****

《模式识别与机器学习》

课程设计报告

|  |  |
| --- | --- |
| 题 目 | 基于bayes和knn的集成学习在pima糖尿病数据集上的表现分析 |
| 学生姓名 | 蒋文博 |
| 学 号 | 8207211211 |
| 专业班级 | 智能2105 |
| 指导老师 | 肖晓明 |
| 学 院 | 自动化院 |

本科生院制

2023年06月

**一、项目简介**

项目计划完成一个数据分类任务，数据集采用pima-indians-diabetes，使用朴素贝叶斯、KNN、以及两者集成算法完成糖尿病预测任务。在任务过程中，比较朴素贝叶斯分类器与KNN分类器的分类效果，且对比基本分类器和集成分类器的预测情况。

**二、数据集简介**

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，存在类别不平衡的问题，阳性：阴性为1:3。



**图 1 数据集简介**

**三、朴素贝叶斯分类器实现**

**1、贝叶斯分类器原理**

贝叶斯决策论是概率框架下实施决策的基本方法。对分类任务来说，在所有相关概率都已知的理想情形下，贝叶斯决策论考虑如何基于这些概率和误判损失来选择最优的类别标记。

先验概率是指在没有考虑任何特征信息的情况下，某个事件或类别发生的概率。在贝叶斯分类器中，先验概率表示每个类别在整个数据集中出现的概率。它是在观察到任何特征信息之前，对类别的初始估计。

例如，假设我们有一个二元分类问题，其中类别A和类别B是两个可能的类别。如果数据集中类别A的样本数量为100，类别B的样本数量为200，那么类别A和类别B的先验概率分别为P(A) = 100 / (100 + 200) = 1/3 和 P(B) = 200 / (100 + 200) = 2/3。

后验概率是指在考虑了特征信息后，某个事件或类别发生的概率。在贝叶斯分类器中，后验概率表示给定观察到的特征条件下，某个样本属于某个类别的概率。

后验概率可以通过贝叶斯定理计算得到，其表达式为：

P(A|X) = (P(X|A) \* P(A)) / P(X)

其中，P(A|X) 表示在给定特征 X 的条件下类别 A 的后验概率，P(X|A) 是在类别 A 的条件下特征 X 的概率，P(A) 是先验概率，P(X) 是特征 X 的边际概率（在所有可能类别下的特征 X 的概率之和）。

贝叶斯分类器通过计算不同类别的后验概率，并选择后验概率最大的类别作为样本的预测类别。

基于贝叶斯公式来估计后验概率的主要困难在于：类条件概率是所有属性上的联合概率，难以从优先的训练样本直接估计而得。为避开这个障碍，朴素贝叶斯分类器采用了“属性条件独立性假设”：对一直类别，假设所有属性相互独立，每个属性独立地对分类结果发生影响。

**2、高斯朴素贝叶斯分类器**

在贝叶斯分类器中处理连续属性情形，通常采用概率密度函数来建模属性的条件概率，使用高斯分布时被称为高斯朴素贝叶斯分类器。

具体而言对于每个类别和每个连续属性，使用概率密度函数（通常假设服从高斯分布）来估计在给定类别下该属性的条件概率。即对于每个类别，计算该类别下每个属性的均值和标准差，然后使用高斯分布的概率密度函数来计算给定属性值的条件概率。

需要注意的是，为了避免概率密度函数为零的情况，通常使用平滑技术来调整估计的概率密度函数。例如，可以使用拉普拉斯平滑或高斯核平滑等方法。

在本次实验中，由于数据集并无缺失，无需使用平滑，仅使用对数似然完成高斯朴素贝叶斯分类器。

3、**贝叶斯分类器具体实现**

首先加载必要库，包括数学库、评价函数、数据库读取等：

1. **import** pandas as pd # pandas用来读取csv数据
2. **import** numpy as np # numpy库进行数学操作
3. **import** math # 数学库
4. **from** sklearn.metrics **import** accuracy\_score,precision\_score, recall\_score,f1\_score,cohen\_kappa\_score # 评价函数相关
5. **from** sklearn.metrics **import** roc\_curve,auc # 评价函数相关
6. **from** collections **import** Counter # 计数类别个数
7. **from** sklearn.model\_selection **import** train\_test\_split # 训练集测试集划分
8. **from** sklearn.preprocessing **import** MinMaxScaler # 数据标准化

对数据集进行读取，标准化处理后，划分测试集与训练集：

1. data = pd.read\_csv('diabetes.csv') # 读取数据集
2. new\_data = data.drop(['Outcome'], axis=1) # 分离feature和label
3. scale = MinMaxScaler().fit(new\_data)  # 标准化规则
4. biao\_data = scale.transform(new\_data)  # 应用规则
6. X\_train, X\_test, y\_train, y\_test = train\_test\_split(biao\_data, data['Outcome'], test\_size=0.2, random\_state=123)
7. # 随机划分测试集训练集，比例为0.2
8. y\_test = y\_test.values
9. y\_train = y\_train.values

