数据挖掘

集成学习器分类设计

学　　院：理 学 院
班　　级：信科 19-1
作　　者：王 浩 男
指导教师：张　　宇

2022 年 11 月

辽宁工程技术大学

1. 数据简介及分析

1.1 数据集背景

1. 数据简介：从 2013 年 5 月到 2022 年 10 月，超过 50 种加密货币的 OHLC 历史数据，这是一个包含 50 多种加密货币历史 OHLC（开高低收）数据的数据集。日期范围为每天 2013 年 5 月至 2022 年 10 月。价格以美元或$. 文件格式是 CSV 表格，加载速度更快。原作者的数据整理是使用自动脚本从 Coin Market Cap 网站上抓取和清理的。

2.数据来源：Kaggle 数据集网站

3.数据网址：https://www.kaggle.com/datasets/maharshipandya/-cryptocurrency-historical-prices-dataset

4.特征类型：多变量

5.实例数目：20052

6.属性数目：8

1.2 数据集属性介绍

1.源数据集

命名文件：加密货币价格数据.csv

- open：特定日期的开盘价（UTC 时间）
- high：该特定日期（UTC 时间）的最高价格
- low：特定日期（UTC 时间）的最低价格
- close：特定日期的收盘价（UTC 时间）
- volume：买入或卖出的资产数量，以基础货币显示
- market Cap：已开采的所有硬币的总价值。它是通过将流通中的硬币数量乘以单个硬币的当前市场价格来计算的
- Price fluctuation：价格货币的波动趋势
- timestamp：所考虑日期的 UTC 时间戳
- crypto name：加密货币的名称
-

	A	B	C	D	E	F	G	H	I	J	K	L	M
		open	high	low	close	volume	marketCap	Price fluctuation	Class	timestamp	crypto_name	date	
0		112.9	118.8	107.143	115.9	55.68378	1.29E+09	-0.699648398	0	2013-05-05T23:59:59.99	Bitcoin	2013/5/5	
1		3.49313	3.69246	3.34606	3.591	58.88243	62298185	-0.515087909	0	2013-05-05T23:59:59.99	Litecoin	2013/5/5	
2		115.98	124.663	106.64	112.3	39.34165	1.25E+09	1.051164429	0	2013-05-06T23:59:59.99	Bitcoin	2013/5/6	
3		3.59422	3.78102	3.11602	3.371	57.17845	58594361	-0.636238369	0	2013-05-06T23:59:59.99	Litecoin	2013/5/6	
4		112.25	113.444	97.7	111.5	40.67223	1.24E+09	1.123491692	0	2013-05-07T23:59:59.99	Bitcoin	2013/5/7	
5		3.37087	3.40672	2.93979	3.333	46.69811	58051265	0.416721117	0	2013-05-07T23:59:59.99	Litecoin	2013/5/7	
6		3.28362	3.49112	3.28362	3.409	48.76506	59508216	-0.112167573	0	2013-05-08T23:59:59.99	Litecoin	2013/5/8	
7		109.6	115.78	109.6	113.6	39.84406	1.26E+09	0.389540448	0	2013-05-08T23:59:59.99	Bitcoin	2013/5/8	
8		3.3994	3.44169	3.29485	3.416	52.62708	59755569	0.170347382	0	2013-05-09T23:59:59.99	Litecoin	2013/5/9	
9		113.2	113.46	109.26	112.7	39.49649	1.25E+09	1.162877124	0	2013-05-09T23:59:59.99	Bitcoin	2013/5/9	
10		112.799	122	111.551	117.2	45.28807	1.31E+09	0.283952506	0	2013-05-10T23:59:59.99	Bitcoin	#######	
11		3.40653	3.63164	3.36719	3.443	44.05824	60358927	0.115361506	0	2013-05-10T23:59:59.99	Litecoin	#######	
12		117.7	118.679	113.01	115.2	49.55433	1.28E+09	-0.080469602	0	2013-05-11T23:59:59.99	Bitcoin	#######	
13		3.46038	3.47981	3.30061	3.349	45.50658	58809513	0.661459458	0	2013-05-11T23:59:59.99	Litecoin	#######	
14		3.36281	3.42224	3.21587	3.269	51.52448	57513320	0.046797173	0	2013-05-12T23:59:59.99	Litecoin	#######	
15		115.64	117.449	113.435	115	51.94572	1.28E+09	-0.287984087	0	2013-05-12T23:59:59.99	Bitcoin	#######	
16		3.26089	3.37343	3.19895	3.28	49.01765	57790879	-0.256699775	0	2013-05-13T23:59:59.99	Litecoin	#######	

