=10)

cross\_ABC = cross\_val\_score(clf\_model, X, y, cv=10, n\_jobs=-1)

plt.plot(cross\_ABC,label='ABC')

plt.title('cross\_val\_score on 10 folds')

plt.xlabel('folds')

plt.ylabel('score')

plt.show()

# 输出模型评分

print("模型评分：", cross\_ABC.mean())

通过scikit-learn自带的Adaboost对葡萄酒数据集的预测模型评分随交叉验证次数变化的曲线如图：

图表, 折线图

描述已自动生成



讨论一：模型在葡萄酒数据集上的表现如何？是过拟合还是欠拟合？

我们在代码中添加对比模型训练得分和交叉验证得分的部分，并使用scikit\_learn内置的学习曲线绘制出两个得分随着训练样本量变化的曲线。

# 模型在训练集和测试集上的性能

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

train\_score = clf\_model.score(X\_train, y\_train)

test\_score = clf\_model.score(X\_test, y\_test)

print("训练集评分：", train\_score)

print("测试集评分：", test\_score)

# 绘制学习曲线

train\_sizes, train\_scores, test\_scores = learning\_curve(

    clf\_model, X, y, cv=10, n\_jobs=-1, train\_sizes=np.linspace(0.1, 1.0, 5))

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

plt.figure()

plt.title("Learning Curve")

plt.xlabel("Training examples")

plt.ylabel("Score")

plt.ylim(0.0, 1.1)

plt.grid()

plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

                 train\_scores\_mean + train\_scores\_std, alpha=0.1, color="r")

plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

                 test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r", label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g", label="Cross-validation score")

plt.legend(loc="best")

plt.show()

文本

描述已自动生成

图表, 折线图

描述已自动生成

观察这条学习曲线，红色曲线表示训练得分，绿色曲线表示交叉验证得分。可见训练得分始终稳定在较高的水平，而交叉训练得分在训练样本较少时较低，随着样本量增加，交叉训练得分逐渐提高，最终达到与训练得分接近的程度，最终训练得分还是略高于交叉验证得分的，因此我认为模型具有些许的过拟合。

讨论二：如何提升模型性能？模型超参数对性能有何影响？

# 要调整模型性能，可以调整的参数有：

# 1. n\_estimators：基分类器的数量

# 2. learning\_rate：学习率

# 3. base\_estimator：基分类器

# 4. algorithm：SAMME和SAMME.R ,默认为SAMME.R,指基分类器的权重计算方式

# 5. random\_state：随机种子

# AdaBoost默认采用的是 SAMME.R算法，即基分类器为决策树，学习率为1，基分类器数量为50

# 建立AdaBoost分类模型（使用SAMME算法）

model\_samme = AdaBoostClassifier(algorithm='SAMME',random\_state=10)

# 使用交叉验证评估SAMME算法模型

scores\_samme = cross\_val\_score(model\_samme, X, y, cv=10)

print("SAMME算法模型评分：", scores\_samme.mean())

# 建立AdaBoost分类模型（使用SAMME.R算法）

model\_samme\_r = AdaBoostClassifier(algorithm='SAMME.R',random\_state=10)

# 使用交叉验证评估SAMME.R算法模型

scores\_samme\_r = cross\_val\_score(model\_samme\_r, X, y, cv=10)

print("SAMME.R算法模型评分：", scores\_samme\_r.mean())

# 从评分结果来看，SAMME算法的模型性能略优于SAMME.R算法的模型性能。

plt.figure(figsize=(6, 4))

plt.plot(scores\_samme, label='SAMME')

plt.plot(scores\_samme\_r, label='SAMME.R')

plt.legend()

plt.title('cross\_val\_score on 10 folds')





