中南大學
CENTRAL SOUTH UNIVERSITY

模式识别与机器学习课程设计报告

题　　目：模式识别与机器学习课程设计报告

学生姓名：钱兴宇

指导老师：高琰

学　　院：自动化学院

专业班级：人工智能 2101 班

本科生院制

2023 年 6 月

模式识别与机器学习课程设计报告

摘要

近年来, 乳腺癌发病率迅速增长, 已成为世界最常见的癌症。乳腺癌如果在早期被发现, 通常是可以治疗的。如果没有尽早发现乳腺癌并医治, 那么它就很有可能恶化。国际糖尿病方面的研究成果表明, 近年来糖尿病发病率呈现不断的上升趋势。随着人工智能的不断进步, 医疗保健数字化进程加速推进, 基于集成学习技术研究乳腺癌、糖尿病发病机制，构建病理分类模型具有重要的现实意义。

因此本次课程设计主要提出了基于 DecisionStump 的 Adaboost 和 RandomForest 集成学习模型和基于 KNN、Bayes 的 Stacking 集成学习模型。并将上述模型分别应用于 Breast-Cancer 数据集、Kim-India—Diabetes 数据集进行分类。针对不同基学习器与集成学习模型提出了改进方法。

关键词：集成学习　乳腺癌　糖尿病　Adaboost　Stacking　RandomForest

目录

第 1 章　绪论

1.1　项目选题与背景介绍

机器学习与模式识别技术在各个行业被广泛应用，包括工作场所内外，影响着人们日常生活的方方面面，市面上也出现越来越强大的计算平台，计算方法的可用性也逐渐提高。机器学习技术与其它领域的融合已经成为当今的一种趋势，随着大数据呈指数级增长，海量医疗数据的处理无疑加重了医护人员的负担，面对冗余复杂的医疗数据，机器学习展现出了快捷、精准的高超处理水平，因此医疗改革的潜力巨大。借助科技的进步攻克人类健康发展面临的难题，探索疾病的本质，成为医疗改革的一大动力。

糖尿病是最常见的疾病之一，它所带来的高死亡率也引起人们极大的关注，1.5% 的死亡率使其成为全球重点关注的公共卫生问题，虽然医疗水平不断进步，但是糖尿病仍然没有找到被根治的方法，人类对其发病原理也知之甚少。因此对于糖尿病一般以预防为主，在患病后需要服用特定药物和合理饮食对血糖进行控制，才能降低因糖尿病带来的并发症甚至死亡的风险。近年来，研究表明糖尿病患病风险的增加主要与年龄、家族患病史、脂肪过多、日常习惯等有关。世界各地的糖尿病患病率正在上升，尤其是在中低收入国家表现最为明显，糖尿病已不再是高收入国家的特有健康问题。中低收入国家往往缺乏为健康生活方式创造支持性环境的有效政策，国民缺乏获得优质保健的机会，这意味着糖尿病的预防和治疗特别是对低收入群体而言，并没有得到重视。

癌症，一直以来都是全世界人民的噩梦。癌症在给人们带来痛苦的同时，也给患者带来了沉重的经济负担，给患者本人及家属都带来了严重的精神压力。目前癌症的控制与主要产生机制是国内外大多数医疗科研机构研究的主流方向。从女性角度来说，乳腺癌是 154 个国家中最常见的癌症，是 103 个国家中的主要死因，目前已经成为最常见的癌症之一。乳腺癌作为威胁女性健康首要的癌症之一，定期体检、早期诊断以及及时治疗可以有效的防止乳腺癌的发生减少不必要的身体伤害以及财产损失。虽然乳腺癌的死亡率占全部死因的 29.31%，但是乳腺癌的治愈率也达到了 80%-90%。乳腺癌给人们的生活带来了沉重的负担，目前对于乳腺癌的一些诊断方法都无法做到早期诊断的目的，所以当前对于乳腺癌的诊断研究仍然存在很大的空间。

机器学习技术的发展为为乳腺癌、糖尿病的早期诊断提出了一个新的研究思路，医学也不在独立于其他学科之外，与计算机和统计学等相结合可以快速、简单有效的检测是否患病以及风险评估，从而降低患病的几率，减轻不必要的痛苦。

1.2　数据集介绍

1.2.1　Breast-Cancer 数据集

威斯康星乳腺癌数据集是一个二分类问题的数据集，一共有 569 条数据，30 列特征，包括细胞核半径、纹理、周长、面积等特征，1 列结果，标记是否患癌。具体信息

见图

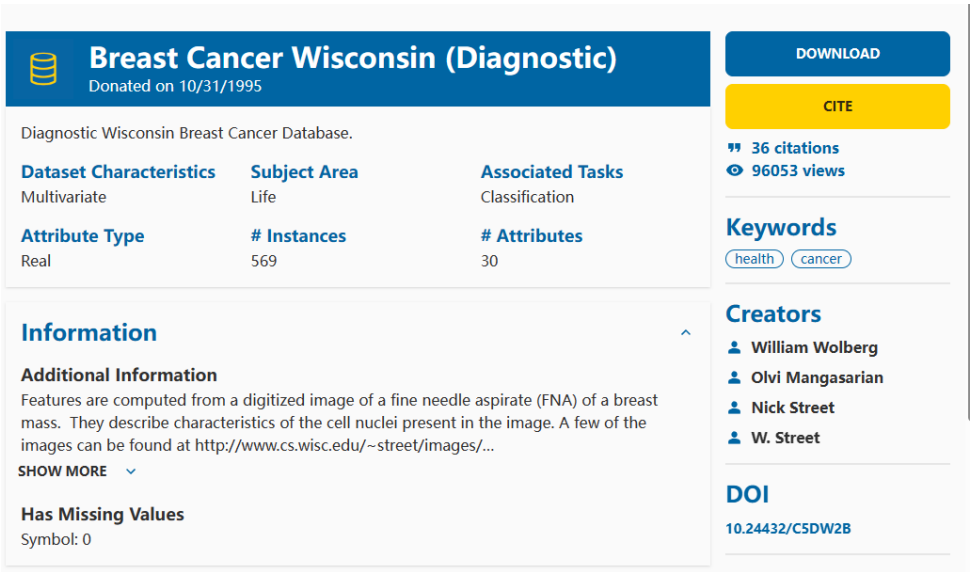

图 1.2.1　Breast-Cancer 数据集简介

1.2.2　Kim-India-Diabetes 数据集

pima-indians-diabetes，该数据集最初来自美国国家糖尿病、消化和肾脏疾病研究所。数据集的目的是基于数据集中包含的某些诊断测量结果，诊断性地预测患者是否患有糖尿病。这里的所有患者都是至少 21 岁的皮马印第安人。数据集由几个医学预测变量和一个目标变量组成，即结果。预测变量包括患者的怀孕次数、BMI、胰岛素水平、年龄等。数据集包含八个特征维度，768 个样本，没有缺失值。数据类别是二分类，存在一定类别不平衡的问题，阳性：阴性为 1:3。

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

数据说明

- Pregnancies：怀孕次数
- Glucose：口服葡萄糖耐量测试中2小时的血浆葡萄糖浓度
- BloodPressure：舒张压（毫米汞柱）
- SkinThickness：三头肌皮肤褶皱厚度（毫米）
- Insulin：2小时血清胰岛素（mu U / ml）
- BMI：体重指数（体重（kg）/（身高（m））^2）
- DiabetesPedigreeFunction：糖尿病谱系功能
- Age：年龄
- Outcome：结果

图 1.2.2　Kim-India-Diabetes 数据集简介

1.3 主要工作

本次机器学习课程设计,所做的主要工作包括数据预处理中的特征提取与数据降维。之后在两个数据集上分别设计使用了基于 DecisionStump 的 Adaboost、RandomForest，基于 KNN、Bayes 的 Stacking 模型进行分类，得到了 F1、fpr、tpr、ConfusionMatrix 等分类指标。通过对上述分类指标进行分析比较，验证了算法改进措施的有效性。

第 2 章　数据预处理

2.1　观察数据分布

对于 BreastCaner 数据集，一共有 30 维特征，我们可以先获取数据的基本信息，通过下图可以看到一共 30 列特征，没有缺失值，每一列特征都是浮点数。

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
 14  smoothness error        569 non-null    float64
 15  compactness error       569 non-null    float64
 16  concavity error         569 non-null    float64
 17  concave points error    569 non-null    float64
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

图 2.1.1　BreastCancer 基本信息

当数据集的维数较多时，直接进行分类往往效果不是很好，因此我们可以通过热图

观察数据相关性，再决定是否进行降维。通过下图可以看出，应该对数据进行降维。

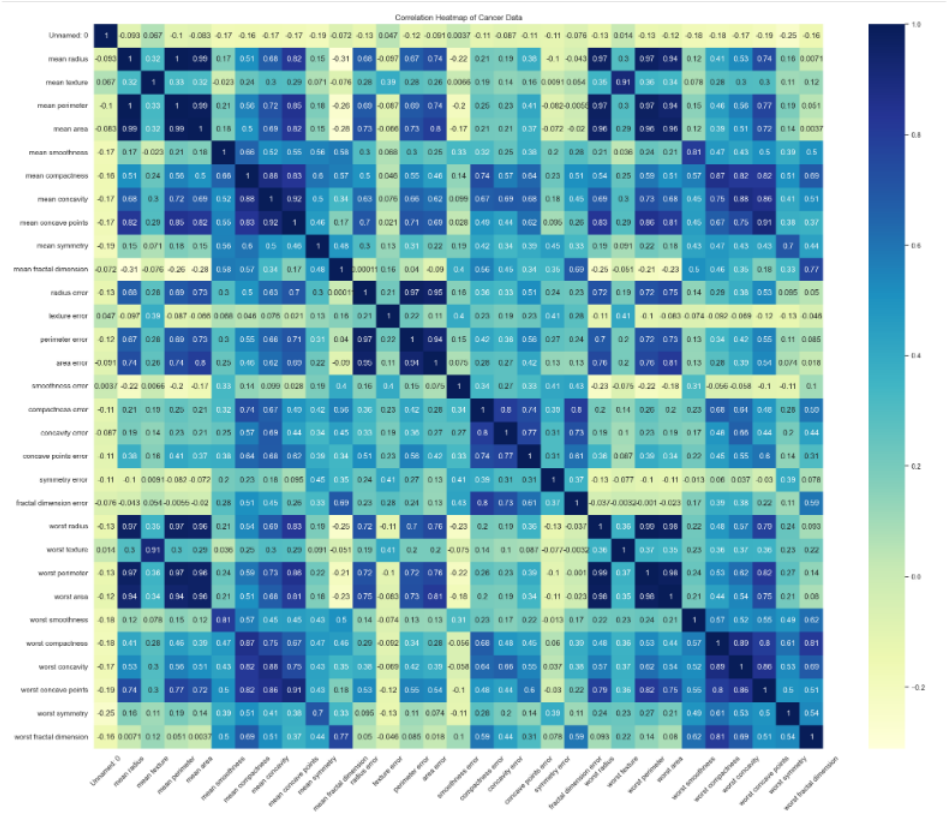

图 2.1.2　BreastCancer 热图

对于 Kim—India-Diabetes 数据集，我们可以先获取数据的基本信息，通过下图可以看到一共有 8 列特征，没有缺失值，大多数为整数特征，最后一列为结果列。