图 1 源数据集——加密货币价格数据.csv 部分数据展示

2. 处理后的数据集

①由于源数据集中的 timestamp：所考虑日期的 UTC 时间戳和 crypto name：加密货币的名称两个类别对本次集成学习分类器的探究无意义，故进行数据删除。

②源数据中存在 78546 条数据，数据集过于庞大，对于目前的实验环境运行时间太慢，故选取其中 20052 条数据集进行实验运行处理。

命名文件：new 加密货币价格数据.csv

- open：特定日期的开盘价（UTC 时间）
- high：该特定日期（UTC 时间）的最高价格
- low：特定日期（UTC 时间）的最低价格
- close：特定日期的收盘价（UTC 时间）
- volume：买入或卖出的资产数量，以基础货币显示
- market Cap：已开采的所有硬币的总价值。它是通过将流通中的硬币数量乘以单个硬币的当前市场价格来计算的
- Price fluctuation：价格货币的波动趋势
- Class:数据标签

具体的数据集如下:

	A	B	C	D	E	F	G	H
1	112.9	118.8	107.143	115.91	55.68378	1.29E+09	-0.69965	0
2	3.49313	3.69246	3.34606	3.59089	58.88243	62298185	-0.51509	0
3	115.98	124.663	106.64	112.3	39.34165	1.25E+09	1.051164	0
4	3.59422	3.78102	3.11602	3.37125	57.17845	58594361	-0.63624	0
5	112.25	113.444	97.7	111.5	40.67223	1.24E+09	1.123492	0
6	3.37087	3.40672	2.93979	3.33274	46.69811	58051265	0.416721	0
7	3.28362	3.49112	3.28362	3.40924	48.76506	59508216	-0.11217	0
8	109.6	115.78	109.6	113.566	39.84406	1.26E+09	0.38954	0
9	3.3994	3.44169	3.29485	3.41615	52.62708	59755569	0.170347	0
10	113.2	113.46	109.26	112.67	39.49649	1.25E+09	1.162877	0
11	112.799	122	111.551	117.2	45.28807	1.31E+09	0.283953	0
12	3.40653	3.63164	3.36719	3.44334	44.05824	60358927	0.115362	0
13	117.7	118.679	113.01	115.243	49.55433	1.28E+09	-0.08047	0
14	3.46038	3.47981	3.30061	3.34896	45.50658	58809513	0.661459	0
15	3.36281	3.42224	3.21587	3.26945	51.52448	57513320	0.046797	0
16	115.64	117.449	113.435	115	51.94572	1.28E+09	-0.28798	0
17	3.26089	3.37343	3.19895	3.27984	49.01765	57790879	-0.2567	0
18	114.82	118.699	114.5	117.98	51.74352	1.32E+09	-0.01124	0
19	117.98	119.8	110.25	111.5	51.691	1.24E+09	-0.27182	0
20	3.28101	3.33185	2.7772	2.8243	41.5722	49847568	4.154106	1
21	111.4	115.81	103.5	114.22	51.89039	1.27E+09	-0.0265	0
22	2.82399	3.03582	2.63925	2.93545	41.13997	51891652	0.32042	0