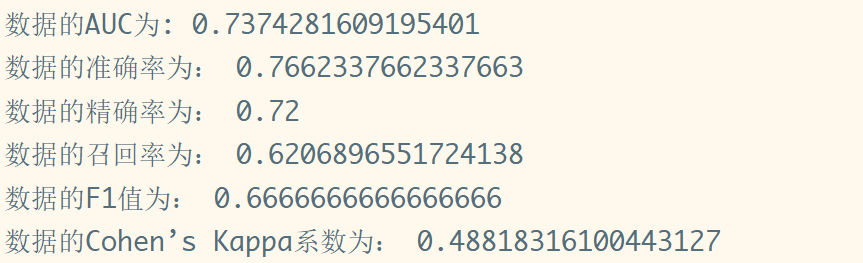
训练过程，计算每个类别的先验概率以及该类别下每个属性的条件概率：

1. **def** train(data, y):
2. **global** cnt
3. cnt = [0, 0]
4. o = []
5. **for** i **in** range(class\_num): # 枚举类别
6. tmp = []
7. **for** j **in** range(feature\_len): # 枚举特征属性
8. tmp1 = []
9. **for** k, sample **in** enumerate(data):
10. **if** y[k] == i:
11. tmp1.append(sample[j]) # 存储该类别下该特征的值
12. cnt[i] += 1 # 类别概率计算
13. tmp.append(tmp1)
14. o.append(tmp)
16. **for** i **in** range(class\_num):
17. cnt[i] = float(cnt[i]) # 计算类别先验概率
19. **global** me # 特定类别下特征属性的均值
20. **global** va # 特定类别下特征属性的方差
21. me = np.zeros((class\_num, feature\_len), dtype=float)
22. va = np.zeros((class\_num, feature\_len), dtype=float)
24. **for** i **in** range(class\_num):
25. **for** j **in** range(feature\_len):
26. # 利用numpy计算均值方差
27. me[i][j] = np.mean(np.array(o[i][j]))
28. va[i][j] = np.var(np.array(o[i][j]))

