# 逻辑回归 & 决策树

Illusionna

19:58, Thursday 14th September, 2023

## 概述

Logistic Regression 和 Decision Tree 都可以用于样本分类任务，但两者原理不同，本次作业选用“乳腺癌”权威数据集，编写 Python 脚本，以准确率和查全率为重要指标，以 $F_1$ 分数为综合指标，采用多种抽样方式划分数据集，较为全面地对比不同途径得到的结果.

相关代码：

https://gitee.com/Illusionna/OnlineSharing/tree/master/LogisticRegression_DecisionTree

# I. 数据集

## 1.1 数据集来源

https://archive.ics.uci.edu/dataset/15/breast+cancer+wisconsin+original

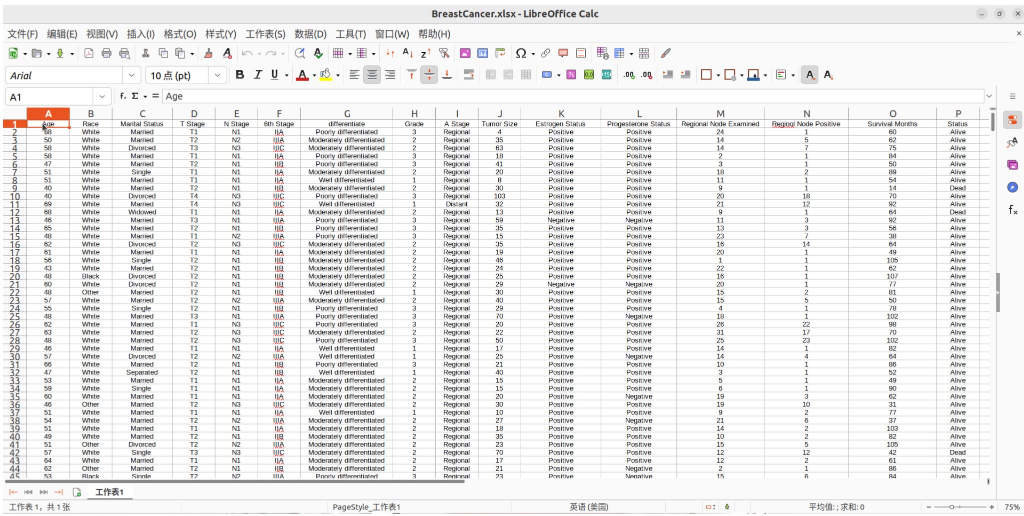


**Figure 1.1: 乳腺癌数据集视图**

## 1.2 数据集描述

乳腺癌数据集一共收录 4024 条样本，包括 15 列属性，以及对应的状态标签 “Alive” “Dead”，同时注意到存在 “Undifferentiated” “anaplastic” “Grade IV” “Regional” 属性值的一类异常标签（非 “Alive” “Dead”）.

## 1.3 数据预处理

### 1.3.1 删除异常标签

一共筛选出 19 条异常样本，直接删除.

**Table 1.1 异常样本**

| Age | Race | Marital Status | T Stage | N Stage | 6th Stage | differentiate | Grade | A Stage | Tumor Size | Estrogen Status | Progesterone Status | Regional Node Examined | Reginol Node Positive | Survival Months | Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 52 | Black | Single | T3 | N3 | IIIC | Undifferentiated | anaplastic | Grade IV | Regional | 100 | Positive | Positive | 23 | 17 | 16 |
| 38 | White | Married | T3 | N1 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 70 | Positive | Positive | 10 | 1 | 102 |
| 37 | Black | Married | T3 | N1 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 60 | Positive | Positive | 5 | 2 | 97 |
| 69 | White | Married | T1 | N1 | IIA | Undifferentiated | anaplastic | Grade IV | Regional | 20 | Positive | Positive | 17 | 3 | 85 |
| 59 | White | Single | T2 | N2 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 40 | Negative | Negative | 10 | 9 | 33 |
| 67 | White | Married | T2 | N3 | IIIC | Undifferentiated | anaplastic | Grade IV | Regional | 21 | Positive | Positive | 25 | 17 | 75 |
| 39 | White | Single | T3 | N2 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 70 | Positive | Positive | 14 | 4 | 79 |
| 69 | White | Widowed | T2 | N3 | IIIC | Undifferentiated | anaplastic | Grade IV | Regional | 28 | Positive | Positive | 14 | 13 | 13 |
| 58 | White | Single | T1 | N2 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 19 | Negative | Negative | 47 | 7 | 9 |
| 43 | White | Married | T3 | N2 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 75 | Negative | Negative | 10 | 5 | 59 |
| 62 | Black | Separated | T4 | N3 | IIIC | Undifferentiated | anaplastic | Grade IV | Regional | 70 | Positive | Positive | 10 | 10 | 34 |
| 63 | White | Married | T2 | N1 | IIB | Undifferentiated | anaplastic | Grade IV | Regional | 25 | Negative | Positive | 10 | 2 | 27 |
| 50 | White | Married | T3 | N1 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 55 | Positive | Positive | 12 | 1 | 89 |
| 57 | White | Married | T1 | N1 | IIA | Undifferentiated | anaplastic | Grade IV | Regional | 18 | Positive | Positive | 5 | 1 | 98 |
| 40 | White | Single | T3 | N1 | IIIA | Undifferentiated | anaplastic | Grade IV | Regional | 52 | Positive | Positive | 11 | 9 | 88 |
| 43 | White | Married | T4 | N3 | IIIC | Undifferentiated | anaplastic | Grade IV | Regional | 59 | Negative | Positive | 16 | 10 | 55 |
| 54 | White | Divorced | T1 | N1 | IIA | Undifferentiated | anaplastic | Grade IV | Regional | 16 | Positive | Negative | 19 | 2 | 101 |
| 45 | White | Married | T1 | N1 | IIA | Undifferentiated | anaplastic | Grade IV | Regional | 13 | Positive | Positive | 5 | 1 | 93 |
| 49 | White | Married | T2 | N1 | IIB | Undifferentiated | anaplastic | Grade IV | Regional | 28 | Positive | Positive | 12 | 3 | 71 |