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

图 2.1.3　Kim-India-Diabetes 基本信息

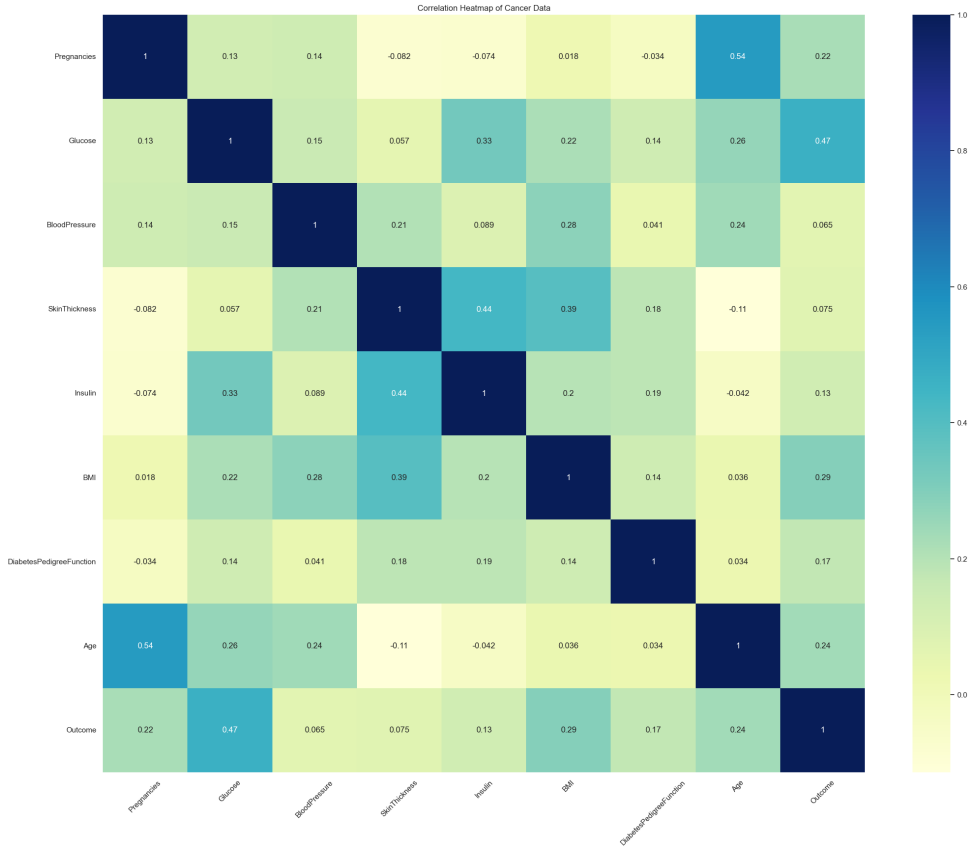

图 2.1.4　Kim-India-Diabetes 热图

2.2　数据降维

PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维全新的正交特征（也被称为主成分）是在原有的 n 维特征的基础上构造出来的。PCA 从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择是与数据本身是密切相关的。其中，第一个新坐标轴选择是原始数据中方差最大的方向，第二个新坐标轴选取是与第一个坐标轴正交的平面中使得方差最大的，第三个轴是与第 1,2 个轴正交的平面中方差最大的。依次类推，可以得到 n 个这样的坐标轴。通过这种方式获得的新的坐标轴，我们发现，大部分方差都包含在前面 k 个坐标轴中，后面的坐标轴所含的方差几乎为 0。于是，我们可以忽略余下的坐标轴，只保留前面 k 个含有绝大部分方差的坐标轴。

$$\begin{aligned} &\max \omega \operatorname{tr}\left(W^T X X^T W\right) \\ &\text{s.t. } W^T W = I \end{aligned} \tag{2.1}$$

为了实现上述过程，我们需要计算样本的协方差矩阵。样本 X 和样本 Y 的协方差定义如下。

$$\operatorname{Cov}(X,Y) = E[(X - E(X))(Y - E(Y))] \tag{2.2}$$

$$\mathrm{Cov}(X,Y)=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) \tag{2.3}$$

然后计算协方差矩阵的特征值与特征向量，并对特征值从大到小排序，保留最大的 n 个特征向量（也就是 n 个主成分），然后将数据转化到这 n 个特征向量构建的新特征空间中。

PCA 实现代码如下：

```
def _PCA(attributes, compents):


    # 对 X 进行去中心化
    attributes = standard_scaler(attributes)
    # 计算样本的协方差矩阵 X*X.T
    covx = np.dot(attributes, attributes.T)
    # 求矩阵的特征值和特征向量
    lamda, Vv = np.linalg.eigh(covx)
    index = np.argsort(-lamda)[:compents]
    diag_lamda = np.sqrt(np.diag(-np.sort(-lamda)[:compents]))
    # 取出对应特征向量
    V_selected = Vv[:, index]
    Zz = V_selected.dot(diag_lamda)
    #print(np.sum(V_selected))
    print(np.shape(Zz))
    return Zz
```

第 3 章　算法流程与实现

3.1　Adaboost

Boosting 方法是一种用来提高弱分类算法准确度的方法, 这种方法通过构造一个预测函数系列, 然后以一定的方式将他们组合成一个预测函数。自适应增强模型（Adaptive Boosting，AdaBoost）由 Freund 和 Schapire 在 1997 年被开发出。该算法是有效而实用的 Boosting 算法的一种，它是按照顺序以一种高强度自适应的方式对弱学习器进行训练，最初为所有训练数据分配相等的权重，生成一个弱学习器，随后使用该训练示例用于测试弱学习器和调整权重值，通过进行多次的数据训练和调整权重值后，最终形成误差较小的强学习器。[1]框架图如下：

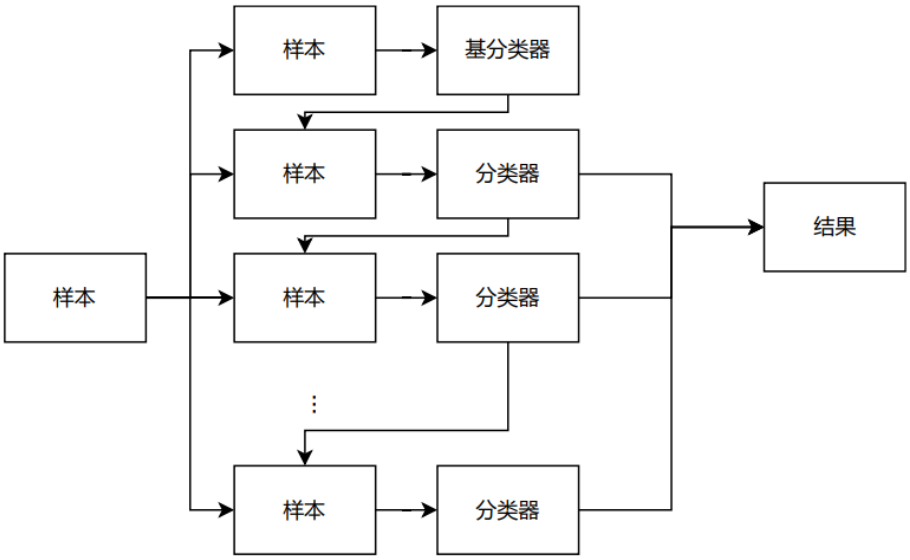

图 3.1.1　Adaboost 框架图

AdaBoost 算法会得到 T 个弱学习器 $h_i(x)$ ，然后将他们加权组合在一起得到最终的强学习器 H(x).

$$H(x) = \sum_{i=1}^{T} \alpha_i h_i(x) i = 1, 2, ... T \tag{3.1}$$

Adaboost 最初讲所有样本、基分类器初始化为相同权重，然后进行训练迭代。通过求得错误率 ε_t，给出每个学习器 $h_t(x)$ 的权重 α_t。

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right) \tag{3.2}$$

并且根据分类结果对样本权重进行调整，分类错误的样本权重提高[2]，分类正确的样本权重降低。调整公式如下：

$$D_{t+1}(x) = \frac{D_t(x) * e^{-\alpha_t f(x) h_t(x)}}{Z_t} \tag{3.3}$$

训练结束后最终得到的模型为：

$$H(x) = \text{sign}\left\{\sum_{t=1}^{T} \alpha_t h_t(x)\right\} \tag{3.4}$$

从上述过程中可以看出 Adaboost 训练过程是一个串行训练的过程，下一次基学习器的训练总是基于这次分类结果调整后权重的样本。从偏差-方差的角度来看，boosting 主要关注降低偏差，因此 boosting 可以基于泛化性能相当弱的学习器构建出效果比较好的强学习器。

在本次课程设计中采用 DecisionStump 作为 Adaboost 的基学习器，DecisionStump 是只有一个节点的决策树，通过将输入与阈值相比较输出 0 或 1. 其代码如下：

```
# 决策树桩
class DecisionStump:
    def __init__(self):
        # 基于划分阈值决定
        self.label = 1
        # 特征索引
        self.feature_index = None
        # 特征划分阈值
        self.threshold = None
        # 权重
        self.alpha = None
```

Adaboost 实现代码如下：

```
# 决策树桩
class AdaBoost:
    # 弱分类器集合
    def __init__(self, n_estimators=5):
        self.n_estimators = n_estimators
        self.estimators = []

    # AdaBoost算法
    def fit(self, X, y):
        # 获取样本数、维数
```

```
        m, n = X.shape
        # 初始化权重相等
        w = np.full(m, 1 / m)
        for _ in range(self.n_estimators):
            # 训练一个弱分类器：决策树桩
            estimator = DecisionStump()
            # 设定一个最小化误差率（初始设为∞）
            min_error = float('inf')
            # 遍历数据集特征，根据最小分类误差率选择最优特征作
                为分类依据
            for i in range(n):
                # 获取特征值
                values = np.expand_dims(X[:, i], axis=1)
                # 特征取值去重
                unique_values = np.unique(values)
                # 尝试将每一个特征值作为分类标准
                for threshold in unique_values:
                    label = 1
                    # 初始化所有预测值为1
                    pred = np.ones(np.shape(y))
                    # 小于分类阈值的预测值为-1
                    pred[X[:, i] < threshold] = -1
                    # 计算误差率
                    error = sum(w[y != pred])
                    # 误差率大于0.5，直接翻转正负预测
                    if error > 0.5:
                        error = 1 - error
                        label = -1
                    # 保存最小误差率相关参数
                    if error < min_error:
                        estimator.label = label
                        estimator.threshold = threshold
                        estimator.feature_index = i
                        min_error = error
            # 计算基分类器的权重
            estimator.alpha = 0.5 * np.log((1.0 - min_error) /
                max(min_error, 1e-12))
```

```
            # 初始化所有的预测值为 1
            preds = np.ones(np.shape(y))
            # 获取所有小于阈值的负类索引
            negative_idx = (estimator.label * X[:, estimator.
                feature_index] < estimator.label * estimator.
                threshold)
            # 将负类设置为 -1
            preds[negative_idx] = -1
            # 更新样本权重
            w *= np.exp(-estimator.alpha * y * preds)
            w /= np.sum(w)
            # 保存该弱分类器
            self.estimators.append(estimator)
```

3.2 RandomForest

随机森林是在决策树的基础上实现的，是种 Bagging 类的有监督集成学习方法，通过将多棵决策树集成在一起提高预测准确率，是最常用的集成学习模型。[3]