图 2 处理后数据集——new 加密货币价格数据.csv 部分数据展示

2 集成思想及设计思路

2.1 个体学习器的选择

集成学习算法是通过组合使用多种学习算法来获得比单独使用任何学习算法更好的功能。本文采用 Bagging 集成学习方法，依据 Kaggle 中 Databases 的 Cryptocurrency Prices Data（加密货币价格数据）训练分类器，任务是波动趋势较大和波动趋势较小的货币进行分类，以帮助客户发现金融市场趋势，针对自身需求购买对应的货币。

Bagging 是一种个体学习器之间不存在强依赖关系且可同时进行训练的并行式集成学习方法，它可以通过结合多个个体学习器来降低泛化误差。具体的过程如下：

对于包含 n 个训练样本的数据集 D，组合的分类器有 k 个，

（1）利用自助法生成 k 个训练集 $D_i(i=1,2,\cdots,\mathrm{k})$，即每个 D_i 都是从原数据集中有放回地抽取 n 个样本得到的

（2）在每个训练集 D_i 上学习一个分类器 M_i

（3）最终的分类结果由所有的分类器投票得到，即分类结果数最多的作为最终的分类结果。一般 Bagging 方法得到的分类器会比单个分类器更准确，原因是其对于噪声数据和过拟合问题表现得学习能力更强。

为了获得较好的集成效果，本文首先选用决策树、K 最近邻算法（KNN）、朴素贝叶斯（朴素贝叶斯）、BP 神经网络作为 Bagging 的 100 个子学习方法；然后，采用 **K 折（K=10）交叉验证方法**划分训练集和测试集，计算出每折集成学习的测试错误率，并对集成学习的多样性进行评价；接着，本文选择预测效果最好的一折得到的模型作为最终的集成分类器；最后，本文通过对集成学习器与单个 K 最近邻算法进行比较，给出对集成学习器的评价。

图 3 代码运行文件介绍

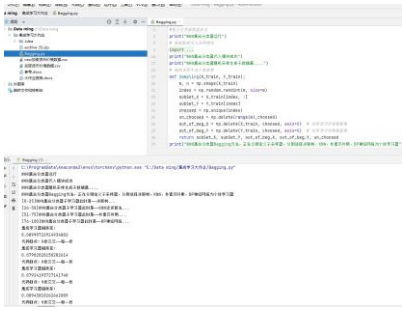

图 4 代码运行过程展示

2.2 个体学习器的训练方法

本文分别选择 25 个相同的决策树、K 最近邻算法（KNN）、朴素贝叶斯（朴素贝叶斯）、BP 神经网络作为 Bagging 的个体学习器。参数见下表：

表 1 Bagging 个体学习器参数

个体学习器	参数
决策树	特征选择标准：基尼指数。叶子节点最少样本数：1。叶子节点最小的样本权重：0
KNN	邻居个数：5
朴素贝叶斯	假设分布：高斯分布；估计方法：极大似然估计
BP 神经网络	激活函数：$f(x)=\max(0,x)$；层数：3；隐层节点数：100