### 1.3.2 字符标签编码

数据编码视图如 **Figure 1.2** 所示.

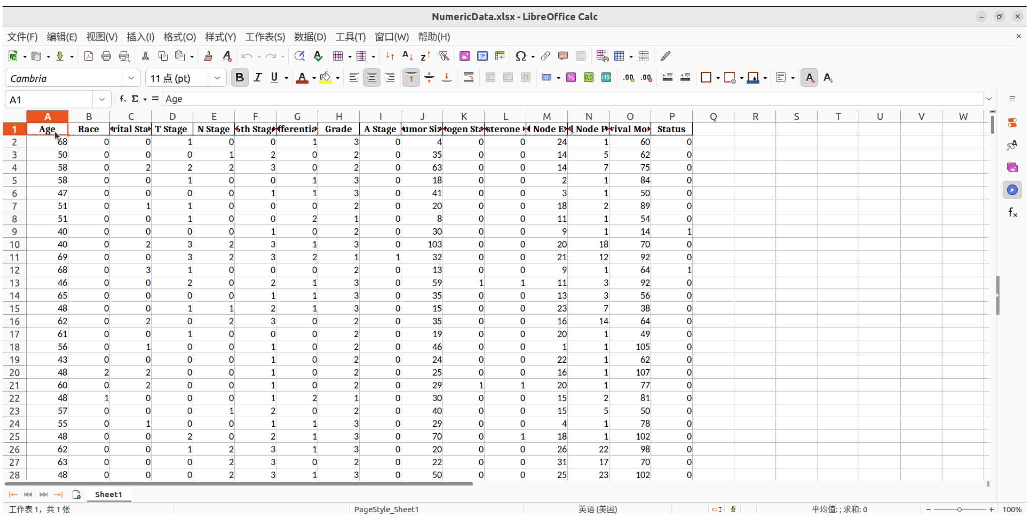


**Figure 1.2: 字符属性值编码**

给字符型分类变量按照 $0-1-2-3-\ldots$ 顺序编码，约定：

**Table 1.2 编码规定**

| Attribute \ Encoding | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Race | White | Other | Black | | |
| Marital Status | Married | Single | Divorced | Widowed | Separated |
| T Stage | T2 | T1 | T3 | T4 | |
| N Stage | N1 | N2 | N3 | | |
| 6th Stage | IIA | IIB | IIIA | IIIC | IIIB |
| differentiate | Moderately | Poorly | Well | | |
| A Stage | Regional | Distant | | | |
| Estrogen Status | Positive | Negative | | | |
| Progesterone Status | Positive | Negative | | | |
| Status | Alive | Dead | | | |

#### 1.3.3 选择性标准化、归一化

**标准化：**

$$f(x) = \frac{x-\mu}{\sigma}$$

**归一化：**

$$f(x) = \frac{x-x_{\min}}{x_{\max}-x_{\min}}$$

其中，$x$ 为某个属性下的样本值，$\mu$ 为属性相应的样本均值，$\sigma$ 为属性对应的样本标准差，$x_{\min}$ 为属性相应的样本最小值，$x_{\max}$ 为属性相应的样本最大值.

# II. 抽样划分数据集

## 2.1 Random

乳腺癌数据集随机三七开，即 70% 的训练集，30% 的测试集.

### 2.2 Hierarchical

将乳腺癌数据集按照标签“Alive”“Dead”先划分开，再各从两者中取 70% 当训练集，30% 当测试集.

### 2.3 Leave-One-Out

乳腺癌数据集只留一个样本作测试集，剩余样本全部作训练集.