图表, 折线图

描述已自动生成

从评分结果来看，SAMME算法的模型性能略优于SAMME.R算法的模型性能。

接下来，讨论决策树数量与学习率对模型性能的影响。

lr\_list = [0.1, 0.5, 1, 1.5, 2]

val\_curves = []

param\_range = np.arange(1, 400, 2)

for lr in lr\_list:

    clf = AdaBoostClassifier(learning\_rate=lr,random\_state=10,algorithm='SAMME')

    train\_scores, val\_scores = validation\_curve(

        clf, X, y, param\_name='n\_estimators', scoring='accuracy',

        param\_range=param\_range, cv=10, n\_jobs=-1       # n\_jobs=-1表示使用所有的CPU核心

    )

    val\_mean = np.mean(val\_scores, axis=1)

    val\_curves.append(val\_mean)

    plt.plot(param\_range, val\_mean, label="lr={}".format(lr))

plt.legend(loc='best')

plt.xlabel('n\_estimators')

plt.ylabel('Accuracy')

plt.title('AdaBoost Learning Curve with Different Learning Rates')

plt.show()

图表, 折线图

描述已自动生成

从图中可以看出，学习率为0.5和1时，模型性能最佳，且收敛速度较快。太低的学习率（如0.1）虽然稳定，但收敛速度较慢，太高的学习率（如1.5和2）则会导致模型性能下降和不稳定。在适中的学习率下，增加决策树数量可以显著提高模型性能，但达到一定数量后，性能趋于稳定，增加更多的树对性能提升有限。

题目二： 编写 AdaBoost-SAMME 算法，并对乳腺癌数据集进行预测，并展示模型评分， 与sklearn 自带的评估器建模结果进行对比， 或与之前学习过的 SVM 算法进行对比。

import numpy as np

from sklearn.datasets import load\_breast\_cancer

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import accuracy\_score

class AdaBoostSAMME:

    def \_\_init\_\_(self, n\_estimators=50):

        self.n\_estimators = n\_estimators

        self.estimators = []

        self.estimator\_weights = []

        self.estimator\_errors = []

    def fit(self, X, y):

        n\_samples = X.shape[0]

        sample\_weights = np.full(n\_samples, 1 / n\_samples)

        classes = np.unique(y)

        for \_ in range(self.n\_estimators):

            # 创建一个弱分类器（这里假设使用决策树作为弱分类器）

            estimator = DecisionTreeClassifier(max\_depth=1)

            estimator.fit(X, y, sample\_weight=sample\_weights)

            # 计算弱分类器的错误率

            y\_pred = estimator.predict(X)

            incorrect = y\_pred != y

            estimator\_error = np.sum(sample\_weights \* incorrect) / np.sum(sample\_weights)

            # 计算弱分类器的权重

            estimator\_weight = np.log((1 - estimator\_error) / estimator\_error) + np.log(len(classes) - 1)

            # 更新样本权重

            sample\_weights \*= np.exp(estimator\_weight \* incorrect)

            # 归一化样本权重

            sample\_weights /= np.sum(sample\_weights)

            # 保存弱分类器和其权重

            self.estimators.append(estimator)

            self.estimator\_weights.append(estimator\_weight)

            self.estimator\_errors.append(estimator\_error)

    def predict(self, X):

        n\_samples = X.shape[0]

        classes = len(self.estimators[0].classes\_)

        predictions = np.zeros((n\_samples, classes))

        for estimator, weight in zip(self.estimators, self.estimator\_weights):

            y\_pred = estimator.predict(X)

            for i in range(n\_samples):

                predictions[i, y\_pred[i]] += weight

        # 对预测结果进行投票

        y\_pred = np.argmax(predictions, axis=1)

        return y\_pred

# 自定义交叉验证

def custom\_cross\_val\_score(estimator, X, y, cv=5):

    n\_samples = X.shape[0]

    fold\_size = n\_samples // cv

    scores = []

    for i in range(cv):

        start = i \* fold\_size

        end = (i + 1) \* fold\_size

        X\_train = np.concatenate((X[:start], X[end:]), axis=0)

        y\_train = np.concatenate((y[:start], y[end:]), axis=0)

        X\_test = X[start:end]

        y\_test = y[start:end]

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

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

        accuracy = np.mean(y\_pred == y\_test)

        scores.append(accuracy)

    return scores

# 加载乳腺癌数据集

cancer = load\_breast\_cancer()

X, y = cancer.data, cancer.target

# 创建AdaBoost-SAMME分类器

clf = AdaBoostSAMME(n\_estimators=52)