2.3 简要说明结合策略

对于分类任务来说，集成学习器的关键是从类别标记集合中预测出一组标记，最常见的策略就是投票法，本文利用加权投票法进行结合策略（weighted voting）

$$H(\mathrm{x})=\mathrm{c}_{\arg\max\sum_{i=1}^{T}w_j\ \ h(x)_i^j}$$

与加权平均法类似，$w_i\geq 0,\ \sum_{i=1}^{T}w_i=1$，$w_i$ 是 h_i 的权重。

2.4 WHN 集合分类器的算法伪代码

这个部分展示当初构想的算法伪代码，具体的完整代码见（代码附录 Bagging 集成学习器代码）

输入： 数据集 $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\}$

定义子学习器分别是 h_i //[0-25]个是决策树，[26-50]是 KNN，[51-75]是朴素贝叶斯，[76-100]是 BP-神经网络

过程：

$D_{train},D_{test}=kFold(Data,k=10)$ //将数据集进行 K=10 折划分

for $k=1,2,\ldots,10$ *do*

for t=1,2,…,100 *do*

$D_t^k,D^{oob(k)}=Sampling(D_{train}{}^k)$ //随机采样得到子训练集和验证集

if $t<25$ *do*

$h_t=Tree(D_t^k,D^{oob(k)})$ //训练决策树学习器

elseif $t<50$ *do*

$h_t=Knn(D_t^k,D^{oob(k)})$ //训练 KNN 学习器

elseif $t<75$ *do*

$h_t=Nav\_Bayes(D_t^k,D^{oob(k)})$ //训练朴素贝叶斯学习器

else *do*

$h_t=Bp(D_t^k,D^{oob(k)})$ //训练 BP-神经网络学习器

end *if*

end for

计算 ε^{oob}

计算多样性 *Variety*

end for

$H=\arg\min(\varepsilon^{oob})$ //选择泛化误差的最小的一折作为最终集成学习器

输出： $H=\{h_1,h_2,\ldots,h_{100}\}$

3. 实验结果及分析

3.1 集成学习器泛化学习能力评价

本文采用 K 折交叉验证方法划分训练集和测试集，采用有放回的抽样方式对训练集进行随机采样生成随机子集和验证集。

由抽样方法可知，每次抽样大约有 37.2%的数据未被抽到，这 37.2%的样本可用作验证集来对泛化性能进行“二次估计”。

设 D_t 为每个个体学习器 h_t 实际使用的训练样本，$H^{oob}(x)$ 表示对样本 x 的包外预测，则有：

$$H^{oob}(x)=\arg\max_{y\in Y}\sum_{t=1}^{T}\mathrm{II}(h_t(x)=y)\cdot\mathrm{II}(x\notin D_t)$$

其中 Bagging 泛化误差的估计为：

$$\varepsilon^{oob}=\frac{1}{|D|}\sum_{(x,y)\in D}\mathrm{II}(H^{oob}(\mathrm{x})\neq \mathrm{y})$$

3.2 多样性评价

①多样性度量（diversity measure）是用于度量集成个体分类器的多样性，即估算个体学习器的多样化程度，典型做法是考虑个体分类器的两两相似/不相似性。给定数据集 $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\}$，对二分类任务，$y_i\in\{-1,+1\}$，分类器 h_i 与 h_j 的预测结果列联表为：

表 2 预测结果列联表

	$h_i=+1$	$h_i=-1$
$h_j=+1$	a	c
$h_j=-1$	b	d

其中，a 表示 h_i 与 h_j 均预测为正类的样本数目；b、c、d 含义由此类推；$a+b+c+d=m$，下面给出一些常见的多样性度量：

- 不合度量（disagreement measure）：

$$dis_{ij}=\frac{b+c}{m}.$$

- 相关系数（correlation coefficient）：

$$\rho_{ij}=\frac{ad-bc}{\sqrt{(a+b)(a+c)(c+d)(b+d)}}.$$

- Q-统计量（Q-statistic）：

$$Q_{ij}=\frac{ad-bc}{ad+bc}.$$

Q_{ij} 与相关系数 ρ_{ij} 的符号相同，且 $\left|Q_{ij}\right|\le\left|\rho_{ij}\right|$.

- κ-统计量（κ-statistic）：

$$\kappa=\frac{p_1-p_2}{1-p_2}.$$

其中，p_1 是两个分类器取得一致的概率；p_2 是两个分类器偶然达成一致的概率，它们可有数据集 D 估算：

$$p_1=\frac{a+d}{m},$$

$$p_2=\frac{(a+b)(a+c)+(c+d)(b+d)}{m^2}.$$

若分类器 h_i 与 h_j 在 D 上完全一致，则 $\kappa=1$；若它们仅是偶然达成一致，则 $\kappa=0$. κ 通常为非负值，仅在达成一致的概率甚至低于偶然性的情况下取负值。