### 2.4 K-Fold Cross Validation

乳腺癌数据集留若干（假设 $s$ 个）样本作测试集，剩余样本全部作训练集，使得 $K$ 轮次循环后，$s \times K = N$，其中 $N = 4005$.

### 2.5 Bootsrap

一类非参数 Monte Carlo 方法概率密度抽样，从数据集中有放回的抽取样本 $N$ 次，把每次抽到的样本构成训练集，剩下没被抽到的构成测试集.

$$\lim_{N\to+\infty} \frac{\text{Test}}{\text{Dataset}} \times 100\% = \lim_{N\to+\infty} (1-\frac{1}{N})^N = \frac{1}{e}$$

## III. 定义指标

由于精确率和召回率此消彼长，当两者出现矛盾时，此时无法确定以哪个评估指标为准，譬如，一个模型精确率高但召回率低，而另一个模型精确率低但召回率高. 因此，需要定义诸多综合型指标.

$$\text{虚警率：FPrate} = \frac{FP}{N} \times 100\%$$

$$\text{准确度：Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \times 100\%$$

$$F_1 = (1+\delta^2) \times \frac{\dfrac{TP}{TP+FP} \times \dfrac{TP}{TP+FN}}{\delta^2 \times \dfrac{TP}{TP+FP} + \dfrac{TP}{TP+FN}}$$

$$\text{G}-\text{Mean} = \sqrt{\frac{TP}{TP+FN} \times \frac{TN}{TN+FP}}$$

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+TN)(TP+FN)(TN+FP)(TN+FN)}}$$

# IV. 逻辑回归

## 4.1 Logistic Regression Results

Table 4.1 逻辑回归比对结果

| Processing | Shuffle | Epoch | Precision | Recall | Accuracy | $F_1$ | MCC |
|---|---|---|---|---|---|---|---|
| None | Random | 10 | 70.103% | 41.975% | 89.767% | 52.510% | 14.731% |
| | | 100 | 80.412% | 40.415% | 88.852% | 53.793% | 15.652% |
| | | 1000 | 75.424% | 46.354% | 89.018% | 57.419% | 17.780% |
| | Hierarchical | 10 | 72.072% | 43.716% | 88.861% | 54.422% | 16.266% |
| | | 100 | 72% | 39.344% | 88.446% | 50.883% | 14.596% |
| | | 1000 | 81.818% | 44.262% | 90.025%†‡ | 57.447% | 16.792% |
| | Bootsrap | 10 | 77.5% | 39.574% | 88.882% | 52.394% | 14.973% |
| | | 100 | 74.265% | 48.792% | 90.195%†‡ | 58.892% | 17.849% |
| | | 1000 | 79.464% | 46.809% | 91.964%†‡ | 57.329% | 16.223% |
| Standardize | Random | 10 | 56.757% | 10.995% | 84.526% | 18.421% | 3.802% |
| | | 100 | 76.190% | 15.920% | 85.108% | 26.337% | 6.144% |
| | | 1000 | 82.609% | 20.321% | 86.938% | 32.618% | 7.742% |
| | Hierarchical | 10 | 76.667% | 12.568% | 86.118% | 21.596% | 4.657% |
| | | 100 | 75% | 19.672% | 86.783% | 31.169% | 7.277% |
| | | 1000 | 100%†‡ | 20.765% | 87.947% | 34.379% | 8.081% |
| | Bootsrap | 10 | 62.162% | 11.005% | 86.254% | 18.699% | 3.780% |
| | | 100 | 77.419% | 23.645% | 88.673% | 36.226% | 8.390% |
| | | 1000 | 84.746% | 24.510% | 88.828% | 38.023% | 8.937% |
| Normalize | Random | 10 | 76.923% | 35.928% | 89.601% | 48.980% | 12.919% |
| | | 100 | 74.419% | 51.892% | 89.850% | 61.146% | 19.602% |
| | | 1000 | 82.022% | 39.891% | 89.517% | 53.676% | 15.121% |
| | Hierarchical | 10 | 77.647% | 36.066% | 88.695% | 49.254% | 13.530% |
| | | 100 | 79.048% | 45.355% | 89.859% | 57.639% | 17.131% |
| | | 1000 | 82.828% | 44.809% | 90.191%†‡ | 58.156% | 17.030% |
| | Bootsrap | 10 | 80.531% | 42.130% | 89.890% | 55.319% | 15.762% |
| | | 100 | 74.603% | 44.762% | 89.772% | 55.952% | 16.409% |
| | | 1000 | 78.814% | 44.286% | 90.444%†‡ | 56.707% | 16.159% |
| Both | Random | 10 | 72.917% | 37.838% | 88.270% | 49.822% | 14.127% |
| | | 100 | 80.734% | 45.833% | 89.601% | 58.472% | 17.759% |
| | | 1000 | 80% | 41.212% | 90.183%†‡ | 53.543% | 14.757% |
| | Hierarchical | 10 | 77.528% | 37.705% | 88.861% | 50.735% | 14.151% |
| | | 100 | 75.258% | 39.891% | 88.861% | 52.143% | 14.915% |
| | | 1000 | 82.609% | 41.530% | 89.776% | 55.273% | 15.762% |
| | Bootsrap | 10 | 76.364% | 38.898% | 89.215% | 51.334% | 14.366% |
| | | 100 | 80.357% | 41.667% | 89.821% | 54.878% | 15.581% |
| | | 1000 | 83.495% | 40.376% | 90.184%†‡ | 54.430% | 15.012% |