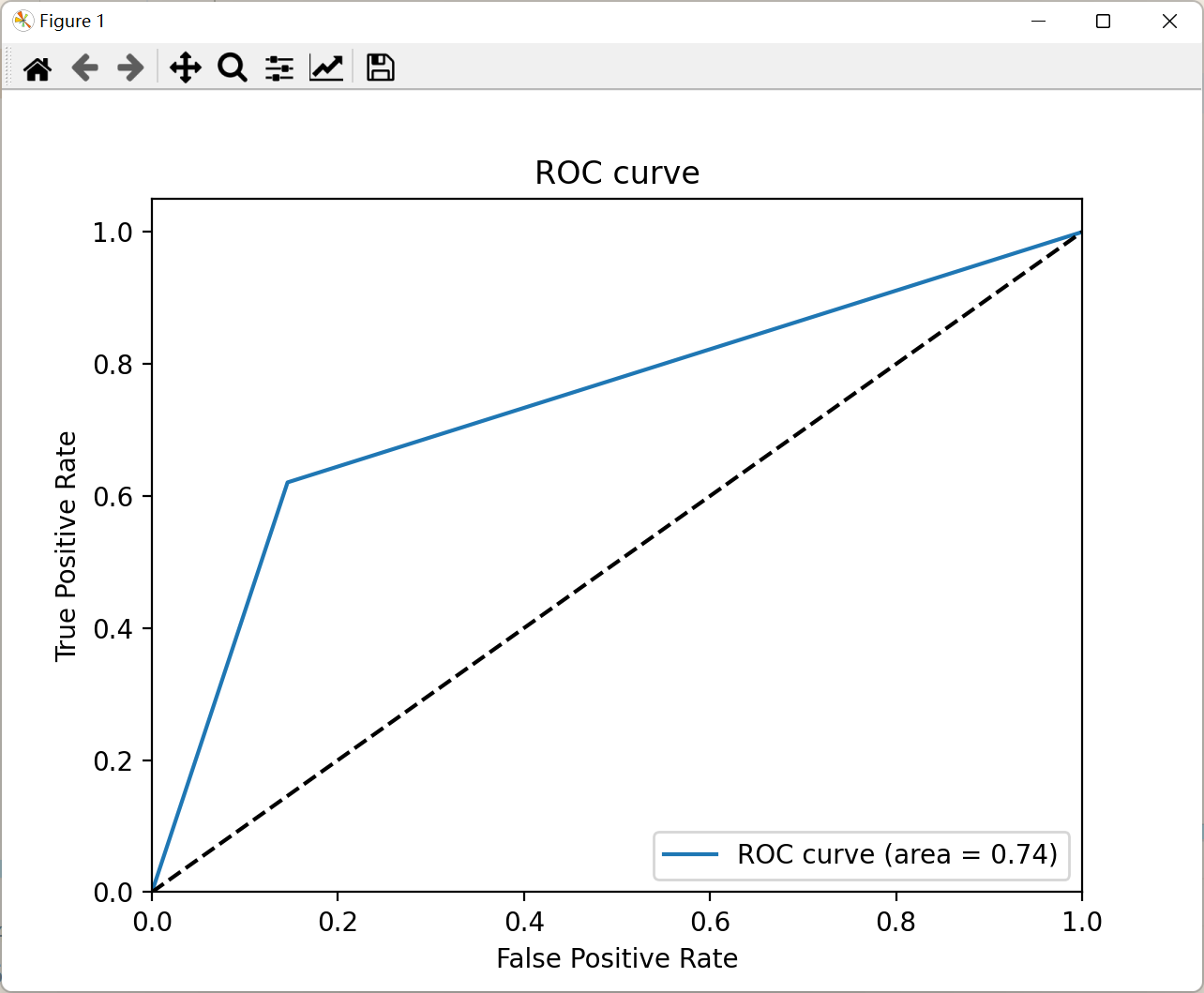
预测过程，根据计算得到的后验概率，选择后验概率最大的类别作为样本的预测类别：

1. **def** predict(sample):
2. tmp = [0, 0]
3. **for** i **in** range(class\_num): # 枚举每个类别
4. tmp[i] = np.log(cnt[i])
5. **for** j **in** range(feature\_len): # 枚举各个特征属性
6. # 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
7. 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])
9. # 选择后验概率最大的类别作为预测类别
10. **if** tmp[0] > tmp[1]:
11. **return** 0
12. **else**:
13. **return** 1

3、**运行效果展示**



**图 2 贝叶斯分类器在测试集上效果展现**



**图 3 贝叶斯分类器在测试集上效果展现**

**四、KNN分类器实现**

**1、KNN分类器原理**

K近邻学习是一种常用的监督学习方法，其工作机制简单：给定测试样本，基于某种距离度量找出训练集中与其最靠近的k个训练样本，然后基于这k个邻居的信息进行预测。

对于给定的未标记样本，KNN算法首先计算该样本与训练集中每个已标记样本之间的距离。常见的距离度量包括欧氏距离、曼哈顿距离、闵可夫斯基距离等。距离的选择取决于具体问题和数据特征。

KNN选择与未标记样本距离最近的K个训练集样本作为其最近邻。K是一个预先指定的参数，可以根据实际问题和数据集的特点进行选择。一般来说，较小的K值会产生较低的偏差但较高的方差，而较大的K值则会产生较高的偏差但较低的方差。

KNN根据K个最近邻的类别标签进行投票决策。一般来说，KNN采用多数表决的方式确定未标记样本的类别。即，选择K个最近邻中出现次数最多的类别作为未标记样本的预测类别。

值得一提的是，KNN分类器对于距离的度量和K值的选择非常敏感。选择合适的距离度量和K值对分类性能至关重要。此外，KNN分类器对于数据集的规模较大时，计算距离的复杂度较高，可能会导致计算开销较大。

**2、K值选择与k-fold交叉验证**

KNN算法中的K值选择对于分类性能至关重要。较小的K值会产生更复杂的决策边界，容易受到噪声的影响，而较大的K值则会产生较平滑的决策边界，可能会忽略掉一些细节。为了选择最优的K值，可以使用K-fold交叉验证方法。

K-fold交叉验证是一种常用的模型评估方法，可用于评估模型的性能并选择最佳的参数。它的基本原理是将数据集分成K个大小相似的子集（称为折叠），然后使用其中K-1个子集作为训练数据，剩余的一个子集作为验证数据，重复K次，每次使用不同的子集作为验证集。最后，将K次的评估结果平均得到最终的性能指标。

具体而言，使用K-fold交叉验证来选择K值的一般步骤如下：

将数据集分成K个大小相似的折叠。

对于每个K值（例如从1到N，N是指定的最大K值），进行以下步骤：

对于每个折叠，使用其他K-1个折叠作为训练数据，剩余的一个折叠作为验证数据。在训练数据上运行KNN算法，并使用验证数据计算模型的性能指标，例如准确率、F1分数等。将K次的性能指标平均，得到该K值下的平均性能。选择平均性能最好的K值作为最优的K值。

通过K-fold交叉验证，可以评估不同K值下的模型性能，并选择最佳的K值。常见的选择标准包括准确率、F1分数、召回率等，具体根据问题的需求来决定。

需要注意的是，K值的选择也要考虑到数据集的大小。当数据集较小时，选择较小的K值可以避免过拟合；而当数据集较大时，选择较大的K值可以减少噪声的影响。

对于本次实验，采用K = 9是比较优秀的参数选择。

3、**KNN分类器具体实现**

首先加载必要库，包括数学库、评价函数、数据库读取等：

1. **import** pandas as pd # pandas用来读取csv数据
2. **import** numpy as np # numpy库进行数学操作
3. **import** math # 数学库
4. **from** sklearn.metrics **import** accuracy\_score,precision\_score, recall\_score,f1\_score,cohen\_kappa\_score # 评价函数相关
5. **from** sklearn.metrics **import** roc\_curve,auc # 评价函数相关
6. **from** collections **import** Counter # 计数类别个数
7. **from** sklearn.model\_selection **import** train\_test\_split # 训练集测试集划分
8. **from** sklearn.preprocessing **import** MinMaxScaler # 数据标准化