②由于本文数据集特点，本文适宜选择相关系数作为多样性评价指标，并在代码最后显示了多样性结合子空间的相关系数大小。

```
系统多样性评价指标——相关系数：
[[1.         0.3319074  0.45292832 ... 0.06897976 0.04643683 0.04643683]
 [0.3319074  1.         0.31917854 ... 0.02927274 0.0158998  0.0158998 ]
 [0.45292832 0.31917854 1.         ... 0.07323322 0.05009714 0.05009714]
 ...
 [0.06897976 0.02927274 0.07323322 ... 1.         0.77420966 0.77420966]
 [0.04643683 0.0158998  0.05009714 ... 0.77420966 1.         1.        ]
 [0.04643683 0.0158998  0.05009714 ... 0.77420966 1.         1.        ]]

In[3]:
```

图 5 系统多样性评价指标——相关系数

3.3 集成学习效果提升效果分析

测试错误率计算结果：

表 3 K 折交叉验证的错误率

1	2	3	4	5	6	7	8	9	10
0.0899	0.0790	0.0792	0.0894	0.0892	0.0862	0.0838	0.0970	0.0888	0.0837

由表 2 可以看出，十折交叉验证的包外错误率大概在 0.07~0.08 之间，错误率较低，表明集成学习的效果较好。

集成学习器与个体学习器错误率结果对比：

表 4 集成学习器与个体学习器错误率

算法	错误率
Bagging 集成学习器	0.0865
决策树	0.0912
KNN	0.0947
朴素贝叶斯	0.0912
BP 神经网络	0.0913

将表中数据借助 Python 语言做成散点图更能展示之间差别，其中 Bagging 集成学习器错误率明显低于单个学习器的错误率，表明集成学习器相较于个体学习器分类的准确率有所提升，集成学习的效果较好。