# 进行自定义交叉验证

custom\_scores = custom\_cross\_val\_score(clf, X, y, cv=10)

# 打印每折交叉验证的准确率

for i, score in enumerate(custom\_scores):

    print(f"Fold {i+1}: {score}")

# 打印平均准确率

print("平均准确率:", np.mean(custom\_scores))

文本

描述已自动生成

主要代码解释：

样本权重初始化：

对应代码：

sample\_weights = np.full(n\_samples, 1 / n\_samples)

弱分类器的错误率：

对应代码：

# 计算弱分类器的错误率

            y\_pred = estimator.predict(X)

            incorrect = y\_pred != y

            estimator\_error = np.sum(sample\_weights \* incorrect) / np.sum(sample\_weights)

弱分类器的权重：

对应代码：

# 计算弱分类器的权重

            estimator\_weight = np.log((1 - estimator\_error) / estimator\_error) + np.log(len(classes) - 1)

更新样本权重：

对应代码：

# 更新样本权重

            sample\_weights \*= np.exp(estimator\_weight \* incorrect)

归一化样本权重：

对应代码：

# 归一化样本权重

            sample\_weights /= np.sum(sample\_weights)

使用sklearn实现：

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

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

sk\_model = AdaBoostClassifier(learning\_rate=1, n\_estimators=52, random\_state=10,algorithm='SAMME')

# 交叉验证

scores = cross\_val\_score(sk\_model, X, y, cv=10,n\_jobs=-1)

# 可视化对比上面的custom\_scores和sklearn的scores

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

plt.plot(range(1, 11), custom\_scores, label='custom\_scores')

plt.plot(range(1, 11), scores, label='sklearn\_scores')

plt.legend()

plt.title(f'comparison of two model,custom\_scores = {np.mean(custom\_scores)}, sklearn\_scores = {np.mean(scores)}')

plt.xlabel('folds')

plt.ylabel('mdoel score')

plt.show()

# 出现了手写的AdaBoost-SAMME分类器的准确率比sklearn的AdaBoostClassifier分类器的准确率高的情况，

**运行得到：**

**图表, 折线图

描述已自动生成**

**实验三：采用 scikit-learn 中的 GradientBoostingRegressor 对加州房价数据集进行预测**

# 导入数据集

from sklearn.datasets import fetch\_california\_housing

import time

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.model\_selection import validation\_curve

import numpy as np

from sklearn.model\_selection import validation\_curve

from sklearn.model\_selection import RandomizedSearchCV

import matplotlib.pyplot as plt

# 导入数据集

data = fetch\_california\_housing()

X\_california = data.data

y\_california = data.target

feature\_names = data.feature\_names

# 输出数据集信息

print("数据集大小:", X\_california.shape)

print("特征数量:", len(feature\_names))

print("特征名称:", feature\_names)

print("标签分布情况:\n", data.target)

plt.rcParams['font.family'] = 'SimHei'      # 中文正常显示

plt.rcParams['axes.unicode\_minus'] = False  # 符号正常显示

# 模型建立：使用 GradientBoostingRegressor 建立分类模型

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.model\_selection import KFold

model = GradientBoostingRegressor()

# 输出：模型评分（使用交叉验证得到模型的综合评分）

# cv = KFold(n\_splits=10, shuffle=True, random\_state=1111)

# cv=KFold(n\_splits=10, shuffle=True, random\_state=1111)：这里使用了 KFold 类，将数据集划分为 10 个折叠（folds），即将数据分成 10 份。

# shuffle=True 表示在划分之前对数据进行随机洗牌，random\_state=1111 是随机数种子，确保每次划分的结果相同

# 两者的功能相似，都将数据集分为 10 个部分，但 KFold 提供了更多的灵活性。通过设置 shuffle=True 和 random\_state 参数，可以控制数据集的洗牌和重复性

# cv=10 则只是简单将数据集等分为 10 份，没有进一步的配置选项

scores = cross\_val\_score(model, X\_california, y\_california, cv=KFold(n\_splits=10, shuffle=True, random\_state=1111),n\_jobs=-1)

print("模型评分:", scores)

print("模型评分均值:", scores.mean())