## 4.2 逻辑回归代码执行结果

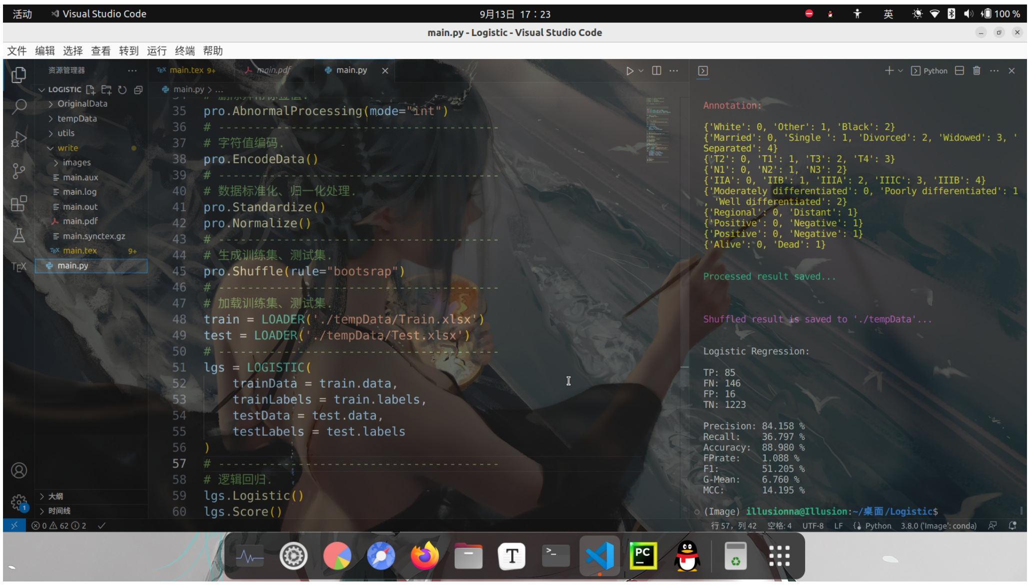


**Figure 4.1: main.py 执行效果（epoch=100）**

# V. 决策树

## 5.1 Decision Tree Results

**Table 5.1 决策树比对结果**

| Shuffle | Precision | Recall | Accuracy | FPrate | $F_1$ | G-Mean | MCC |
|---|---|---|---|---|---|---|---|
| Random | 75.862% | 49.162% | 90.100% | 2.329% | 59.661% | 7.690% | 18.318% |
| Hierarchical | 77.869% | 51.913% | 90.441% | 2.244% | 62.295% | 7.635% | 19.626% |
| Bootsrap | 83.333% | 53.488% | 91.180% | 1.534% | 63.889% | 7.046% | 19.895% |

决策树选取 Gini 系数作为评判指标，树的最大深度 maxDepth $= 4$，从决策树比对结果表格窥见：Bootsrap $>$ Hierarchical $>$ Random，即采用自助抽样得到的决策树不论是 Precision 还是 Recall 乃至 Accuracy 都较高. 并且综合性指数 $F_1 = 63.889\%$ 胜过其余两个 Shuffle.