决策树是种基于信息熵的树形结构，对一个特征按照信息量的大小划分在不同的子树中，当子树中所有样本均属于同一类别时停止划分，常用的特征划分算法有 ID3、C4.5、CART 等。

信息熵用于衡量变量的不确定性，其中 p_i 是第 i 个事件发生的概率，k 是事件的总数。信息熵值越大表示信息量越大，信息的价值越高。[4]

$$H(x) = \sum_{i=1}^{k} p_i I\left(X = x_i\right) = -\sum_{i=1}^{k} p_i \log p_i \tag{3.5}$$

ID3 是每次特征选择时使用信息增益最大的进行分裂。对每个特征计算信息增益，每次分裂选择信息增益最大的特征，这样做是为了消除样本间的不确定性。信息增益 IG 如下所示，其中 H(X) 是信息熵，X 为样本，p 为概率。

$$IG(F) = H(x) - \sum_{i} \frac{|X_i|}{|X|} H\left(X_i\right) \tag{3.6}$$

C4.5 是在 ID3 方法上进一步的改进，考虑到 ID3 算法在特征特别多时，划分得会特别多出现过拟合问题，因此除以特征的信息熵，如下公式所示。针对缺失值，C4.5 采用将它划分到每个结点的方法，以不同概率划分到不同子节点中。

$$C45(F) = \frac{IG(F)}{H(F)} \tag{3.7}$$

CART 采用基尼系数最小的进行分裂，核心思想是关注样本被预测错误的概率，计算出[5] 样本中两两不相同的概率。计算公式如下，n 为类别数，p_i 为类别 i 预测成功的概率。因此，这个值越小效果意味着样本是同类可能性越大。

$$\begin{aligned} &\text{Gini}\left(F_j\right)=\sum_{i=0}^{n} p_i\left(1-p_i\right)=1-\sum_{i=0}^{n} p_i^2 \\ &\text{s.t.}\ \sum_{i=1}^{n} p_i=1 \end{aligned} \tag{3.8}$$

随机森林具有很高的随机性，每次采样以及每次分裂特征选择均不同，可有效应用在大规模的样本上，具有很高的准确率。所以森林中每棵树的结构很简单的情况下也可以取得较号的成果。但同时也可能会在数据集上过拟合，导致实际准确率较低。在样本值较多的特征上随机森林分裂性能较差，需要进一步处理。

另一方面，在此次课程设计中考虑将 Adaboost 中调整样本权重的方法引入随机森林中，也即在每次训练的过程中

随机森林结构图如下。

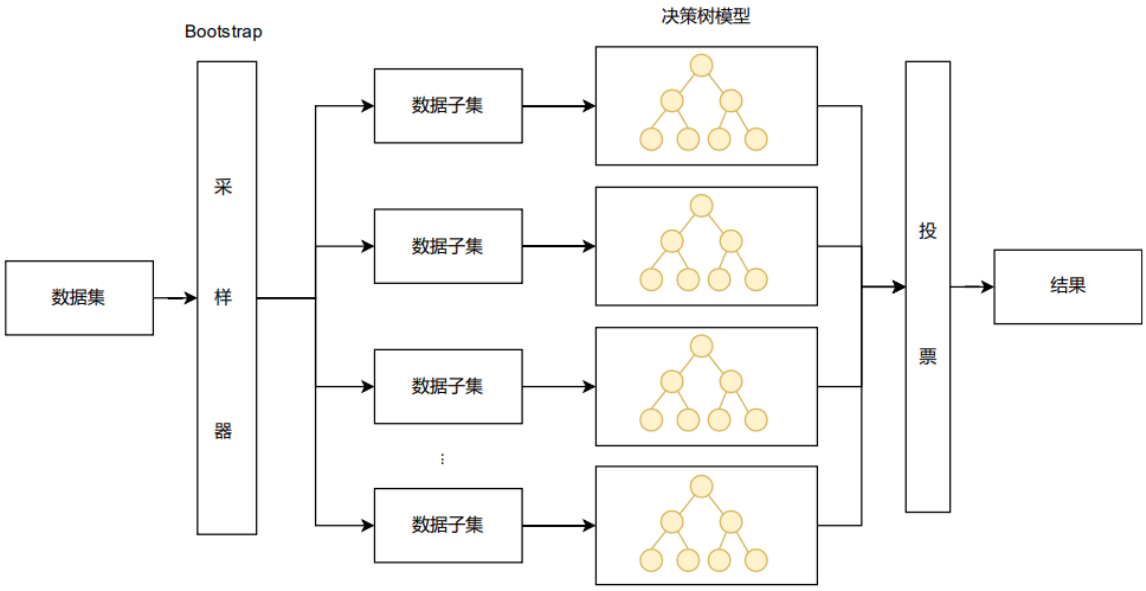

图 3.2.1　RandomForest 框架图

随机森林实现代码如下：

```
# 随机森林算法
    def fit_origin_RF(self, X, y):
        # 获取样本数、维数
        m, n = X.shape

        for _ in range(self.n_estimators):
```

```
        # 训练一个弱分类器：决策树桩
        estimator = DecisionStump()
        # 设定一个最小化误差率（初始设为∞）
        min_error = float('inf')
        k = 5
        rand = np.random.randint(0, n, k)
        # 遍历数据集特征，根据最小分类误差率选择最优特征作
            为分类依据
        for i in rand:
            # 获取特征值
            values = np.expand_dims(X[:, i], axis=1)
            # 特征取值去重
            unique_values = np.unique(values)
            # 尝试将每一个特征值作为分类阈值
            for threshold in unique_values:
                label = 1
                # 初始化所有预测值为1
                pred = np.ones(np.shape(y))
                # 小于分类阈值的预测值为-1
                pred[X[:, i] < threshold] = -1
                # 计算误差率
                error = sum(w[y != pred])
                # 误差率大于0.5，直接翻转正负预测
                if error > 0.5:
                    error = 1 - error
                    label = -1
                # 保存最小误差率相关参数
                if error < min_error:
                    estimator.label = label
                    estimator.threshold = threshold
                    estimator.feature_index = i
                    min_error = error

        # 初始化所有的预测值为 1
        preds = np.ones(np.shape(y))
        # 获取所有小于阈值的负类索引
        negative_idx = (estimator.label * X[:, estimator.
```

```
                feature_index] < estimator.label * estimator.
                threshold)
            # 将负类设置为 -1
            preds[negative_idx] = -1

            self.estimators.append(estimator)
```

3.3 KNN-Bayes Stacking

3.3.1 KNN

k 近邻法 (k-nearest neighbor, k-NN) 是 1967 年由 Cover T 和 Hart P 提出的一种基本分类与回归方法。它的工作原理是：存在一个样本数据集合，也称作为训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一个数据与所属分类的对应关系。

KNN 算法是一种非常特别的机器学习算法，因为它没有一般意义上的学习过程。它的工作原理是利用训练数据对特征向量空间进行划分，并将划分结果作为最终算法模型。存在一个样本数据集合，也称作训练样本集，并且样本集中的每个数据都存在标签，即我们知道样本集中每一数据与所属分类的对应关系。输入没有标签的数据后，将这个没有标签的数据的每个特征与样本集中的数据对应的特征进行比较，然后提取样本中特征最相近的数据（最近邻）的分类标签。一般而言，我们只选择样本数据集中前 k 个最相似的数据，这就是 KNN 算法中 k 的由来，通常 k 是不大于 20 的整数。最后，选择 k 个最相似数据中出现次数最多的类别，作为新数据的分类。

KNN 算法的预测过程十分简单，对于一个需要预测的输入向量 x，我们只需要在[6]训练数据集中寻找 k 个与向量 x 距离最近的向量的集合，然后把 x 的类别预测为这 k 个样本中类别最多的那一类。

KNN 实现代码如下：

```
def knn_predict1(sample, k=9):
    # print(sample)
    dists = []
    for i, a in enumerate(X_train):
        dists.append((dist(sample, a), Y_train[i]))
    dists.sort()
    # print(dists)
    dist_k_min = dists[:k]
    tmp = 0
    for a in dist_k_min:
        if a[1] == 0:
```

```
            tmp += 1
        else:
            tmp -= 1
    # print(tmp)
    # print(int(tmp < 0))
    return float(-tmp / 9)
```

3.3.2 Bayes

朴素贝叶斯方法是在各个输入特征量相互独立的前提夏根据贝叶斯定理提出的一种分类方法。对于训练数据集，在输入特征相互独立独立的条件下求出每个类别的后验概率，并且将最大后验概率的类别最为预测类别。

先验概率：

$$P(Y=c), c \in \{-1, 1\} \tag{3.9}$$

条件概率：

$$P(X=x \mid Y=c)=P\left(X^{(1)}=x^{(1)}, \ldots, X^{(n)}=x^{(n)} \mid Y=c\right) \tag{3.10}$$

朴素贝叶斯对上述条件概率做了如下独立性假设：

$$\begin{aligned} P(X=x \mid Y=c) &= P\left(X^{(1)}=x^{(1)}, \ldots, X^{(n)}=x^{(n)} \mid Y=c\right) \\ &= \prod_{j=1}^{n} P\left(X^{(j)}=x^{(j)} \mid Y=c\right) \end{aligned} \tag{3.11}$$

后验概率：

$$P(Y=c \mid X=x)=\frac{P(X=x \mid Y=c) P(Y=c)}{\sum_{c \in\{-1,+1\}} P(X=x \mid Y=c) P(Y=c)} \tag{3.12}$$

最终朴素贝叶斯分类器可以表示为：

$$y=\arg \max _{c} P(Y=c) \prod_{j=1}^{n} P\left(X^{(j)}=x^{(j)} \mid Y=c\right) \tag{3.13}$$

朴素贝叶斯实现代码如下：

```
def bayes_predict1(sample):
    tmp = [0, 0]
    for i in range(class_num):
        tmp[i] = np.log(cnt[i])
        for j in range(feature_len):
```

```
            tmp[i] += np.log(1 / math.sqrt(2 * math.pi * va[i
                ][j])) - (sample[j] - me[i][j]) * (
                        sample[j] - me[i][j]) / (2 * va[i][j])

    # print(tmp[0], tmp[1])
    # print((int)(tmp[0] <= tmp[1]))
    return tmp[1] - tmp[0]

def bayes_train(data, y):
    global cnt
    cnt = [0, 0]
    o = []
    for i in range(class_num): # 枚举类别
        tmp = []
        for j in range(feature_len): # 枚举特征属性
            tmp1 = []
            for k, sample in enumerate(data):
                if y[k] == i:
                    tmp1.append(sample[j]) # 存储该类别下该特
                        征的值
                    cnt[i] += 1 # 类别概率计算
            tmp.append(tmp1)
        o.append(tmp)

    for i in range(class_num):
        cnt[i] = float(cnt[i]) # 计算类别先验概率

    global me # 特定类别下特征属性的均值
    global va # 特定类别下特征属性的方差
    me = np.zeros((class_num, feature_len), dtype=float)
    va = np.zeros((class_num, feature_len), dtype=float)

    for i in range(class_num):
        for j in range(feature_len):
            # 利用numpy计算均值方差
            me[i][j] = np.mean(np.array(o[i][j]))
            va[i][j] = np.var(np.array(o[i][j]))
```