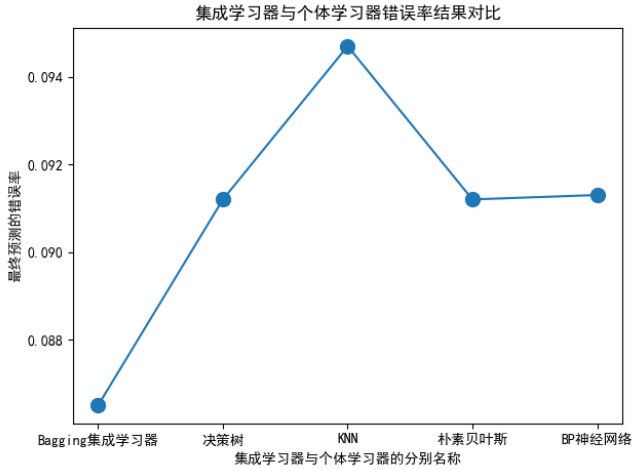

图 6 集成学习器与个体学习器错误率结果对比

代码附录

Bagging 集成学习器代码

```
#集合分类器构建命名
print("WHN 集合分类器运行")
#  根据数据导入各种模块
import numpy as np
import csv
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib #j bolib 模块
from sklearn.model_selection import KFold # K 折交叉验证模块
from collections import Counter
from sklearn.metrics import matthews_corrcoef
from sklearn import tree #  决策树算法模块
from sklearn.neighbors import KNeighborsClassifier # KNN 算法模块
from sklearn.naive_bayes import GaussianNB #  朴素贝叶斯模块中的高斯贝叶斯分类器
from sklearn.neural_network import MLPClassifier # BP 神经网络模块
print("WHN 集合分类器代入模块成功")
print("WHN 集合分类器随机采样生成子数据集....")
#  随机采样生成子数据集
def Sampling(X_train, Y_train):
    m, n = np.shape(X_train)
    index = np.random.randint(m, size=m)
    subSet_X = X_train[index, :]
    subSet_Y = Y_train[index]
    choosed = np.unique(index)
    un_choosed = np.delete(range(m),choosed)
    out_of_beg_X = np.delete(X_train, choosed, axis=0)   #  加密货币价格数据集
    out_of_beg_Y = np.delete(Y_train, choosed, axis=0)   #  加密货币价格数据集
    return subSet_X, subSet_Y, out_of_beg_X, out_of_beg_Y, un_choosed
print("WHN 集合分类器 Bagging 方法，正在分别定义子采样器，分别选择决策树，KNN，朴素贝叶斯，BP 神经网络为个体学习器")
#  定义 Bagging 集成学习器，分别选择决策树，KNN，朴素贝叶斯，BP 神经网络为个体学习器
print("[0-25]WHN 集合分类器子学习器此时是——决策树...")
print("[26-50]WHN 集合分类器子学习器此时是——KNN 近邻算法...")
print("[51-75]WHN 集合分类器子学习器此时是——朴素贝叶斯...")
print("[76-100]WHN 集合分类器子学习器此时是——BP 神经网络...")
def Bagging(X_train, X_test, Y_train, Y_test):
    model = []
```

```
    m = len(X_train)
    Num = 100 #子学习器个数
    testGallary = []
    e_knn = [];e_tree = [];e_gnb = [];e_bp = [];#定义列表，存放个体学习器错误
率
    sumList = [[] for i in range(m)]
    for t in np.arange(Num):
        if t < 25:
            #  子学习器是决策树
            Tree = tree.DecisionTreeClassifier()
            subSet_X, subSet_Y, out_of_beg_X, out_of_beg_Y, un_choosed =
Sampling(X_train, Y_train)
            Tree.fit(subSet_X, subSet_Y)
            y = Tree.predict(out_of_beg_X)
            testGallary.append(Tree.predict(X_test))
            model.append(Tree)
            #  决策树错误率计算
            Tree.fit(X_train, Y_train)
            e_tree.append(1 - Tree.score(X_train, Y_train))
            #  记录预测结果
            i = 0
            for item in un_choosed:
                sumList[item].append(y[i])
                i += 1
        elif t < 50:
                #  子学习器是 KNN
                KNN = KNeighborsClassifier()
                subSet_X, subSet_Y, out_of_beg_X, out_of_beg_Y, un_choosed =
Sampling(X_train, Y_train)
                KNN.fit(subSet_X, subSet_Y)
                y = KNN.predict(out_of_beg_X)
                testGallary.append(KNN.predict(X_test))
                model.append(KNN)
                # KNN 错误率计算
                KNN.fit(X_train, Y_train)
                e_knn.append(1 - KNN.score(X_train, Y_train))
                #  记录预测结果
                i = 0
                for item in un_choosed:
                    sumList[item].append(y[i])
                    i += 1
        elif t < 75:
            #  子学习器是朴素贝叶斯
            GNB = GaussianNB()
```

```
                subSet_X, subSet_Y, out_of_beg_X, out_of_beg_Y, un_choosed =
Sampling(X_train, Y_train)
                GNB.fit(subSet_X, subSet_Y)
                y = GNB.predict(out_of_beg_X)
                testGallary.append(GNB.predict(X_test))
                model.append(GNB)
                #  朴素贝叶斯错误率计算
                GNB.fit(X_train, Y_train)
                e_gnb.append(1 - GNB.score(X_train, Y_train))
                #  记录预测结果
                i = 0
                for item in un_choosed:
                    sumList[item].append(y[i])
                    i += 1
            else:
                #  子学习器是 BP 神经网络
                BP = MLPClassifier()