文本

描述已自动生成

文本

描述已自动生成

**讨论三：自行选择超参数寻优的方法， 确定模型最优超参数，并将建模结果与之前学过的调参后的模型(如随机森林、 Adaboost)进行比较（可从建模时间， 模型预测效果等方面进行比较） 。**

# 考虑调整GradientBoostingRegressor模型中的几个超参数：

# n\_estimators：基学习器的数量

# learning\_rate：学习率

# max\_depth：基学习器的最大深度

# min\_samples\_split：内部节点再划分所需最小样本数

# min\_samples\_leaf：叶子节点最少样本数

# 定义超参数范围

# param\_range = [0.01, 0.1, 0.5, 1.0]

def roughly\_show(param\_name,param\_range):

    # 使用validation\_curve函数计算学习曲线，该函数会在不同超参数值上拟合模型，并在训练集和验证集上计算性能指标（这里使用均方根误差）。

    # param\_name是要调优的超参数名称，这里是learning\_rate。param\_range是超参数的范围

    # scoring指定了使用的性能指标，这里是负均方根误差

    train\_scores, valid\_scores = validation\_curve(

        GradientBoostingRegressor(),  # 使用GradientBoostingRegressor模型

        X\_california, y\_california,  # 数据集

        param\_name=param\_name,  # 要调优的超参数名称

        param\_range=param\_range,  # 超参数范围

        cv=KFold(n\_splits=10, shuffle=True, random\_state=1111),  # 交叉验证策略

        scoring="neg\_mean\_squared\_error",  # 使用均方根误差作为性能指标

        n\_jobs=-1  # 并行运行

    )

    # 计算性能指标的均值和标准差

    train\_scores\_mean = -train\_scores.mean(axis=1)

    train\_scores\_std = train\_scores.std(axis=1)

    valid\_scores\_mean = -valid\_scores.mean(axis=1)

    valid\_scores\_std = valid\_scores.std(axis=1)

    # 绘制学习曲线

    plt.figure(figsize=(6, 4))

    plt.title(param\_name+"对GradientBoostingRegressor模型性能的影响")

    plt.xlabel(param\_name)

    plt.ylabel("均方根误差")   # 均方根误差MSE

    plt.grid()

    # rfill\_between函数绘制训练集和验证集性能指标的均值和标准差的范围，以可视化性能的稳定性

    plt.fill\_between(param\_range, train\_scores\_mean - train\_scores\_std,

                    train\_scores\_mean + train\_scores\_std, alpha=0.1, color="r")

    plt.fill\_between(param\_range, valid\_scores\_mean - valid\_scores\_std,

                    valid\_scores\_mean + valid\_scores\_std, alpha=0.1, color="g")

    # 使用plot函数绘制训练集和验证集的性能指标随超参数值变化的曲线

    plt.plot(param\_range, train\_scores\_mean, 'o-', color="r", label="训练集性能指标")   # 训练集的性能指标

    plt.plot(param\_range, valid\_scores\_mean, 'o-', color="g", label="验证集性能指标") # 验证集的性能指标

    plt.legend(loc="best")

    plt.show()

roughly\_show("learning\_rate",[0.01, 0.1, 0.2, 0.5, 1.0, 1.5, 2.0])

roughly\_show("n\_estimators",[1, 10, 20, 50, 100, 200, 500, 1500])

roughly\_show("max\_depth",[ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19])

roughly\_show("min\_samples\_split",[2, 4, 6, 8, 10, 20, 60, 100])

# roughly\_show("n\_estimators",[None, 1, 2, 5, 10, 15, 20])

**图表, 折线图

描述已自动生成**

**图表, 折线图

描述已自动生成**

**图表, 折线图

描述已自动生成**

**图表, 散点图

描述已自动生成**

# 由上图可视化图片，可以看出：

# 1、学习率对模型性能的影响非常大

# 接下来可利用以上信息做更细致的RandomizedSearchCV调参

# 定义超参数范围

param\_dist = {

    "n\_estimators": [500, 1000, 1500],

    "learning\_rate": [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0],

    "max\_depth": [7, 9, 11, 13, 15]