Stacking 方法是采用组合策略将多个模型组合起来，将多个模型输出转化成一个输出结果，可采用线性组合、逻辑回归等的组合策略，Stacking 方法处理流程如下图所示，其中采用的是随机抽取方式，用任意多的样本训练其中一个模型，一般情况下会用全部样本。每个模型得到一个输出结果后再训练一个模型，得到最终的输出结果。在本次课程设计中，我将 KNN 和 Bayes 两种基学习器的结果作为第二层学习器的输入，再进行分类。

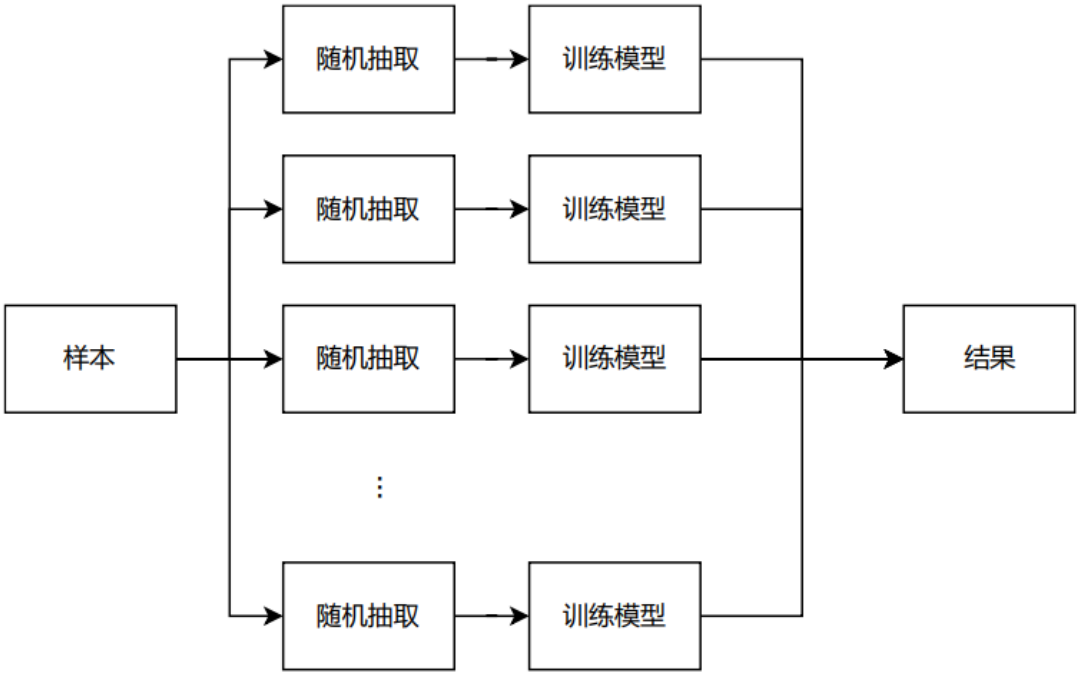

图 3.3.1　Stacking 框架图

KNN-Bayes Stacking 实现代码如下：

```
def train(X_train, y_train):
    global a
    a = 0.05
    global b
    b = 0.3
    epoch = 8 # 训练周期
    lr = 0.005 # 学习率
    global w
    w = 0.5
    X = []
    Y = []
    for i in range(epoch): # 梯度下降法
        ax = 0
```

```
        by = 0
        cnt = 0
        for j, sample in enumerate(X_train):
            x = bayes_predict1(sample) # bayes预测后验概率之差
                作为输入
            y = knn_predict1(sample) # knn邻居投票票数作为输入
            z = int(a * x + b * y > 0)
            if(y_train[j] == z):
                cnt += 1
            # 对参数求梯度
            if(y_train[j] == 1):
                ax += (a * x + b * y - y_train[j] + w) * x * 2
                by += (a * x + b * y - y_train[j] + w) * y * 2
            else :
                ax += (a * x + b * y - y_train[j] + w) * x
                by += (a * x + b * y - y_train[j] + w) * y
        ax /= X_train.shape[0]
        by /= X_train.shape[0]
        cnt /= X_train.shape[0]
        X.append(i)
        Y.append(cnt)
        print(ax, by)
        # 对参数进行梯度更新
        a -= lr * ax
        b -= lr * by
```

第 4 章　结果分析

4.1　评价指标

在本次课程设计中我选取 Recall、F1、ConfusionMatrix 等常见的分类指标作为评价指标。部分评价指标的计算公式如下：

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.1}$$

$$F1 = \frac{2 \text{ Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.3}$$

混淆矩阵是机器学习中总结分类模型预测结果的情形分析表，以矩阵形式将数据集中的记录按照真实的类别与分类模型预测的类别判断两个标准进行汇总。其中矩阵的行表示真实值，矩阵的列表示预测值。二分类问题的混淆矩阵如下图所示，X 轴为预测标签的类别，Y 轴为真实标签的类别。网格中数数量表示真实类别为 i 的情况下预测类别为 j 的数量，颜色越深表示预测的结果数量越多。

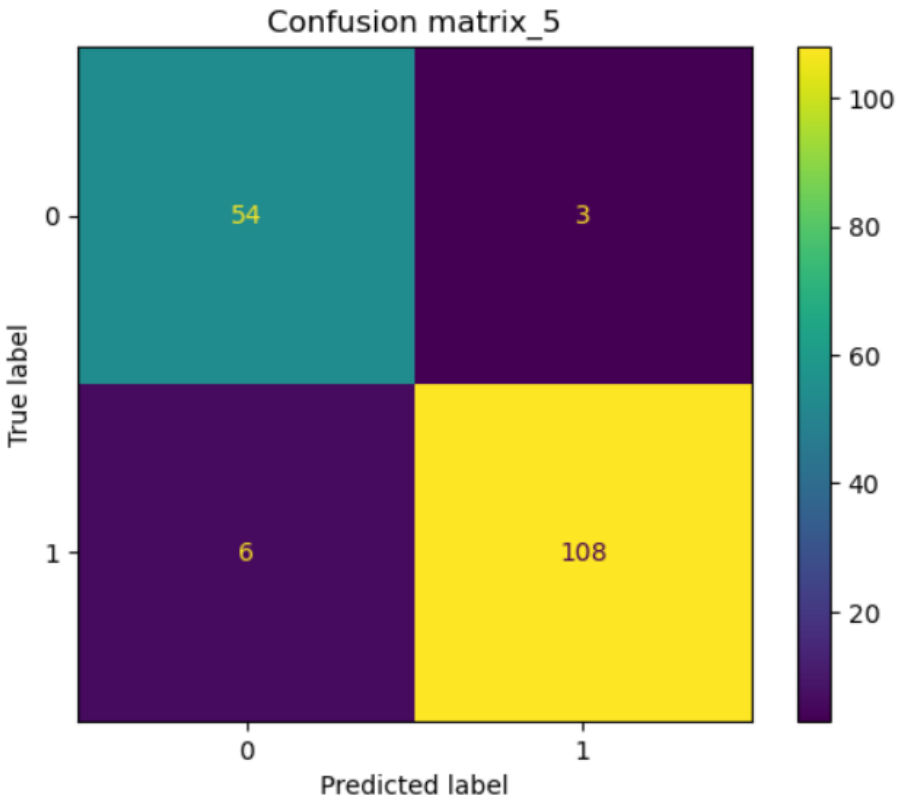

图 4.1.1　混淆矩阵

评价指标计算的代码实现如下：

```
def calculate_recall_fpr_tpr(y_true, y_pred):
    y_true = y_true.flatten()
    y_pred = y_pred.flatten()
    Tp = 0
    Fp = 0
    Tn = 0
    Fn = 0
    for label, pred in zip(y_true, y_pred):
        if (label == -1) and (pred == -1):
            Tp = Tp + 1
        elif (label == 1) and (pred == -1):
            Fp = Fp + 1
        elif (label == 1) and (pred == 1):
            Tn = Tn + 1
        elif (label == -1) and (pred == 1):
            Fn = Fn + 1
        else:
            print('Labels error!')
            return -1
    recall = Tp / (Tp + Fn)
    fpr = Fp / (Fp + Tn)
    tpr = Tp / (Tp + Fn)
    print("Recall:", recall)
    print("FPR:", fpr)
    print("TPR:", tpr)


def ROC(y_true, y_score):
    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr)
    print("AUC:", roc_auc)
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC
       curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='—
       ')
```

```
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()


def conf_matrix(y_test, y_pred, t):
    cm = confusion_matrix(y_test, y_pred)  # 混淆矩阵
    print("Confusion matrix of Label is \n", cm)
    # confusion_matrix(混淆矩阵), display_labels(标签名称列表)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
       display_labels=['0', '1'])
    # 画出混淆矩阵
    disp.plot()
    plt.title(f"Confusion matrix_{t}")
    # 保存
    # plt.savefig(f"Confusion_matrix_RF{t}")
    # 显示
    plt.show()
```

4.2 实验结果

首先在 BreastCancer 数据集上使用 Adaboost 进行分类，分类结果如下。

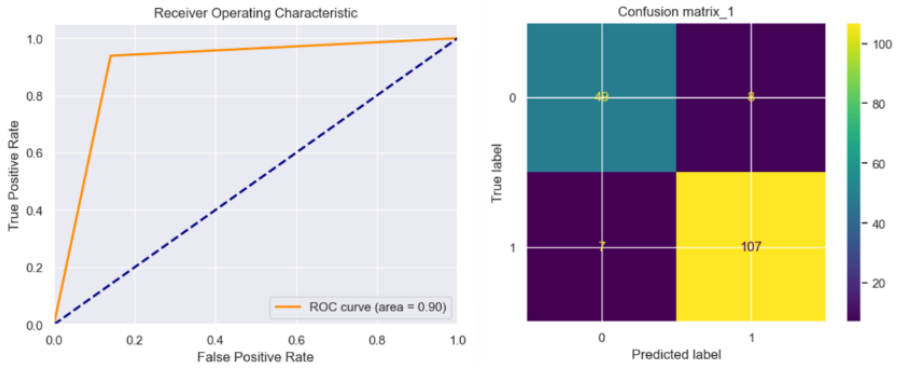

图 4.2.1 一个基学习器分类结果

Adaboost 其他分类指标结果表 2-4.