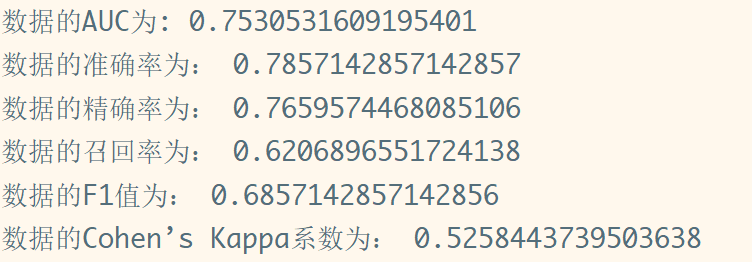
对数据集进行读取，标准化处理后，划分测试集与训练集：

1. data = pd.read\_csv('diabetes.csv') # 读取数据集
2. new\_data = data.drop(['Outcome'], axis=1) # 分离feature和label
3. scale = MinMaxScaler().fit(new\_data)  # 标准化规则
4. biao\_data = scale.transform(new\_data)  # 应用规则
6. X\_train, X\_test, y\_train, y\_test = train\_test\_split(biao\_data, data['Outcome'], test\_size=0.2, random\_state=123)
7. # 随机划分测试集训练集，比例为0.2
8. y\_test = y\_test.values
9. y\_train = y\_train.values

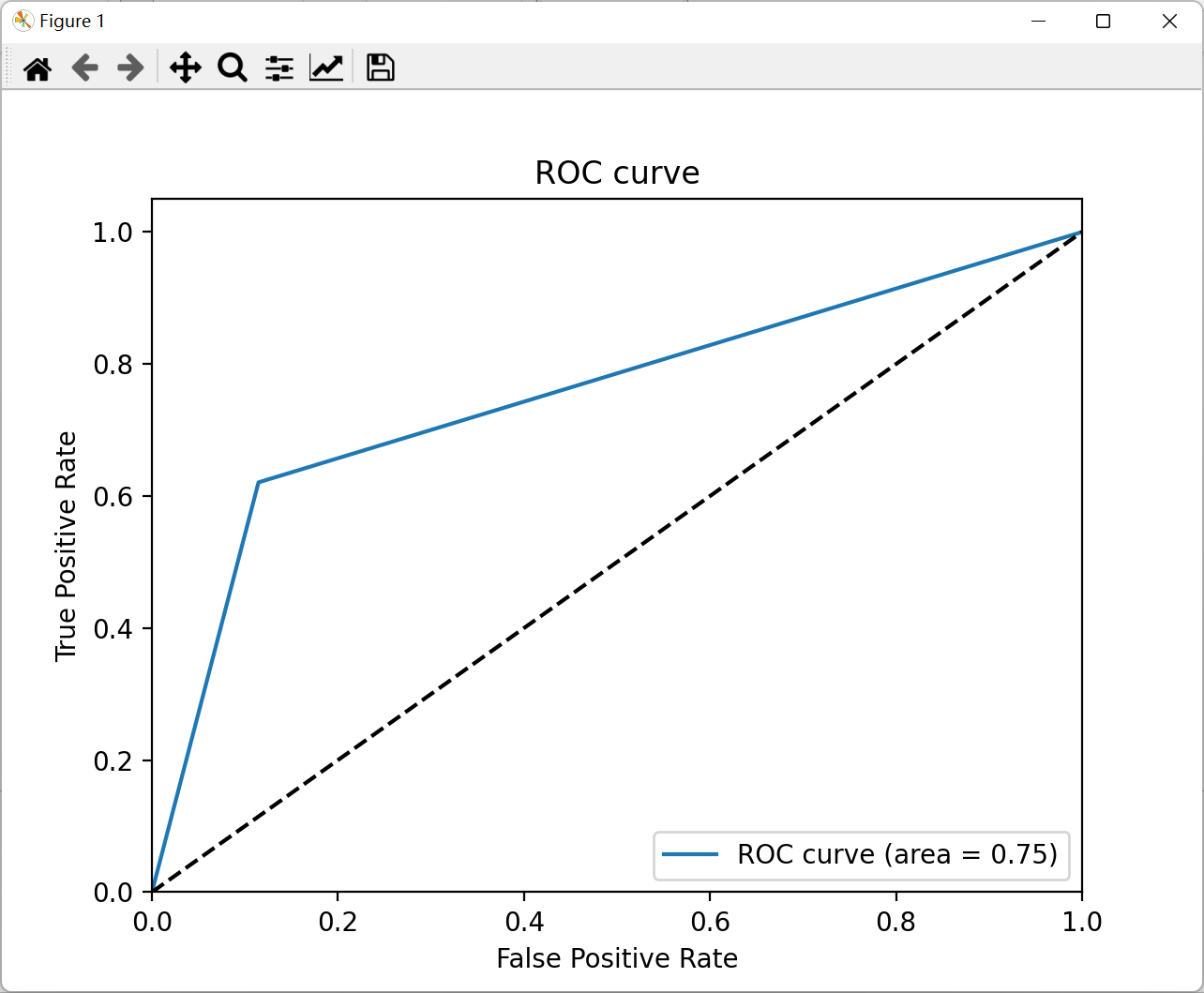
预测过程，根据距离公式计算和每个训练集样本的距离，选择距离最小的k个进行投票预测：

1. **def** dist(a, b): # 距离计算
2. tmp = 0
3. **for** i, x **in** enumerate(a):
4. tmp += (x - b[i]) \* (x - b[i])
5. **return** np.sqrt(tmp)
7. **def** predict(sample, k):
8. # print(sample)
9. dists = []
10. **for** i, a **in** enumerate(X\_train):
11. dists.append((dist(sample, a), Y\_train[i])) # 计算与每个样本的距离
12. dists.sort() # 排序
13. dist\_k\_min = dists[:k] # 选择最小的k个样本
14. tmp = 0
15. **for** a **in** dist\_k\_min: # 对于k个邻居样本进行投票
16. **if** a[1] == 0:
17. tmp += 1
18. **else**:
19. tmp -= 1
20. **return** tmp < 0