## 5.2 决策树代码执行结果

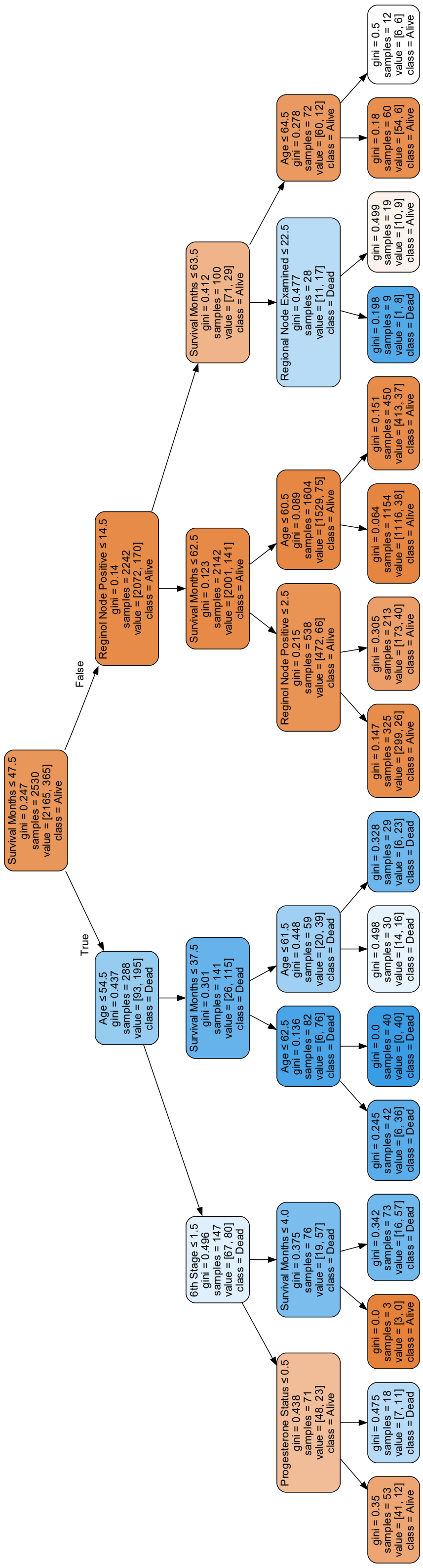


**Figure 5.1: 决策树结果**

# VI. 结果分析

Logistic Regression 的比对结果 **Table 3.1** 反映，只对起源数据集进行编码处理，不进行标准化或归一化操作，采用 Bootsrap 抽样迭代 1000 次，逻辑回归结果 Accuracy 可达 91.964%，综合回归的结果：

$$\text{None} > \text{Both} > \text{Normalize} > \text{Standardize}$$

$$\text{Bootsrap} > \text{Hierarchical} > \text{Random}$$

一个有趣结果是，Standardize 处理采用 Hierarchical 抽样，训练迭代 1000 次，测试集的 Precision 居然满达 **100%**，测试集全部查准，笔者已将对应的训练集和测试集上传至文章概述提供的链接.

决策树祖先结点 SurvivalMonths $\leqslant 47.5$，其 Gini 系数仅有 0.247，这意味分类的纯粹度相当高，决策树分类十分有效.

综上所述，逻辑回归和决策树对于分类问题（乳腺癌数据集）的预测结果效果都非常良好，最高可达 90% 左右的 Accuracy.

# Code. 代码

## Code1(Logistic): main.py

```
'''
# System --> Linux & Python3.8.0
# File ----> main.py
# Author --> Illusionna
# Create --> 2023/09/12 19:22:39
'''
# -*- encoding: utf-8 -*-


# ----------------------
# ----------------------
import os
from utils.Loader import LOADER
```

```
from utils.Processing import PROCESSING
from utils.LogisticRegression import LOGISTIC
# ------------------------
# ------------------------
def cls() -> None:
    os.system('clear')
cls()
# ------------------------
# ------------------------
# 加载乳腺癌数据集.
io = './OriginalData/BreastCancer.xlsx'
# ------------------------
lor = LOADER(io)
# ------------------------
pro = PROCESSING(
    data = lor.data,
    labels = lor.labels,
    attribute = lor.attribute
)
# ------------------------
# 删除异常标签值.
pro.AbnormalProcessing(mode="int")
# ------------------------
# 字符值编码.
pro.EncodeData()
# ------------------------
# 数据标准化、归一化处理.
pro.Standardize()
pro.Normalize()
# ------------------------
# 生成训练集、测试集.
pro.Shuffle(rule='bootsrap')
```

```
# ———————————————————————
# 加载训练集、测试集.
train = LOADER('./tempData/Train.xlsx')
test = LOADER('./tempData/Test.xlsx')
# ———————————————————————
lgs = LOGISTIC(
    trainData = train.data,
    trainLabels = train.labels,
    testData = test.data,
    testLabels = test.labels,
    epoch = 1000
)
# ———————————————————————
# 逻辑回归.
lgs.Logistic()
lgs.Score()
# ———————————————————————
# ———————————————————————
```

## Code2(Decision): main.py

```
'''
# System --> Windows & Python3.8.0
# File ----> main.py
# Author --> Illusionna
# Create --> 2023/09/14 03:41:34
'''
# -*- encoding: UTF-8 -*-


import os
from utils.Loader import LOADER
```

```
from utils.Processing import PROCESSING
from utils.DecisionTree import DECISION
# -----------------------
# -----------------------
def cls() -> None:
    os.system('cls')
cls()
# -----------------------
# -----------------------
# 加载乳腺癌数据集.
io = './OriginalData/BreastCancer.xlsx'
# # -----------------------
lor = LOADER(io)
# # -----------------------
pro = PROCESSING(
    data = lor.data,
    labels = lor.labels,
    attribute = lor.attribute
)
# -----------------------
# 删除异常标签值.
pro.AbnormalProcessing(mode="int")
# -----------------------
# 字符值编码.
pro.EncodeData()
# -----------------------
# 数据标准化、归一化处理.
# pro.Standardize()
# pro.Normalize()
# -----------------------
# 生成训练集、测试集.
pro.Shuffle(rule='bootsrap')
```

```
# ----------------------
# ----------------------
train  = LOADER('./tempData/Train.xlsx')
test  = LOADER('./tempData/Test.xlsx')
# ----------------------
tree  = DECISION(
    trainData = train.data,
    trainLabels  = train.labels,
    testData = test.data,
    testLabels  = test.labels
)
# ----------------------
tree.Decision()
# ----------------------
tree.Score()
tree.Chart(test.attribute)
# ----------------------
# ----------------------
```