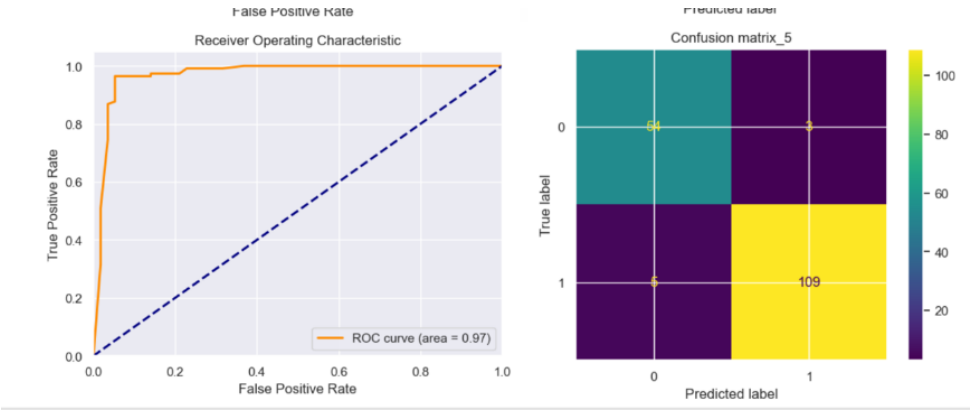

图 4.2.2　五个基学习器分类结果

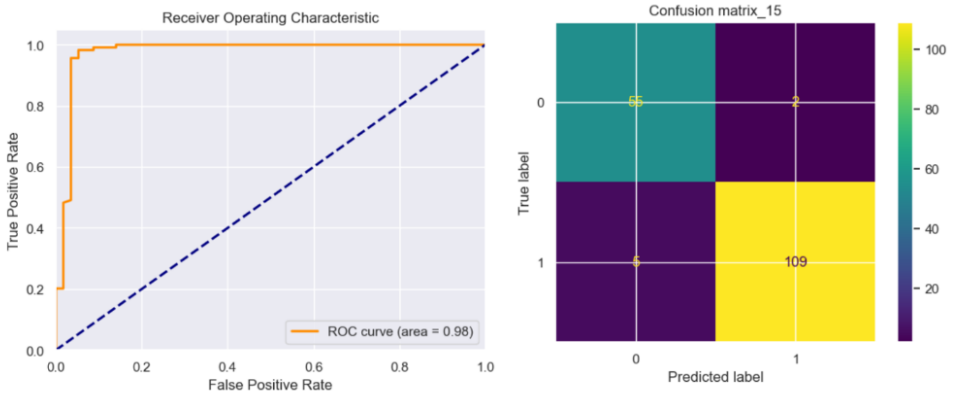

图 4.2.3　十五个基学习器分类结果

表 4.2.1　Adaboost 其他分类指标

弱分类器数量	Recall	FPR	TPR	准确率	F1
1	0.8596	0.0614	0.8596	0.9123	0.9345
5	0.9474	0.0439	0.9474	0.9532	0.9646
15	0.9649	0.0439	0.9649	0.9591	0.9689

降维后 Adaboost 分类结果如下.

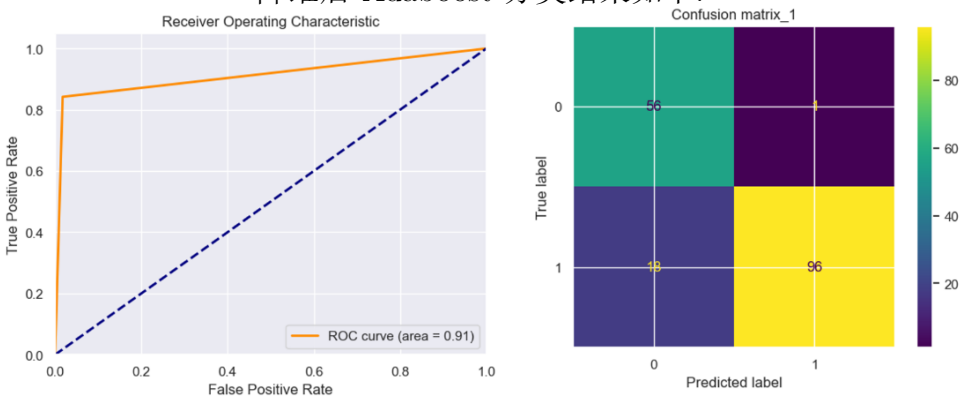

图 4.2.4　降维后一个基学习器分类结果

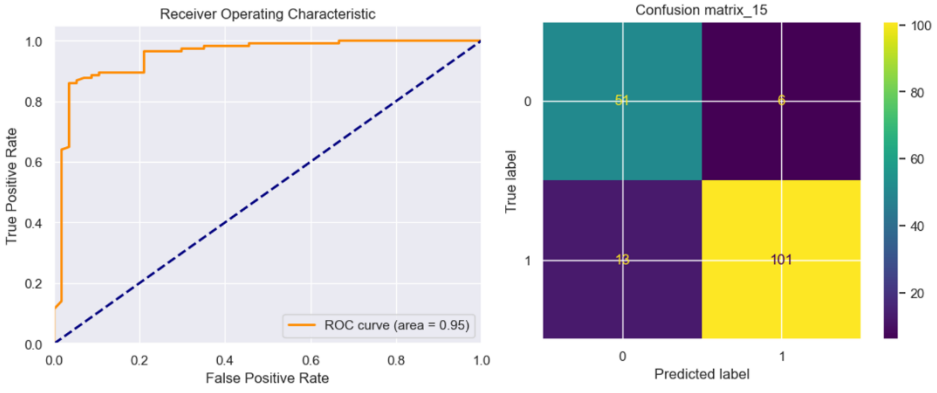

图 4.2.5　降维后十五个基学习器分类结果

随机森林分类结果如下，部分分类指标见表 2-6.

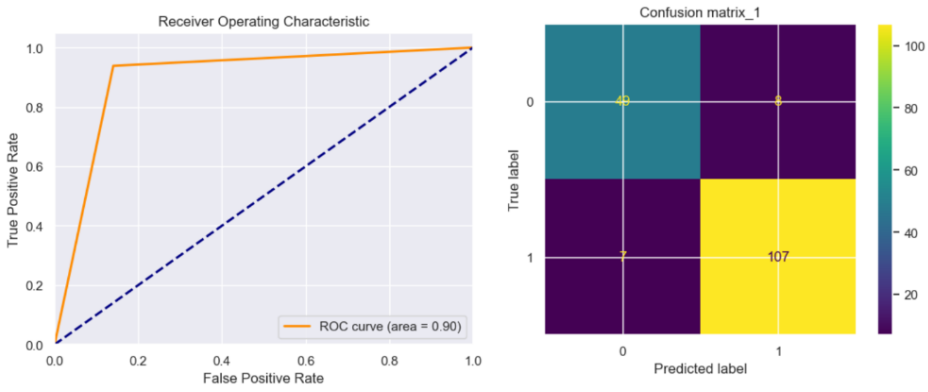

图 4.2.6　一棵树分类结果

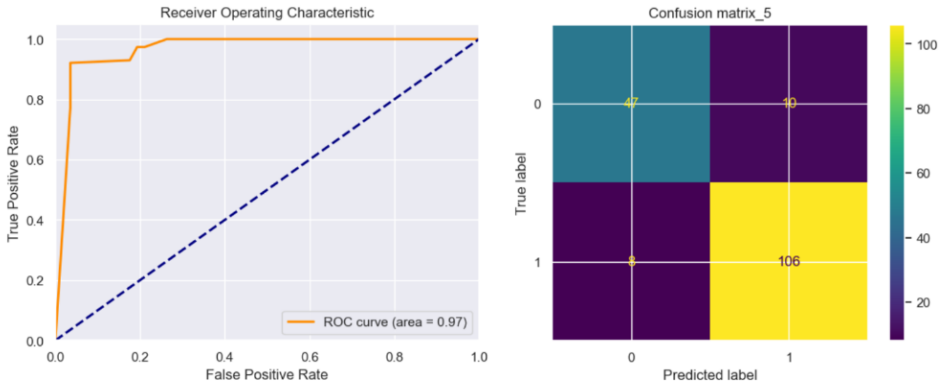

图 4.2.7　五棵树分类结果

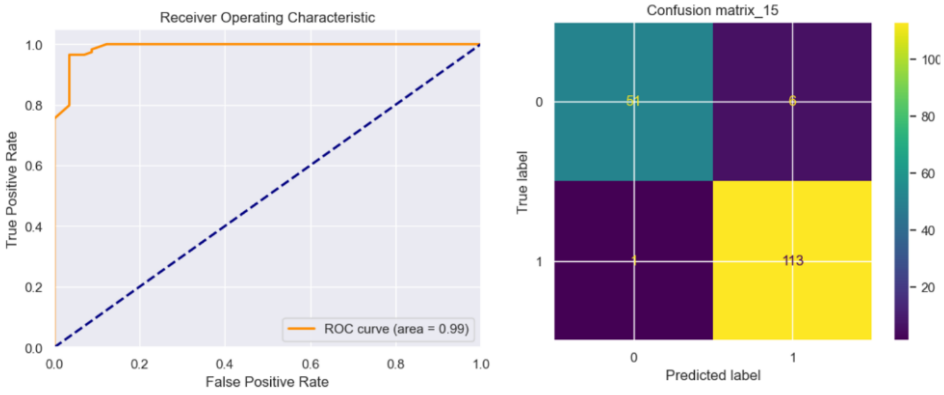

图 4.2.8　十五棵树分类结果

表 4.2.2　RandomForest 其他分类指标

弱分类器数量	Recall	FPR	TPR	准确率	F1
1	0.8596	0.0614	0.8596	0.9123	0.9345
5	0.8246	0.0702	0.8246	0.8947	0.9217
15	0.8947	0.0088	0.8947	0.9591	0.9700