                subSet_X, subSet_Y, out_of_beg_X, out_of_beg_Y, un_choosed =
Sampling(X_train, Y_train)
                BP.fit(subSet_X, subSet_Y)
                y = BP.predict(out_of_beg_X)
                testGallary.append(BP.predict(X_test))
                model.append(BP)
                # BP 神经网络错误率计算
                BP.fit(X_train, Y_train)
                e_bp.append(1 - BP.score(X_train, Y_train))
                #  记录预测结果
                i = 0
                for item in un_choosed:
                    sumList[item].append(y[i])
                    i += 1
        #  计算集成学习器错误率
        index = []
        for i in range(len(sumList)):
            if sumList[i] != []:
                index.append(i)
        oob = []
        for i in index:
            dic_oob = Counter(sumList[i]).most_common(1)
            oob.append(dic_oob[0][0])
        oob_y = Y_train[index]
        accr_oob = 0
        for item in np.arange(len(oob_y)):
            if oob[item] == oob_y[item]:
```

```
                    accr_oob += 1
        err = 1 - accr_oob / m
        print('集成学习器错误率：')
        print(err)
        # 计算多样性评价指标（相关系数）
        corr_coefficient = np.empty([Num, Num])
        for i in range(Num):
            for j in range(Num):
                corr_coefficient[i, j] = matthews_corrcoef(testGallary[i],
testGallary[j])
        # 个体学习器错误率
        errknn = np.max(e_knn)
        errtree = np.max(e_tree)
        errgnb = np.max(e_gnb)
        errbp = np.max(e_bp)
        return err, corr_coefficient, testGallary, model, errknn, errtree, errgnb, errbp
# 集成学习
# 读取加密货币价格数据集
csv_reader = csv.reader(open('new 加密货币价格数据.csv','r'))
dataSet = []
for line in csv_reader:
        dataSet.append(list(map(float,line)))
DataSet = np.array(dataSet)
m,n = np.shape(DataSet)
htru_X = DataSet[:, 0:n-1]
htru_Y = DataSet[:, n-1]
# 划分训练集和测试集
X_train, X_test, Y_train, Y_test = train_test_split(htru_X, htru_Y, test_size=0.3)
# 十折交叉验证进行训练并计算错误率
kf = KFold(n_splits=10)
#定义列表
e_bag = [];bag_model = [];dicBag = [];Hoob = [];CorrGallary = []
#进行训练
for train, test in kf.split(np.arange(m)):
        X_train = htru_X[train]
        Y_train = htru_Y[train]
        X_test = htru_X[test]
        Y_test = htru_Y[test]
        err, corr_coefficient, testGallary, model,errknn, errtree, errgnb, errbp =
Bagging(X_train, X_test, Y_train, Y_test)
        #计算集成学习的错误率
        for i in range(np.size(testGallary, axis=1)):
            dic_bag = Counter(np.array(testGallary)[:, i]).most_common(1)
            dicBag.append(dic_bag[0][0])
```

```
        count = 0
        n = len(Y_test)
        for i in range(n):
            if dicBag[i] == Y_test[i]:
                count +=1
        e_bag.append(1-count/n)
        bag_model.append(model)
        CorrGallary.append(corr_coefficient)
        Hoob.append(err)
        print("代码断点：K 折交叉验证的每一折")
#集成学习器错误率
best = np.argmax(e_bag)
BagModel = bag_model[best]
errBag = e_bag[best]
print('WHN 集合分类器——决策树错误率:')
print(errtree)
print('WHN 集合分类器——KNN 错误率:')
print(errknn)
print('WHN 集合分类器——朴素贝叶斯错误率：')
print(errgnb)
print('WHN 集合分类器——BP 神经网络错误率:')
print(errbp)
print('WHN 集合分类器——集成学习错误率：')
print(errBag)
print('系统多样性评价指标——相关系数：')
print(corr_coefficient)
```

集成学习效果错误率对比

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
#准备 X 轴和 Y 轴的数据
x_speed = np.array(['Bagging 集成学习器','决策树','KNN','朴素贝叶斯','BP 神经网络'])
y_distance= np.array([0.0865,0.0912,0.0947,0.09120,0.0913])
#设置 x 轴和 Y 轴的标签
plt.xlabel('WHN 集成学习器与个体学习器的分别名称')
plt.ylabel('最终预测的错误率')
#绘制散点图
plt.scatter(x_speed, y_distance,s=100,alpha=10)
plt.title("WHN 集成学习器与个体学习器错误率结果对比")
```

```
plt.plot(x_speed,y_distance)
plt.show()
```

附加问题 1

证明：集成学习器的泛化错误率随着个体学习器数量增加而越来越小？

附加问题 2

证明：分类问题下，前提条件是个体学习器是弱学习器，弱学习器之间的差异性越大，集成效果越来越好？