## Code3: Loader.py

```
'''
# System --> Linux & Python3.8.0
# File ----> Loader.py
# Author --> Illusionna
# Create --> 2023/09/12 19:19:06
'''
# -*- encoding: utf-8 -*-


import pandas as pd

```

```
class LOADER:
    """
    数据加载类.
    """
    def __init__(self, io: str) -> None:
        self.io = io
        LOADER.Loader(self)

    def Loader(self) -> None:
        """
        Load the original data.
        """
        try:
            df = pd.read_excel(self.io, header=None)
            self.attribute = df.loc[0].tolist()
            self.labels = df.loc[:,(df.shape[1]-1)].drop(index=0).
                tolist()
            self.data = df.loc[:,[i for i in range(0, (df.shape
                [1]-1), 1)]].drop(index=0).values.tolist()
        except:
            print("Failed to load data.")
            print("数据加载失败.")
            print("Check Excel table format and path.")
            print("检查表格格式和路径.")
            print("Reference link of picture format.")
            print("参考格式图片链接.\n")
            print(r"https://gitee.com/Illusionna/OnlineSharing/raw
                /master/View_of_Excel.png")
            print("\n")
```

## Code4: Processing.py

```
'''
# System --> Linux & Python3.8.0
# File ----> Processing.py
# Author --> Illusionna
# Create --> 2023/09/12 19:20:00
'''
# -*- encoding: utf-8 -*-


import os
import random
import numpy as np
import pandas as pd
from typing import Literal

COLOURS = {
    'default':'\033[39m',
    'green':'\033[32m',
    'pink':'\033[35m',
    'red':'\033[31m',
    'orange':'\033[33m'
}

class PROCESSING:
    """
    数据预处理类.
    """
    def __init__(self, data: list, labels: list, attribute: list) ->
        None:
        self.data = data
        self.label = labels
        self.attribute = attribute
```

```python

    RULE = Literal['int', 'float']
    def AbnormalProcessing(self, mode:RULE) -> None:
        """
        异常值函数：处理标签异常的样本，视 mode 为正常.
        """
        remarkList = []
        for i in range(0, len(self.label), 1):
            if isinstance(self.label[i], eval(mode)) == True:
                remarkList.append(i)
        print(COLOURS['red'], end='')
        print('Abnormal Samples:')
        print(COLOURS['orange'])
        dictionary = {}
        for i in range(0, len(remarkList), 1):
            temp = self.data[remarkList[i]]
            dictionary[f'Error{i+1}'] = temp
        df = pd.DataFrame(dictionary).T
        print(df)
        transformerD = pd.DataFrame(self.data)
        transformerD.drop(remarkList, inplace=True)
        transformerL = pd.DataFrame(self.label)
        transformerL.drop(remarkList, inplace=True)
        print(COLOURS['green'])
        print('Abnormal samples removed...\n')
        print(COLOURS['default'])
        self.data = transformerD.values.tolist()
        self.label = transformerL.T.values.tolist()[0]

    def EncodeData(self) -> None:
        """
        属性值编码函数：将字符型值按照 0,1,2,... 编码.
```

```
        """
        matrix = []
        for i in range(0, len(self.label), 1):
            temp = list(self.data[i])
            temp.append(self.label[i])
            matrix.append(temp)
        transformer = pd.DataFrame(matrix).T
        statisticsList = []
        write = []
        for i in range(0, len(transformer), 1):
            column = transformer.values.tolist()[i]
            judge = (
                isinstance(column[0], int)
                |
                isinstance(column[0], float)
            )
            if judge:
                pass
            else:
                temp = pd.value_counts(column)
                pos = 0
                for j in range(0, len(temp), 1):
                    temp[j] = pos
                    pos = -~pos
                write.append(dict(temp))
                statisticsList.append((i, dict(temp)))
        print(COLOURS['red'], end='')
        print('Annotation:')
        print(COLOURS['orange'])
        for i in range(0, len(write), 1):
            print(write[i])
        print(COLOURS['default'])
```

```python
        data = []
        pos = 0
        for i in range(0, len(matrix), 1):
            sample = []
            for j in range(0, len(matrix[i]), 1):
                if j == statisticsList[pos][0]:
                    sample.append(write[pos][matrix[i][j]])
                    pos = -~pos
                else:
                    sample.append(matrix[i][j])
            pos = 0
            data.append(sample)
        dictionary = {}
        for i in range(0, len(self.attribute), 1):