3、**运行效果展示**



**图 4 knn分类器在测试集上效果展现**



**图 5 knn分类器在测试集上效果展现**

**四、stacking集成学习**

**1、stacking方法**

Stacking是一种集成学习方法，旨在将多个基本分类器的预测结果结合起来，以获得更好的分类性能。它通过层级结构组织多个分类器，其中第一层的分类器用于生成预测结果，而第二层的分类器（元分类器）则将第一层的预测结果作为输入，并生成最终的集成预测。

准备数据集：首先，需要准备一个有标签的训练数据集，其中包含特征和对应的类别标签。

创建多个基本分类器：选择多个不同的基本分类器作为第一层分类器。这些分类器可以是不同的算法，也可以是同一算法的不同设置或不同子集的训练数据。

划分训练集和验证集：将训练数据集分成两个部分，一部分用于训练第一层的基本分类器，另一部分作为验证集用于训练元分类器。

训练第一层分类器：使用训练数据集训练每个基本分类器。每个基本分类器根据特征生成预测结果，这些预测结果构成了第二层分类器的输入。

生成第二层训练集：使用训练数据集中的每个样本，将第一层分类器的预测结果作为特征，将对应的真实标签作为目标，构建一个新的训练数据集作为元分类器的输入。

训练元分类器：使用第二层训练集训练元分类器，可以是任意分类算法，如逻辑回归、决策树、支持向量机等。

进行预测：对于新的未标记样本，首先使用第一层分类器生成预测结果，然后将这些预测结果作为输入，通过元分类器生成最终的集成预测。

Stacking集成学习是一种将多个基本分类器的预测结果结合起来的方法，通过层级结构和元分类器的组合，实现更强大的分类性能。在本次实验中，使用贝叶斯分类器、knn分类器作为基本分类器，在最后使用线性回归组合两者，以得到更好的分类效果。

**2、具体实现**

首先加载必要库，包括数学库、评价函数、数据库读取等：

1. **import** pandas as pd # pandas用来读取csv数据
2. **import** numpy as np # numpy库进行数学操作
3. **import** math # 数学库
4. **from** sklearn.metrics **import** accuracy\_score,precision\_score, recall\_score,f1\_score,cohen\_kappa\_score # 评价函数相关
5. **from** sklearn.metrics **import** roc\_curve,auc # 评价函数相关
6. **from** collections **import** Counter # 计数类别个数
7. **from** sklearn.model\_selection **import** train\_test\_split # 训练集测试集划分
8. **from** sklearn.preprocessing **import** MinMaxScaler # 数据标准化

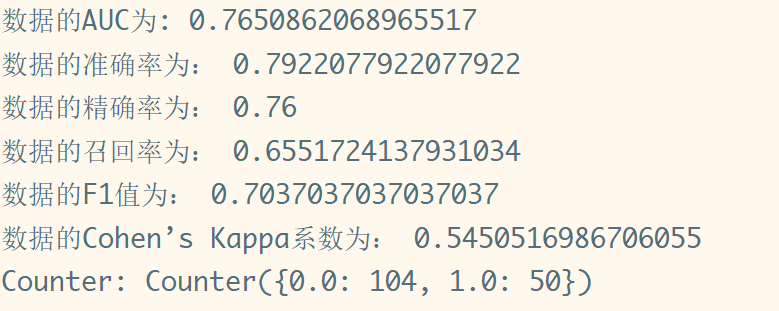
对数据集进行读取，标准化处理后，划分测试集与训练集：

1. data = pd.read\_csv('diabetes.csv') # 读取数据集
2. new\_data = data.drop(['Outcome'], axis=1) # 分离feature和label
3. scale = MinMaxScaler().fit(new\_data)  # 标准化规则
4. biao\_data = scale.transform(new\_data)  # 应用规则
6. X\_train, X\_test, y\_train, y\_test = train\_test\_split(biao\_data, data['Outcome'], test\_size=0.2, random\_state=123)
7. # 随机划分测试集训练集，比例为0.2
8. y\_test = y\_test.values
9. y\_train = y\_train.values

训练过程，根据bayes分类器、knn分类器的输出作为输入数据，对后续回归分类器进行训练：

1. **def** train(X\_train, y\_train):
2. **global** a
3. a = 0.05
4. **global** b
5. b = 0.3
6. epoch = 10 # 训练周期
7. lr = 0.005 # 学习率
8. **global** w
9. w = 0.5
10. **for** i **in** range(epoch): # 梯度下降法
11. ax = 0
12. by = 0
13. **for** j, sample **in** enumerate(X\_train):
14. x = bayes.predict1(sample) # bayes预测后验概率之差作为输入
15. y = kn.predict1(sample) # knn邻居投票票数作为输入
16. # 对参数求梯度
17. **if**(y\_train[j] == 1):
18. ax += (a \* x + b \* y - y\_train[j] + w) \* x \* 2
19. by += (a \* x + b \* y - y\_train[j] + w) \* y \* 2
20. **else** :
21. ax += (a \* x + b \* y - y\_train[j] + w) \* x
22. by += (a \* x + b \* y - y\_train[j] + w) \* y
23. ax /= X\_train.shape[0]
24. by /= X\_train.shape[0]
25. **print**(ax, by)
26. # 对参数进行梯度更新
27. a -= lr \* ax
28. b -= lr \* by
29. **print**(a, b)
30. **return** (a, b)