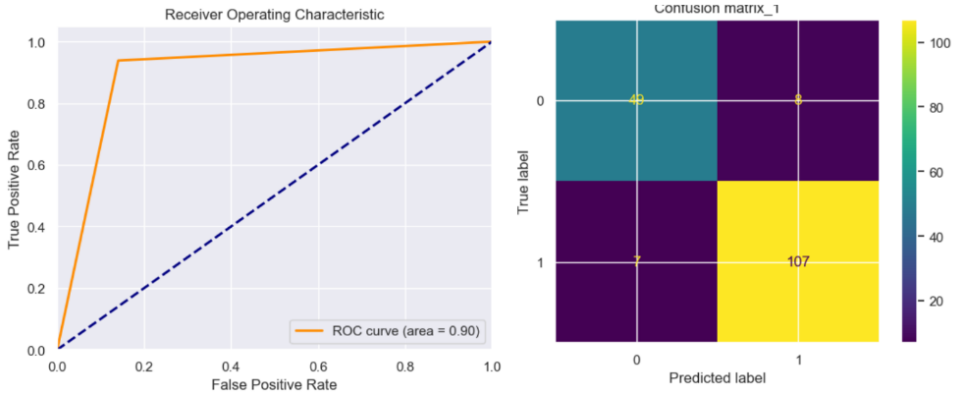

图 4.2.9　改进后一棵树分类结果

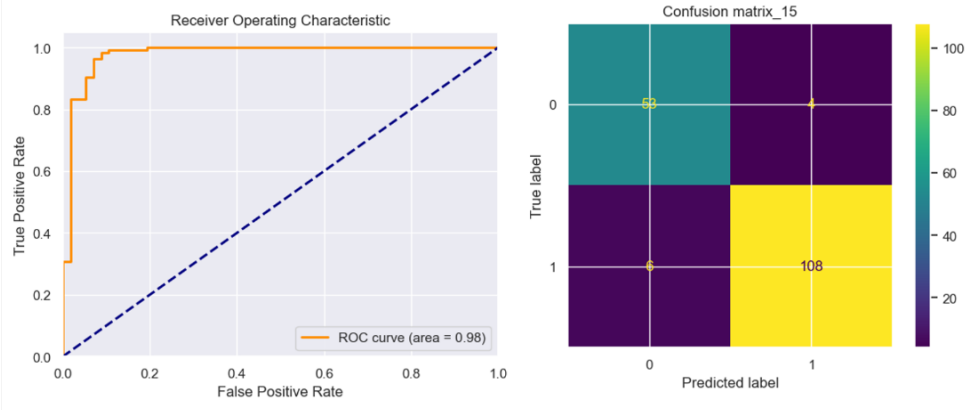

图 4.2.10　改进后十五棵树分类结果

下面是对 Kim-India-Diabetes 数据集分类的结果展示。

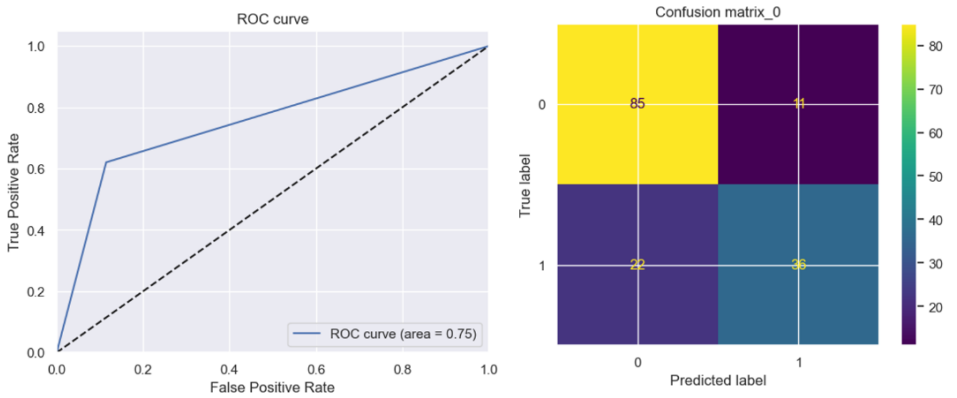

图 4.2.11　KNN 分类结果

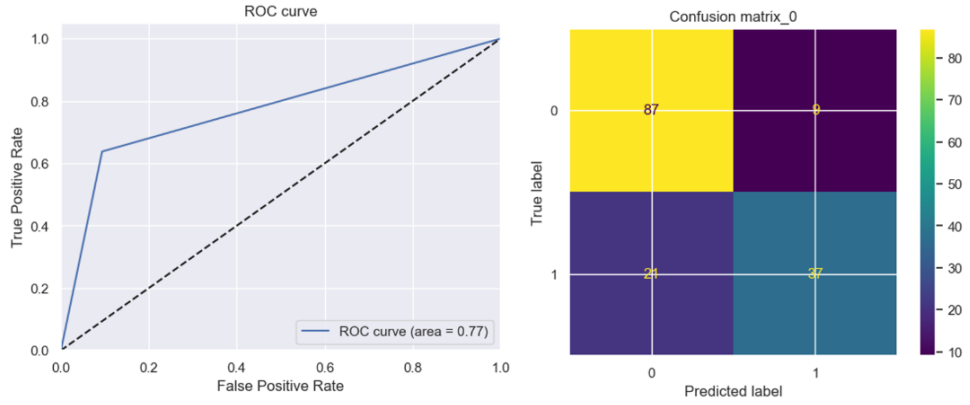

图 4.2.12　降维后 KNN 分类结果

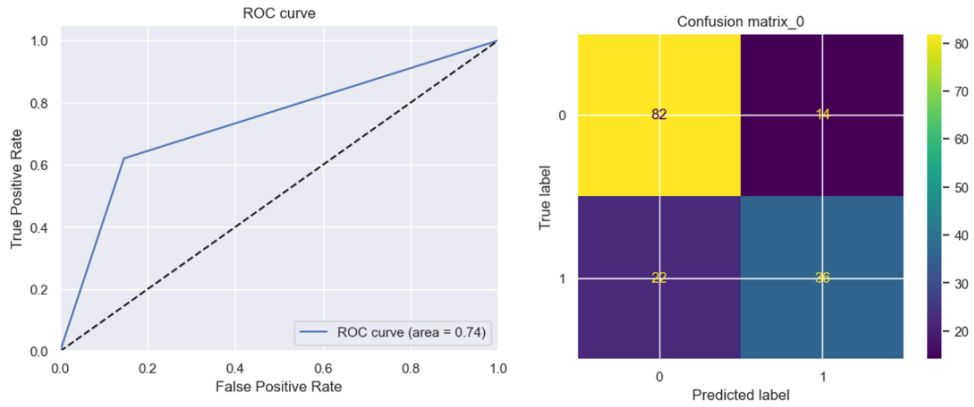

图 4.2.13　贝叶斯分类结果

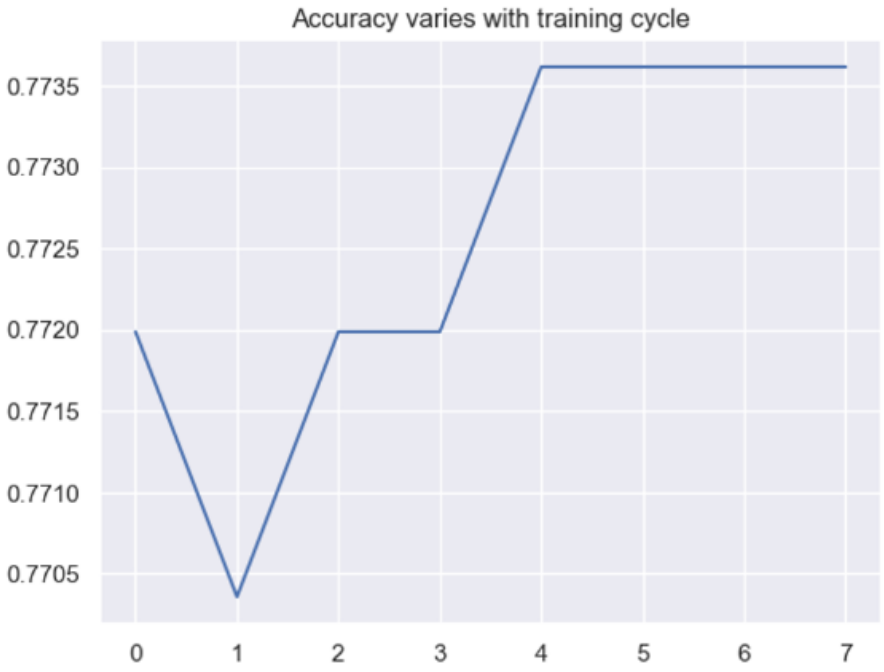

图 4.2.14　训练精度曲线

表 4.2.3　分类器性能指标

方法	AUC	准确率	精确率	召回率	F1 值
KNN	0.7531	0.7857	0.7660	0.6207	0.6857
降维后 KNN	0.7721	0.8052	0.8043	0.6379	0.7115
Bayes	0.7374	0.7662	0.7200	0.6207	0.6667
Stacking	0.7651	0.7922	0.7600	0.6552	0.7037

第 5 章　总结与展望

5.1　总结

在本次课程设计过程中，我进一步加深了对于机器学习与模式识别各种算法的理解，同时页完成了不同数据集上多种分类模型的构建，并且取得了不错的效果。

通过这次课程设计，在发现问题、解决问题的过程中我逐步增强了自己写代码、改代码的能力，并且能通过互联网等多种渠道查阅博客、文献来解决自己心中疑惑。我也更加体会到了将理论知识与实践相结合的重要性，今后学习中一定要知行合一，利用在课堂上学到的理论知识指导实践，同时又在实践中不断加深自己对理论知识的理解。

尤其是在课程设计中我主要构建了多个集成学习的模型，在这个过程中，我逐步培养了对于集成学习的兴趣。当我通过组合几个分类能力并不强的基学习器达到较好的分类效果时，我的内心中满是欣喜与激动。我将在以后的课程学习中进一步探索集成学习的无限魅力！

通过这门课程设计，我不仅学到了机器学习理论知识，还提高了自己的实践能力。我深刻认识到理论与实践相结合的重要性，只有将所学的知识应用于实际问题中，才能真正掌握和理解机器学习的精髓。我也意识到机器学习领域的快速发展和广泛应用，未来将会有更多的机会和挑战等待着我们。

最后，我要感谢所有在这门课程设计中给予我帮助和支持的老师和同学们。他们的指导和交流使我受益匪浅，也增添了学习的乐趣。我会继续保持对机器学习的兴趣和热情，不断学习和探索，在未来的学习和工作中将机器学习应用于实际问题，为社会的发展做出贡献！

5.2　展望

虽然在两个数据集上最终分类效果还不错，但是仍然存在一些可以改进的地方。

1. 解决类别不平衡的问题。

2. 如何选择合理的超参数的问题。

为了解决上述问题，后续可以首先进行类别不平衡的调整。使用过采样，欠采样或者使用 SMOTE 算法人为生成新样本，增大少数类别样本的数量，适当减少多数类别样本的数量，均衡化样本分布。或者在学习器学习样本分布的过程中增大少数类别学习的权重，减少多数类别学习的权重，同样可以解决类别不平衡问题。

对于参数的选择，需要再次进行多次实验，根据实验结果选出最佳的参数。

参考文献

[1] NEMADE V, FEGADE V. Machine Learning Techniques for Breast Cancer Prediction[J/OL]. Procedia Computer Science, 2023, 218: 1314-1320. https://www.sciencedirect.com/science/article/pii/S1877050923001102. DOI: https://doi.org/10.1016/j.procs.2023.01.110.

[2] UDDIN K M M, BISWAS N, RIKTA S T, et al. Machine learning-based diagnosis of breast cancer utilizing feature optimization technique[J/OL]. Computer Methods and Programs in Biomedicine Update, 2023, 3: 100098. https://www.sciencedirect.com/science/article/pii/S2666990023000071. DOI: https://doi.org/10.1016/j.cmpbup.2023.100098.

[3] MINNOOR M, BATHS V. Diagnosis of Breast Cancer Using Random Forests[J/OL]. Procedia Computer Science, 2023, 218: 429-437. https://www.sciencedirect.com/science/article/pii/S187705092300025X. DOI: https://doi.org/10.1016/j.procs.2023.01.025.

[4] RAETS C, EL AISATI C, DE RIDDER M, et al. An Evolutionary Random Forest to measure the Dworak tumor regression grade applied to colorectal cancer[J/OL]. Measurement, 2022, 205: 112131. https://www.sciencedirect.com/science/article/pii/S0263224122013276. DOI: https://doi.org/10.1016/j.measurement.2022.112131.

[5] XU Y, HU X, WEI J, et al. VF-CART: A communication-efficient vertical federated framework for the CART algorithm[J/OL]. Journal of King Saud University - Computer and Information Sciences, 2023, 35(1): 237-249. https://www.sciencedirect.com/science/article/pii/S1319157822004116. DOI: https://doi.org/10.1016/j.jksuci.2022.11.013.

[6] HOU J, DAI Z, LIU C, et al. Enhanced photoelectric properties for BiZn0.5Zr0.5O3 modified KNN-based lead-free ceramics[J/OL]. Journal of Alloys and Compounds, 2023, 960: 170639. https://www.sciencedirect.com/science/article/pii/S0925838823019424. DOI: https://doi.org/10.1016/j.jallcom.2023.170639.