            dictionary[self.attribute[i]] = pd.DataFrame(data).T.values.tolist()[i]
        os.makedirs('./tempData', exist_ok=True)
        pd.DataFrame(dictionary).to_excel('./tempData/NumericData.xlsx', index=None)
        print(COLOURS['green'])
        print('Processed result saved...\n')
        print(COLOURS['default'])
        for i in range(0, len(data), 1):
            self.label[i] = data[i][-1]
            del data[i][-1]
        self.data = data

    def Standardize(self) -> None:
        """
        功能: 数据标准化.\n
        函数: Z-score 法则.\n
        参数: 要求 “纯” 数据矩阵, 不含表头属性和表格索引, 类
```

```
            型 list.
        """
        self.data = pd.DataFrame(self.data).T.values.tolist()
        temp = []
        for i in range(0, len(self.data), 1):
            tempList = []
            avg = np.average(self.data[i])
            var = np.var(self.data[i])
            for j in range(0, len(self.data[i]), 1):
                value = (self.data[i][j]-avg) / var
                tempList.append(value)
            temp.append(tempList)
        self.data = temp
        self.data = pd.DataFrame(self.data).T.values.tolist()

    def Normalize(self) -> None:
        """
        功能: 数据归一化.\n
        范围: 归一化后数据区间 [0, 1].\n
        参数: 要求 “纯” 数据矩阵, 不含表头属性和表格索引, 类
            型 list.
        """
        self.data = pd.DataFrame(self.data).T.values.tolist()
        temp = []
        for i in range(0, len(self.data), 1):
            tempList = []
            for j in range(0, len(self.data[i]), 1):
                value = (self.data[i][j] - min(self.data[i])) / (
                    max(self.data[i]) - min(self.data[i]))
                tempList.append(value)
            temp.append(tempList)
        self.data = temp
```

```
        self.data = pd.DataFrame(self.data).T.values.tolist()

    MODE = Literal['random', 'hierarchical', 'LOO', 'KFCV', '
        bootsrap']
    def Shuffle(self, rule:MODE="random", num:int=4) -> None
        or list:
        """
        数据洗牌：打乱数据集，划分训练集和测试集.\n
        rule = {
            "random": RandomShuffle( )\n
            "hierarchical": HierarchicalShuffle( )\n
            "LOO" --> LeaveOneOut: LOOShuffle( )\n
            "KFCV" --> KFoldCrossValidation: KFCVShuffle( )\n
            "bootsrap": BootsrapShuffle( )
        }
        """
        os.makedirs("./tempData", exist_ok=True)
        self.matrix = []
        for i in range(0, len(self.data), 1):
            temp = self.data[i]
            temp.append(self.label[i])
            self.matrix.append(temp)
        if rule == "random":
            PROCESSING.__RandomShuffle(self)
        elif rule == "hierarchical":
            PROCESSING.__HierarchicalShuffle(self)
        elif rule == "LOO":
            PROCESSING.__LOOShuffle(self, testSetIndex=num)
        elif rule == "KFCV":
            print("Shuffled ...\n")
            return PROCESSING.__KFCVShuffle(self, K=num)
        elif rule == "bootsrap":
```

```
                PROCESSING.__BootsrapShuffle(self)
            else:
                PROCESSING.__RandomShuffle(self)
            print(COLOURS['pink'])
            print("Shuffled result is saved to './tempData'...\n")
            print(COLOURS['default'])
            self.data = pd.DataFrame(self.data).drop([len(self.data[0])
                -1], axis=1)
            self.data = self.data.values.tolist()

    def __SaveTempData(self, trainSet:list, testSet: list) -> None:
        """
        洗牌划分后的训练集和测试集寄存在 './tempData' 文件夹下.
        """
        trainSet = pd.DataFrame(trainSet).T.values.tolist()
        testSet = pd.DataFrame(testSet).T.values.tolist()
        trainDictionary = dict(zip(self.attribute, trainSet))
        testDictionary = dict(zip(self.attribute, testSet))
        dfTrain = pd.DataFrame(trainDictionary)
        dfTest = pd.DataFrame(testDictionary)
        dfTrain.to_excel('./tempData/Train.xlsx', index=False)
        dfTest.to_excel('./tempData/Test.xlsx', index=False)

    def __RandomShuffle(self) -> None:
        """
        留出法抽样：数据集随即划分，默认 ratio = 0.7，即传统的前
            后七三随机开.
        """
        random.shuffle(self.matrix)
        sampleNumber = len(self.matrix)
        ratio = 0.7
        trainNumber = int(sampleNumber*ratio)
```

```
        del ratio
        trainSet = []
        testSet = []
        for i in range(0, trainNumber, 1):
            trainSet.append(self.matrix[i])
        for i in range(trainNumber, sampleNumber, 1):
            testSet.append(self.matrix[i])