3、**运行效果展示**



**图 6 stacking集成分类器在测试集上效果展现**

**五、结论分析**

在本次实验中，我们比较了KNN分类器、贝叶斯分类器和Stacking集成学习分类器在分类任务上的性能表现。我们使用了一个包含特征和对应类别标签的数据集，并进行了一系列实验和性能评估。

首先，针对KNN分类器，我们观察到K值的选择对分类性能有显著影响。较小的K值可能导致过拟合，而较大的K值可能导致欠拟合。因此，在选择K值时需要进行参数调优，通过交叉验证等方法选择最优的K值。此外，KNN分类器对于具有连续属性的数据集效果较好，但在处理大规模数据时计算复杂度较高。

其次，关于贝叶斯分类器，我们发现它基于贝叶斯定理进行分类，具有较好的分类性能和数学基础。贝叶斯分类器假设特征之间是条件独立的，因此对于具有强特征独立性的数据集效果较好。它还可以通过最大后验概率估计进行参数学习，对于缺少标记数据的情况下，可以利用先验知识进行分类。

最后，我们研究了Stacking集成学习分类器。Stacking通过将多个基本分类器的预测结果结合起来，通过元分类器对它们进行整合，获得更好的分类性能。在实验中，我们观察到Stacking分类器在分类准确率上表现出色。通过层级结构和元分类器的组合，Stacking能够充分利用不同基本分类器的优势，进一步提高分类性能。

综合分析实验结果，我们得出以下结论：

对于KNN分类器，选择合适的K值是至关重要的，需要根据具体数据集进行调优，同时对于连续属性的数据集效果较好。

贝叶斯分类器适用于具有较强特征独立性的数据集，可以利用先验知识进行分类，但对于特征之间的依赖关系较强的数据集效果可能较差。

Stacking集成学习分类器在分类准确率上表现出色，通过结合多个基本分类器和元分类器的组合，能够提高分类性能。

根据实验结果，我们可以根据具体问题的需求选择适合的分类器。如果需要处理连续属性的数据集，可以考虑KNN分类器；如果特征之间具有较强独立性，可以尝试贝叶斯分类器；如果追求更高的分类性能，可以考虑使用Stacking集成学习分类器。

**六、参考文献**

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21-27.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction (2nd edition). Springer.

Zhang, H. (2004). The optimality of Naive Bayes. In Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004), 562-567.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

Wolpert, D. H. (1992). Stacked generalization. Neural Networks, 5(2), 241-259.

Xu, J., & Li, J. (2002). Stacking generalization and ensemble of classifiers. Machine Learning, 48(1-3), 58-99.

**附录**

**Bayes.py**

1. **import** pandas as pd # pandas用来读取csv数据
2. **import** numpy as np # numpy库进行数学操作
3. **import** math # 数学库
4. **from** sklearn.metrics **import** accuracy\_score,precision\_score, recall\_score,f1\_score,cohen\_kappa\_score # 评价函数相关
5. **from** sklearn.metrics **import** roc\_curve,auc # 评价函数相关
6. **from** collections **import** Counter # 计数类别个数
7. **from** sklearn.model\_selection **import** train\_test\_split # 训练集测试集划分
8. **from** sklearn.preprocessing **import** MinMaxScaler # 数据标准化
10. **import** matplotlib.pyplot as plt
12. class\_num = 2
13. feature\_len = 8
15. **def** predict(sample):
16. tmp = [0, 0]
17. **for** i **in** range(class\_num): # 枚举每个类别
18. tmp[i] = np.log(cnt[i])
19. **for** j **in** range(feature\_len): # 枚举各个特征属性
20. # 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
21. 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])
23. # 选择后验概率最大的类别作为预测类别
24. **if** tmp[0] > tmp[1]:
25. **return** 0
26. **else**:
27. **return** 1

30. **def** predict1(sample):
31. tmp = [0, 0]
32. **for** i **in** range(class\_num):
33. tmp[i] = np.log(cnt[i])
34. **for** j **in** range(feature\_len):
35. tmp[i] += np.log(1 / math.sqrt(2 \* math.pi \* va[i][j])) - (sample[j] - me[i][j]) \* (
36. sample[j] - me[i][j]) / (2 \* va[i][j])
38. # print(tmp[0], tmp[1])
39. # print((int)(tmp[0] <= tmp[1]))
40. **return** tmp[1] - tmp[0]
42. **def** train(data, y):
43. **global** cnt
44. cnt = [0, 0]
45. o = []
46. **for** i **in** range(class\_num): # 枚举类别
47. tmp = []
48. **for** j **in** range(feature\_len): # 枚举特征属性
49. tmp1 = []
50. **for** k, sample **in** enumerate(data):
51. **if** y[k] == i:
52. tmp1.append(sample[j]) # 存储该类别下该特征的值
53. cnt[i] += 1 # 类别概率计算
54. tmp.append(tmp1)
55. o.append(tmp)
57. **for** i **in** range(class\_num):
58. cnt[i] = float(cnt[i]) # 计算类别先验概率
60. **global** me # 特定类别下特征属性的均值
61. **global** va # 特定类别下特征属性的方差
62. me = np.zeros((class\_num, feature\_len), dtype=float)
63. va = np.zeros((class\_num, feature\_len), dtype=float)
65. **for** i **in** range(class\_num):
66. **for** j **in** range(feature\_len):
67. # 利用numpy计算均值方差
68. me[i][j] = np.mean(np.array(o[i][j]))
69. va[i][j] = np.var(np.array(o[i][j]))