}

# 使用RandomizedSearchCV函数进行随机搜索，搜索超参数的组合

# n\_iter指定了搜索的次数，cv指定了交叉验证策略，n\_jobs指定了并行运行的任务数

# scoring指定了使用的性能指标，这里是负均方根误差

# random\_state指定了随机种子，这里是1111

# verbose指定了搜索过程的详细程度，这里是2，表示打印出每次搜索的结果

search = RandomizedSearchCV(

    GradientBoostingRegressor(),  # 使用GradientBoostingRegressor模型

    param\_distributions=param\_dist,  # 超参数范围

    n\_iter=10,  # 搜索次数，也就是不同的超参数组合的个数

    cv=KFold(n\_splits=10, shuffle=True, random\_state=1111),  # 交叉验证策略

    scoring="neg\_mean\_squared\_error",  # 使用均方根误差作为性能指标

    n\_jobs=-1,  # 并行运行

    random\_state=1111 # 随机种子

)

# 使用随机搜索训练模型

search.fit(X\_california, y\_california)

# 打印出最佳模型的超参数组合

# print("最佳参数组合 ",search.best\_params\_)

print("最佳参数组合 ",search.best\_estimator\_)

# 打印出最佳模型的性能指标

print(-search.best\_score\_)



**应用上面的最佳参数组合，与随机森林和Adaboost对比**import numpy as np

import pandas as pd

from sklearn.datasets import fetch\_california\_housing

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

from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor, AdaBoostRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

import time

import matplotlib.pyplot as plt

# 加载加利福尼亚房价数据集

california = fetch\_california\_housing()

X, y = california.data, california.target

# 划分数据集为训练集和测试集

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

# 定义模型及其超参数

models = {

    'Gradient Boosting': GradientBoostingRegressor(learning\_rate=0.3, max\_depth=11, n\_estimators=500),

    'Random Forest': RandomForestRegressor(n\_estimators=100, max\_depth=10, random\_state=42),

    'AdaBoost': AdaBoostRegressor(estimator=DecisionTreeRegressor(max\_depth=1), n\_estimators=50, random\_state=42)

}

# 比较模型的建模时间和预测效果

results = []

for name, model in models.items():

    # 记录建模时间

    start\_time = time.time()

    model.fit(X\_train, y\_train)

    end\_time = time.time()

    # 预测

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

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

    # 计算均方误差和R2得分

    mse\_train = mean\_squared\_error(y\_train, y\_pred\_train)

    mse\_test = mean\_squared\_error(y\_test, y\_pred\_test)

    r2\_train = r2\_score(y\_train, y\_pred\_train)

    r2\_test = r2\_score(y\_test, y\_pred\_test)

    # 记录结果

    results.append({

        'Model': name,

        'Training Time (s)': end\_time - start\_time,

        'Train MSE': mse\_train,

        'Test MSE': mse\_test,

        'Train R2': r2\_train,

        'Test R2': r2\_test

    })

# 打印比较结果

results\_df = pd.DataFrame(results)

print(results\_df)

# 可视化比较结果

# MSE比较

results\_df.set\_index('Model')[['Train MSE', 'Test MSE']].plot(kind='bar', figsize=(10, 6))

plt.title('Model Comparison - MSE')

plt.ylabel('MSE')

plt.show()

# R2得分比较

results\_df.set\_index('Model')[['Train R2', 'Test R2']].plot(kind='bar', figsize=(10, 6))

plt.title('Model Comparison - R2 Score')

plt.ylabel('R2 Score')

plt.show()

# 训练时间比较

results\_df.set\_index('Model')[['Training Time (s)']].plot(kind='bar', figsize=(10, 6))

plt.title('Model Comparison - Training Time')

plt.ylabel('Training Time (s)')

plt.show()

文本

描述已自动生成

**图表, 条形图

描述已自动生成**

**图表, 条形图

描述已自动生成**

**图表

描述已自动生成**

**可见，从预测效果上来看Grandient Boosting是强于随机森林和Adaboost的，但是它的训练时长远大于后两者。**