附录 A 附录代码

```
 import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_breast_cancer  # 乳腺癌数据集 469, 30
from sklearn.model_selection import train_test_split    # 数据集划分
from sklearn.metrics import accuracy_score  # 准确率
from sklearn.metrics import f1_score        # F1值
from sklearn.metrics import roc_curve, auc  # ROC曲线和AUC
from sklearn.metrics import confusion_matrix,
   ConfusionMatrixDisplay    # 混淆矩阵

import math # 数学库
from sklearn.metrics import accuracy_score,precision_score,
   recall_score,f1_score,cohen_kappa_score # 评价函数相关
from sklearn.metrics import roc_curve,auc # 评价函数相关
from collections import Counter # 计数类别个数
from sklearn.model_selection import train_test_split # 训练集测试集划分
from sklearn.preprocessing import MinMaxScaler # 数据标准化

# 决策树桩
class DecisionStump:
    def __init__(self):
        # 基于划分阈值决定
        self.label = 1
        # 特征索引
        self.feature_index = None
        # 特征划分阈值
        self.threshold = None
        # 权重
        self.alpha = None
```

```
class AdaBoost:
    # 弱分类器集合
    def __init__(self, n_estimators=5):
        self.n_estimators = n_estimators
        self.estimators = []

    # AdaBoost算法
    def fit(self, X, y):
        # 获取样本数、维数
        m, n = X.shape
        # 初始化权重相等
        w = np.full(m, 1 / m)
        for _ in range(self.n_estimators):
            # 训练一个弱分类器：决策树桩
            estimator = DecisionStump()
            # 设定一个最小化误差率（初始设为∞）
            min_error = float('inf')
            # 遍历数据集特征，根据最小分类误差率选择最优特征作
               为分类依据
            for i in range(n):
                # 获取特征值
                values = np.expand_dims(X[:, i], axis=1)
                # 特征取值去重
                unique_values = np.unique(values)
                # 尝试将每一个特征值作为分类标准
                for threshold in unique_values:
                    label = 1
                    # 初始化所有预测值为1
                    pred = np.ones(np.shape(y))
                    # 小于分类阈值的预测值为-1
                    pred[X[:, i] < threshold] = -1
                    # 计算误差率
                    error = sum(w[y != pred])
                    # 误差率大于0.5，直接翻转正负预测
                    if error > 0.5:
                        error = 1 - error
```

```
                        label = -1
                    # 保存最小误差率相关参数
                    if error < min_error:
                        estimator.label = label
                        estimator.threshold = threshold
                        estimator.feature_index = i
                        min_error = error
            # 计算基分类器的权重
            estimator.alpha = 0.5 * np.log((1.0 - min_error) /
                max(min_error, 1e-12))
            # 初始化所有的预测值为 1
            preds = np.ones(np.shape(y))
            # 获取所有小于阈值的负类索引
            negative_idx = (estimator.label * X[:, estimator.
               feature_index] < estimator.label * estimator.
               threshold)
            # 将负类设置为 -1
            preds[negative_idx] = -1
            # 更新样本权重
            w *= np.exp(-estimator.alpha * y * preds)
            w /= np.sum(w)
            # 保存该弱分类器
            self.estimators.append(estimator)

    # 随机森林算法
    def fit_origin_RF(self, X, y):
        # 获取样本数、维数
        m, n = X.shape
        # 初始化权重相等
        w = np.full(m, 1 / m)
        for _ in range(self.n_estimators):
            # 训练一个弱分类器: 决策树桩
            estimator = DecisionStump()
            # 设定一个最小化误差率（初始设为∞）
            min_error = float('inf')
            k = 5
            rand = np.random.randint(0, n, k)
```

```
        # 遍历数据集特征，根据最小分类误差率选择最优特征作为分类依据
        for i in rand:
            # 获取特征值
            values = np.expand_dims(X[:, i], axis=1)
            # 特征取值去重
            unique_values = np.unique(values)
            # 尝试将每一个特征值作为分类阈值
            for threshold in unique_values:
                label = 1
                # 初始化所有预测值为1
                pred = np.ones(np.shape(y))
                # 小于分类阈值的预测值为-1
                pred[X[:, i] < threshold] = -1
                # 计算误差率
                error = sum(w[y != pred])
                # 误差率大于0.5，直接翻转正负预测
                if error > 0.5:
                    error = 1 - error
                    label = -1
                # 保存最小误差率相关参数
                if error < min_error:
                    estimator.label = label
                    estimator.threshold = threshold
                    estimator.feature_index = i
                    min_error = error
        # 计算基分类器的权重
        estimator.alpha = 0.5 * np.log((1.0 - min_error) / max(min_error, 1e-12))
        # 初始化所有的预测值为 1
        preds = np.ones(np.shape(y))
        # 获取所有小于阈值的负类索引
        negative_idx = (estimator.label * X[:, estimator.feature_index] < estimator.label * estimator.threshold)
        # 将负类设置为 -1
        preds[negative_idx] = -1
```

```
            # 更新样本权重
            #w *= np.exp(-estimator.alpha * y * preds)
            #w /= np.sum(w)
            # 保存该弱分类器
            self.estimators.append(estimator)

    # 改进后随机森林算法 with weight-sample
    def fit_RF(self, X, y):
        # 获取样本数、维数
        m, n = X.shape
        # 初始化权重相等
        w = np.full(m, 1 / m)
        for _ in range(self.n_estimators):
            # 训练一个弱分类器: 决策树桩
            estimator = DecisionStump()
            # 设定一个最小化误差率（初始设为∞）
            min_error = float('inf')
            k = 5
            rand = np.random.randint(0, n, k)
            # 遍历数据集特征, 根据最小分类误差率选择最优特征作
                为分类依据
            for i in rand:
                # 获取特征值
                values = np.expand_dims(X[:, i], axis=1)
                # 特征取值去重
                unique_values = np.unique(values)
                # 尝试将每一个特征值作为分类阈值
                for threshold in unique_values:
                    label = 1
                    # 初始化所有预测值为1
                    pred = np.ones(np.shape(y))
                    # 小于分类阈值的预测值为-1
                    pred[X[:, i] < threshold] = -1
                    # 计算误差率
                    error = sum(w[y != pred])
                    # 误差率大于0.5, 直接翻转正负预测
                    if error > 0.5:
```

```
                        error = 1 - error
                        label = -1
                    # 保存最小误差率相关参数
                    if error < min_error:
                        estimator.label = label
                        estimator.threshold = threshold
                        estimator.feature_index = i
                        min_error = error
            # 计算基分类器的权重
            estimator.alpha = 0.5 * np.log((1.0 - min_error) /
                max(min_error, 1e-12))
            # 初始化所有的预测值为 1
            preds = np.ones(np.shape(y))
            # 获取所有小于阈值的负类索引
            negative_idx = (estimator.label * X[:, estimator.
               feature_index] < estimator.label * estimator.
               threshold)
            # 将负类设置为 -1
            preds[negative_idx] = -1
            # 更新样本权重
            w *= np.exp(-estimator.alpha * y * preds)
            w /= np.sum(w)
            # 保存该弱分类器
            self.estimators.append(estimator)

    def predict(self, X):
        m = len(X)
        y_pred = np.zeros((m, 1))
        # 计算每个弱分类器的预测值
        for estimator in self.estimators:
            # 初始化所有预测值为1
            predictions = np.ones(np.shape(y_pred))
            # 获取所有小于阈值的负类索引
            negative_idx = (estimator.label * X[:, estimator.
               feature_index] < estimator.label * estimator.
               threshold)
            # 将负类设为 -1
```

```
            predictions[negative_idx] = -1
            # 对每个弱分类器的预测结果乘以alpha加权后进行累加
            y_pred += estimator.alpha * predictions
        y_score = y_pred
        # 返回最终预测结果
        y_pred = np.sign(y_pred).flatten()
        return y_pred, y_score


def calculate_recall_fpr_tpr(y_true, y_pred):
    y_true = y_true.flatten()
    y_pred = y_pred.flatten()
    Tp = 0
    Fp = 0
    Tn = 0
    Fn = 0
    for label, pred in zip(y_true, y_pred):
        if (label == -1) and (pred == -1):
            Tp = Tp + 1
        elif (label == 1) and (pred == -1):
            Fp = Fp + 1
        elif (label == 1) and (pred == 1):
            Tn = Tn + 1
        elif (label == -1) and (pred == 1):
            Fn = Fn + 1
        else:
            print('Labels error!')
            return -1
    recall = Tp / (Tp + Fn)
    fpr = Fp / (Fp + Tn)
    tpr = Tp / (Tp + Fn)
    print("Recall:", recall)
    print("FPR:", fpr)
    print("TPR:", tpr)


def ROC(y_true, y_score):
```

```
    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr)
    print("AUC:", roc_auc)
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC
       curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--
       ')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()


def conf_matrix(y_test, y_pred, t):
    cm = confusion_matrix(y_test, y_pred)  # 混淆矩阵
    print("Confusion matrix of Label is \n", cm)
    # confusion_matrix(混淆矩阵), display_labels(标签名称列表)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
       display_labels=['0', '1'])
    # 画出混淆矩阵
    disp.plot()
    plt.title(f"Confusion matrix_{t}")
    # 保存
    # plt.savefig(f"Confusion_matrix_RF{t}")
    # 显示
    plt.show()


import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 假设你的数据存储在一个名为 "data" 的 DataFrame 中
data = pd.read_csv("cancer.csv")
```

```
# 计算特征之间的相关系数矩阵
corr_matrix = data.corr()

# 绘制特征图
plt.figure(figsize=(25, 20))
sns.heatmap(corr_matrix, annot=True, cmap="YlGnBu")
plt.title("Correlation Heatmap of Cancer Data")

# 调整 x 轴和 y 轴的刻度文本角度，以防止重叠
plt.xticks(rotation=45)
plt.yticks(rotation=0)

# 显示图形
plt.show()


# 进行不同数量弱分类器的集成学习
for t in [1, 5, 15]:
    print("————————————————————AdaBoost
        ————————————————————————————")
    print("弱分类器数量：", t)
    # 创建 Adaboost 模型实例
    clf = AdaBoost(n_estimators=t)
    # 模型拟合
    clf.fit(X_train, y_train)
    # 模型预测
    y_pred, y_score = clf.predict(X_test)
    # 计算模型预测的分类准确率
    accuracy = accuracy_score(y_test, y_pred)
    # 计算模型预测的recall, FPR, TPR
    calculate_recall_fpr_tpr(y_test, y_pred)
    # 计算模型预测的F1值
    F1 = f1_score(y_test, y_pred, labels=None, pos_label=1,
                  average='binary', sample_weight=None,
                     zero_division="warn")
    print("准确率:", accuracy)
```

```
    print("F1:", F1)