72. **def** init():
73. data = pd.read\_csv('diabetes.csv')
75. # X\_train, X\_test, y\_train, y\_test = train\_test\_split(new\_data, data['Outcome'], test\_size=0.2, random\_state=123)
76. # X\_train = X\_train.values
77. # X\_test = X\_test.values
79. new\_data = data.drop(['Outcome'], axis=1)
80. scale = MinMaxScaler().fit(new\_data)  ## 训练规则
81. biao\_data = scale.transform(new\_data)  ## 应用规则
83. X\_train, X\_test, y\_train, y\_test = train\_test\_split(biao\_data, data['Outcome'], test\_size=0.2, random\_state=123)
84. y\_test = y\_test.values
85. y\_train = y\_train.values
87. train(X\_train, y\_train)
89. y\_pred = []
90. **for** i, sample **in** enumerate(X\_test):
91. y\_pred.append(float(predict(sample)))
92. # print(y\_test[i])
94. 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
95. **print**('数据的AUC为:', auc(fpr, tpr))
96. **print**('数据的准确率为：', accuracy\_score(y\_test, y\_pred))
97. **print**('数据的精确率为：', precision\_score(y\_test, y\_pred))
98. **print**('数据的召回率为：', recall\_score(y\_test, y\_pred))
99. **print**('数据的F1值为：', f1\_score(y\_test, y\_pred))
100. **print**('数据的Cohen’s Kappa系数为：', cohen\_kappa\_score(y\_test, y\_pred))
101. **print**('Counter:', Counter(y\_pred))
103. roc\_auc = auc(fpr, tpr)  # 计算AUC（曲线下面积）
104. **print**("AUC: %.2f" % roc\_auc)  # 打印AUC值
106. # 绘制ROC曲线
107. plt.figure()
108. plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc\_auc)
109. plt.plot([0, 1], [0, 1], 'k--')  # 绘制对角线
110. plt.xlim([0.0, 1.0])
111. plt.ylim([0.0, 1.05])
112. plt.xlabel('False Positive Rate')
113. plt.ylabel('True Positive Rate')
114. plt.title('ROC curve')
115. plt.legend(loc="lower right")
116. plt.show()

**knn.py**

1. **import** pandas as pd
2. **import** numpy as np
3. **import** math
4. **from** sklearn.metrics **import** accuracy\_score,precision\_score, recall\_score,f1\_score,cohen\_kappa\_score
5. **from** sklearn.metrics **import** roc\_curve,auc
6. **from** collections **import** Counter
7. **from** sklearn.model\_selection **import** train\_test\_split
8. **from** sklearn.preprocessing **import** MinMaxScaler
10. **import** matplotlib.pyplot as plt
12. class\_num = 2
13. feature\_len = 8
15. **def** dist(a, b): # 距离计算
16. tmp = 0
17. **for** i, x **in** enumerate(a):
18. tmp += (x - b[i]) \* (x - b[i])
19. **return** np.sqrt(tmp)
21. **def** predict(sample, k):
22. # print(sample)
23. dists = []
24. **for** i, a **in** enumerate(X\_train):
25. dists.append((dist(sample, a), Y\_train[i])) # 计算与每个样本的距离
26. dists.sort() # 排序
27. dist\_k\_min = dists[:k] # 选择最小的k个样本
28. tmp = 0
29. **for** a **in** dist\_k\_min: # 对于k个邻居样本进行投票
30. **if** a[1] == 0:
31. tmp += 1
32. **else**:
33. tmp -= 1
34. **return** tmp < 0
36. **def** predict1(sample, k=9):
37. # print(sample)
38. dists = []
39. **for** i, a **in** enumerate(X\_train):
40. dists.append((dist(sample, a), Y\_train[i]))
41. dists.sort()
42. # print(dists)
43. dist\_k\_min = dists[:k]
44. tmp = 0
45. **for** a **in** dist\_k\_min:
46. **if** a[1] == 0:
47. tmp += 1
48. **else**:
49. tmp -= 1
50. # print(tmp)
51. # print(int(tmp < 0))
52. **return** float(-tmp / 9)
54. **def** train(x, y):
55. **global** X\_train
56. **global** Y\_train
57. X\_train = x
58. Y\_train = y

61. **def** init():
62. data = pd.read\_csv('diabetes.csv') # 读取数据集
63. new\_data = data.drop(['Outcome'], axis=1) # 分离feature和label
64. scale = MinMaxScaler().fit(new\_data)  # 标准化规则
65. biao\_data = scale.transform(new\_data)  # 应用规则
67. X\_train, X\_test, y\_train, y\_test = train\_test\_split(biao\_data, data['Outcome'], test\_size=0.2, random\_state=123)
68. # 随机划分测试集训练集，比例为0.2
69. y\_test = y\_test.values
70. y\_train = y\_train.values
72. train(X\_train, y\_train)
74. y\_pred = []
75. **for** i, sample **in** enumerate(X\_test):
76. y\_pred.append(float(predict(sample, 9)))
77. # print(y\_test[i])
78. # print()
80. fpr, tpr, threshold = roc\_curve(y\_test, y\_pred)
81. **print**('数据的AUC为:', auc(fpr, tpr))
82. **print**('数据的准确率为：', accuracy\_score(y\_test, y\_pred))
83. **print**('数据的精确率为：', precision\_score(y\_test, y\_pred))
84. **print**('数据的召回率为：', recall\_score(y\_test, y\_pred))
85. **print**('数据的F1值为：', f1\_score(y\_test, y\_pred))
86. **print**('数据的Cohen’s Kappa系数为：', cohen\_kappa\_score(y\_test, y\_pred))
87. **print**('Counter:', Counter(y\_pred))
89. roc\_auc = auc(fpr, tpr)  # 计算AUC（曲线下面积）
90. **print**("AUC: %.2f" % roc\_auc)  # 打印AUC值
92. # 绘制ROC曲线
93. plt.figure()
94. plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc\_auc)
95. plt.plot([0, 1], [0, 1], 'k--')  # 绘制对角线
96. plt.xlim([0.0, 1.0])
97. plt.ylim([0.0, 1.05])
98. plt.xlabel('False Positive Rate')
99. plt.ylabel('True Positive Rate')
100. plt.title('ROC curve')
101. plt.legend(loc="lower right")
102. plt.show()
104. # init()

**fusion.py**

1. **import** main as bayes
2. **import** knn as kn
3. **import** pandas as pd
4. **import** numpy as np
5. **import** math
6. **from** sklearn.metrics **import** accuracy\_score,precision\_score, recall\_score,f1\_score,cohen\_kappa\_score
7. **from** sklearn.metrics **import** roc\_curve,auc
8. **from** collections **import** Counter
9. **from** sklearn.model\_selection **import** train\_test\_split
10. **from** sklearn.preprocessing **import** MinMaxScaler
12. **import** matplotlib.pyplot as plt
14. **def** train(X\_train, y\_train):
15. **global** a
16. a = 0.05
17. **global** b
18. b = 0.3
19. epoch = 8 # 训练周期
20. lr = 0.005 # 学习率
21. **global** w
22. w = 0.5
23. X = []
24. Y = []
25. **for** i **in** range(epoch): # 梯度下降法
26. ax = 0
27. by = 0
28. cnt = 0
29. **for** j, sample **in** enumerate(X\_train):
30. x = bayes.predict1(sample) # bayes预测后验概率之差作为输入
31. y = kn.predict1(sample) # knn邻居投票票数作为输入
32. z = int(a \* x + b \* y > 0)
33. **if**(y\_train[j] == z):
34. cnt += 1
35. # 对参数求梯度
36. **if**(y\_train[j] == 1):
37. ax += (a \* x + b \* y - y\_train[j] + w) \* x \* 2
38. by += (a \* x + b \* y - y\_train[j] + w) \* y \* 2
39. **else** :
40. ax += (a \* x + b \* y - y\_train[j] + w) \* x
41. by += (a \* x + b \* y - y\_train[j] + w) \* y
42. ax /= X\_train.shape[0]
43. by /= X\_train.shape[0]
44. cnt /= X\_train.shape[0]
45. X.append(i)
46. Y.append(cnt)
47. **print**(ax, by)
48. # 对参数进行梯度更新
49. a -= lr \* ax
50. b -= lr \* by
52. plt.plot(X, Y)
53. plt.title('Accuracy varies with training cycle')
54. plt.show()
56. **print**(a, b)
57. **return** (a, b)
59. **def** predict(sample):
60. x = bayes.predict1(sample)
61. y = kn.predict1(sample)
62. **print**(x, y, a \* x + b \* y + w)
63. **return** a \* x + b \* y > 0
65. bayes.init()
66. kn.init()
68. data = pd.read\_csv('diabetes.csv')
69. new\_data = data.drop(['Outcome'], axis=1)
70. scale = MinMaxScaler().fit(new\_data)  ## 训练规则
71. biao\_data = scale.transform(new\_data)  ## 应用规则
73. X\_train, X\_test, y\_train, y\_test = train\_test\_split(biao\_data, data['Outcome'], test\_size=0.2, random\_state=123)
74. y\_test = y\_test.values
75. y\_train = y\_train.values
77. train(X\_train, y\_train)
79. y\_pred = []
80. **for** i, sample **in** enumerate(X\_test):
81. y\_pred.append(float(predict(sample)))
82. **print**(y\_test[i])
84. fpr, tpr, threshold = roc\_curve(y\_test, y\_pred)
85. **print**('数据的AUC为:', auc(fpr, tpr))
86. **print**('数据的准确率为：', accuracy\_score(y\_test, y\_pred))
87. **print**('数据的精确率为：', precision\_score(y\_test, y\_pred))
88. **print**('数据的召回率为：', recall\_score(y\_test, y\_pred))
89. **print**('数据的F1值为：', f1\_score(y\_test, y\_pred))
90. **print**('数据的Cohen’s Kappa系数为：', cohen\_kappa\_score(y\_test, y\_pred))
91. **print**('Counter:', Counter(y\_pred))