    ROC(y_test, y_score)
    conf_matrix(y_test, y_pred, t)

def standard_scaler(x):
    '''对数据进行归一化'''
    mean = np.mean(x, axis=0)
    std = np.std(x, axis=0)
    # print("矩阵的初值为:", x)
    # print("均值:", mean, " 标准差:", std)
    trans_data = x - mean
    trans_data = trans_data/std
    # print("归一化之后的矩阵为:", trans_data)
    return trans_data

def _PCA(attributes, compents):


    # 对 X 进行去中心化
    attributes = standard_scaler(attributes)
    # 计算样本的协方差矩阵 X*X.T
    covx = np.dot(attributes, attributes.T)
    # 求矩阵的特征值和特征向量
    lamda, Vv = np.linalg.eigh(covx)
    index = np.argsort(-lamda)[:compents]
    diag_lamda = np.sqrt(np.diag(-np.sort(-lamda)[:compents]))
    # 取出对应特征向量
    V_selected = Vv[:, index]
    Zz = V_selected.dot(diag_lamda)
    #print(np.sum(V_selected))
    print(np.shape(Zz))
    return Zz


def dist(a, b): # 距离计算
    tmp = 0
    for i, x in enumerate(a):
```

```
        tmp += (x - b[i]) * (x - b[i])
    return np.sqrt(tmp)

def knn_predict(sample, k):
    # print(sample)
    dists = []
    for i, a in enumerate(X_train):
        dists.append((dist(sample, a), Y_train[i])) # 计算与每
            个样本的距离
    dists.sort() # 排序
    dist_k_min = dists[:k] # 选择最小的k个样本
    tmp = 0
    for a in dist_k_min: # 对于k个邻居样本进行投票
        if a[1] == 0:
            tmp += 1
        else:
            tmp -= 1
    return tmp < 0

def knn_predict1(sample, k=9):
    # print(sample)
    dists = []
    for i, a in enumerate(X_train):
        dists.append((dist(sample, a), Y_train[i]))
    dists.sort()
    # print(dists)
    dist_k_min = dists[:k]
    tmp = 0
    for a in dist_k_min:
        if a[1] == 0:
            tmp += 1
        else:
            tmp -= 1
    # print(tmp)
    # print(int(tmp < 0))
    return float(-tmp / 9)
```

```
def train(x, y):
    global X_train
    global Y_train
    X_train = x
    Y_train = y


def knn_init():
    data = pd.read_csv('diabetes.csv') # 读取数据集
    new_data = data.drop(['Outcome'], axis=1) # 分离feature和
       label
    scale = MinMaxScaler().fit(new_data)  # 标准化规则
    biao_data = scale.transform(new_data)  # 应用规则

    X_train, X_test, y_train, y_test = train_test_split(
       biao_data, data['Outcome'], test_size=0.2, random_state
       =123)
    # 随机划分测试集训练集, 比例为0.2
    y_test = y_test.values
    y_train = y_train.values

    train(X_train, y_train)

    y_pred = []
    for i, sample in enumerate(X_test):
        y_pred.append(float(knn_predict(sample, 9)))
        # print(y_test[i])
        # print()

    fpr, tpr, threshold = roc_curve(y_test, y_pred)
    print('数据的AUC为:', auc(fpr, tpr))
    print('数据的准确率为: ', accuracy_score(y_test, y_pred))
    print('数据的精确率为: ', precision_score(y_test, y_pred))
    print('数据的召回率为: ', recall_score(y_test, y_pred))
    print('数据的F1值为: ', f1_score(y_test, y_pred))
    print('数据的Cohen' s Kappa系数为: ', cohen_kappa_score(
       y_test, y_pred))
```

```
    print('Counter:', Counter(y_pred))

    roc_auc = auc(fpr, tpr)  # 计算AUC（曲线下面积）
    print("AUC: %.2f" % roc_auc)  # 打印AUC值
    conf_matrix(y_test, y_pred, 0)
    # 绘制ROC曲线
    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' %
       roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')  # 绘制对角线
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve')
    plt.legend(loc="lower right")
    plt.show()


num = 4
pca = PCA(num)


def pca_knn_init():
    data = pd.read_csv('diabetes.csv') # 读取数据集
    new_data = data.drop(['Outcome'], axis=1) # 分离feature和
       label
    scale = MinMaxScaler().fit(new_data)  # 标准化规则
    biao_data = scale.transform(new_data)  # 应用规则
    biao_data = pca.fit_transform(biao_data)
    X_train, X_test, y_train, y_test = train_test_split(
       biao_data, data['Outcome'], test_size=0.2, random_state
       =123)
    # 随机划分测试集训练集，比例为0.2
    y_test = y_test.values
    y_train = y_train.values
```

```
    train(X_train, y_train)

    y_pred = []
    for i, sample in enumerate(X_test):
        y_pred.append(float(knn_predict(sample, 9)))
        # print(y_test[i])
        # print()

    fpr, tpr, threshold = roc_curve(y_test, y_pred)
    print('数据的AUC为:', auc(fpr, tpr))
    print('数据的准确率为:', accuracy_score(y_test, y_pred))
    print('数据的精确率为:', precision_score(y_test, y_pred))
    print('数据的召回率为:', recall_score(y_test, y_pred))
    print('数据的F1值为:', f1_score(y_test, y_pred))
    print('数据的Cohen's Kappa系数为:', cohen_kappa_score(
       y_test, y_pred))
    print('Counter:', Counter(y_pred))

    roc_auc = auc(fpr, tpr)  # 计算AUC(曲线下面积)
    print("AUC: %.2f" % roc_auc)  # 打印AUC值
    conf_matrix(y_test, y_pred, 0)
    # 绘制ROC曲线
    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' %
       roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')  # 绘制对角线
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve')
    plt.legend(loc="lower right")
    plt.show()

pca_knn_init()
```

```
def bayes_predict(sample):
    tmp = [0, 0]
    for i in range(class_num): # 枚举每个类别
        tmp[i] = np.log(cnt[i])
        for j in range(feature_len): # 枚举各个特征属性
            # 根据该特征属性的均值方差，做出高斯分布概率，代入
               得到后验概率
            tmp[i] += np.log(1 / math.sqrt(2 * math.pi * va[i
               ][j])) - (sample[j] - me[i][j]) * (sample[j] -
               me[i][j]) / (2 * va[i][j])

    # 选择后验概率最大的类别作为预测类别
    if tmp[0] > tmp[1]:
        return 0
    else:
        return 1


def bayes_predict1(sample):
    tmp = [0, 0]
    for i in range(class_num):
        tmp[i] = np.log(cnt[i])
        for j in range(feature_len):
            tmp[i] += np.log(1 / math.sqrt(2 * math.pi * va[i
               ][j])) - (sample[j] - me[i][j]) * (
                        sample[j] - me[i][j]) / (2 * va[i][j])

    # print(tmp[0], tmp[1])
    # print((int)(tmp[0] <= tmp[1]))
    return tmp[1] - tmp[0]

def bayes_train(data, y):
    global cnt
    cnt = [0, 0]
    o = []
    for i in range(class_num): # 枚举类别
        tmp = []
```

```
        for j in range(feature_len): # 枚举特征属性
            tmp1 = []
            for k, sample in enumerate(data):
                if y[k] == i:
                    tmp1.append(sample[j]) # 存储该类别下该特
                        征的值
                    cnt[i] += 1 # 类别概率计算
            tmp.append(tmp1)
        o.append(tmp)

    for i in range(class_num):
        cnt[i] = float(cnt[i]) # 计算类别先验概率

    global me # 特定类别下特征属性的均值
    global va # 特定类别下特征属性的方差
    me = np.zeros((class_num, feature_len), dtype=float)
    va = np.zeros((class_num, feature_len), dtype=float)

    for i in range(class_num):
        for j in range(feature_len):
            # 利用numpy计算均值方差
            me[i][j] = np.mean(np.array(o[i][j]))
            va[i][j] = np.var(np.array(o[i][j]))


def bayes_init():
    data = pd.read_csv('diabetes.csv')



    new_data = data.drop(['Outcome'], axis=1)
    scale = MinMaxScaler().fit(new_data)  ## 训练规则
    biao_data = scale.transform(new_data)  ## 应用规则

    X_train, X_test, y_train, y_test = train_test_split(
        biao_data, data['Outcome'], test_size=0.2, random_state
        =123)
```

```
y_test = y_test.values
y_train = y_train.values

bayes_train(X_train, y_train)

y_pred = []
for i, sample in enumerate(X_test):
    y_pred.append(float(bayes_predict(sample)))
    # print(y_test[i])

fpr, tpr, threshold = roc_curve(y_test, y_pred)# 计算ROC曲
   线的假正率（FPR），真正率（TPR）和阈值
print('数据的AUC为:', auc(fpr, tpr))
print('数据的准确率为：', accuracy_score(y_test, y_pred))
print('数据的精确率为：', precision_score(y_test, y_pred))
print('数据的召回率为：', recall_score(y_test, y_pred))
print('数据的F1值为：', f1_score(y_test, y_pred))
print('数据的Cohen' s Kappa系数为：', cohen_kappa_score(
   y_test, y_pred))
print('Counter:', Counter(y_pred))

roc_auc = auc(fpr, tpr)  # 计算AUC（曲线下面积）
print("AUC: %.2f" % roc_auc)  # 打印AUC值
conf_matrix(y_test, y_pred, 0)
# 绘制ROC曲线
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' %
   roc_auc)
plt.plot([0, 1], [0, 1], 'k--')  # 绘制对角线
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.legend(loc="lower right")
plt.show()
```

```
def train(X_train, y_train):
    global a
    a = 0.05
    global b
    b = 0.3
    epoch = 8 # 训练周期
    lr = 0.005 # 学习率
    global w
    w = 0.5
    X = []
    Y = []
    for i in range(epoch): # 梯度下降法
        ax = 0
        by = 0
        cnt = 0
        for j, sample in enumerate(X_train):
            x = bayes_predict1(sample) # bayes预测后验概率之差
                作为输入
            y = knn_predict1(sample) # knn邻居投票票数作为输入
            z = int(a * x + b * y > 0)
            if(y_train[j] == z):
                cnt += 1
            # 对参数求梯度
            if(y_train[j] == 1):
                ax += (a * x + b * y - y_train[j] + w) * x * 2
                by += (a * x + b * y - y_train[j] + w) * y * 2
            else :
                ax += (a * x + b * y - y_train[j] + w) * x
                by += (a * x + b * y - y_train[j] + w) * y
        ax /= X_train.shape[0]
        by /= X_train.shape[0]
        cnt /= X_train.shape[0]
        X.append(i)
        Y.append(cnt)
        #print(ax, by)
        # 对参数进行梯度更新
```

```
            a -= lr * ax
            b -= lr * by

        plt.plot(X, Y)
        plt.title('Accuracy varies with training cycle')
        plt.show()

    # print(a, b)
        return (a, b)

def predict(sample):
        x = bayes_predict1(sample)
        y = knn_predict1(sample)
    # print(x, y, a * x + b * y + w)
        return a * x + b * y > 0



data = pd.read_csv('diabetes.csv')
new_data = data.drop(['Outcome'], axis=1)
scale = MinMaxScaler().fit(new_data)  ## 训练规则
biao_data = scale.transform(new_data)  ## 应用规则

X_train, X_test, y_train, y_test = train_test_split(biao_data,
    data['Outcome'], test_size=0.2, random_state=123)
y_test = y_test.values
y_train = y_train.values

train(X_train, y_train)

y_pred = []
for i, sample in enumerate(X_test):
    y_pred.append(float(predict(sample)))
    #print(y_test[i])

fpr, tpr, threshold = roc_curve(y_test, y_pred)
print('数据的AUC为:', auc(fpr, tpr))
```

```
print('数据的准确率为：', accuracy_score(y_test, y_pred))
print('数据的精确率为：', precision_score(y_test, y_pred))
print('数据的召回率为：', recall_score(y_test, y_pred))
print('数据的F1值为：', f1_score(y_test, y_pred))
print('数据的Cohen' s Kappa系数为：', cohen_kappa_score(y_test
    , y_pred))
print('Counter:', Counter(y_pred))
```