        PROCESSING.__SaveTempData(self, trainSet, testSet)

    def __HierarchicalShuffle(self) -> None:
        """
        分层抽样：默认 ratio = 0.7，从正样本中 70% 放到训练集，
            再从反样本中抽 70% 放到训练集，测试集是剩余 30% 的
            正反样本.
        """
        ratio = 0.7
        trainSet = []
        testSet = []
        positiveSet = []
        negativeSet = []
        sampleNumber = len(self.matrix)
        for i in range(0, sampleNumber, 1):
            if self.matrix[i][-1] == 1:
                positiveSet.append(self.matrix[i])
            else:
                negativeSet.append(self.matrix[i])
        random.shuffle(positiveSet)
        random.shuffle(negativeSet)
        for i in range(0, int(len(positiveSet)*ratio), 1):
            trainSet.append(positiveSet[i])
        for i in range(0, int(len(negativeSet)*ratio), 1):
            trainSet.append(negativeSet[i])
```

```
        for i in range(int(len(positiveSet)*ratio), len(positiveSet), 1):
            testSet.append(positiveSet[i])
        for i in range(int(len(negativeSet)*ratio), len(negativeSet), 1):
            testSet.append(negativeSet[i])
        random.shuffle(trainSet)
        random.shuffle(testSet)
        PROCESSING.__SaveTempData(self, trainSet, testSet)

    def __LOOShuffle(self, testSetIndex:int=0) -> None:
        """
        留一法抽样：参数 num/testSetIndex 是选取一个测试样本的索引.
        """
        sampleNumber = len(self.matrix)
        trainSet = []
        testSet = []
        testSet.append(self.matrix[testSetIndex])
        L = list(
            set([i for i in range(0, sampleNumber, 1)])
            -
            set([testSetIndex])
        )
        for i in L:
            trainSet.append(self.matrix[i])
        PROCESSING.__SaveTempData(self, trainSet, testSet)

    def __KFCVShuffle(self, K:int=4) -> list:
        """
        K-折交叉验证抽样：参数 num/K 为数据集划分成子集的个数.
```

```
        """
        random.shuffle(self.matrix)
        sampleNumber = len(self.matrix)
        tensor = []
        trainSet = []
        testSet = []
        pos = 0
        step = np.floor(sampleNumber/K)
        for i in range(0, (K-1), 1):
            matrix = []
            while pos<(step*(i+1)):
                matrix.append(self.matrix[pos])
                pos = -~pos
            tensor.append(matrix)
        matrix = []
        for t in range(pos, sampleNumber, 1):
            matrix.append(self.matrix[pos])
            pos = -~pos
        tensor.append(matrix)
        return tensor

    def __BootsrapShuffle(self) -> None:
        """
        自助法概率抽样：数据集较小、难以有效划分训练集和测试集时有效，但会改变起源数据集的分布，引入估计偏差，数据量大时建议使用留出法和 K 折交叉验证法.
        """
        sampleNumber = len(self.matrix)
        indexList = []
        for i in range(0, sampleNumber, 1):
            random.seed(random.random())
            index = random.randint(0, (sampleNumber-1))
```

```
            indexList.append(index)
        indexListTrain = list(set(indexList))
        indexListTest = list(
            set([i for i in range(0, sampleNumber, 1)])
            -
            set(indexListTrain)
        )
        trainSet = []
        testSet = []
        for i in indexListTrain:
            trainSet.append(self.matrix[i])
        for i in indexListTest:
            testSet.append(self.matrix[i])
        PROCESSING.__SaveTempData(self, trainSet, testSet)
```

## Code5: LogisticRegression.py

```
'''
# System --> Linux & Python3.8.0
# File ----> LogisticRegression.py
# Author --> Illusionna
# Create --> 2023/09/12 19:19:32
'''
# -*- encoding: utf-8 -*-


from numpy import sqrt
from sklearn.linear_model import LogisticRegression

class LOGISTIC:
    def __init__(self, trainData: list, trainLabels: list, testData:
        list, testLabels: list, epoch:int=100) -> None:
```

```
        self.trainX = trainData
        self.trainY = trainLabels
        self.testX = testData
        self.testY = testLabels
        self.epoch = epoch

    def Logistic(self) -> None:
        model = LogisticRegression(
            multi_class = "auto",
            solver = "lbfgs",
            max_iter = self.epoch
        )
        lgs = model.fit(self.trainX, self.trainY)
        prediction = lgs.predict(self.testX)
        self.TP = 0
        self.FN = 0
        self.FP = 0
        self.TN = 0
        for i in range(0, len(prediction), 1):
            if ((prediction[i] == 1) and (self.testY[i] == 1)):
                self.TP = -~self.TP
            elif ((prediction[i] == 0) and (self.testY[i] == 1)):
                self.FN = -~self.FN
            elif ((prediction[i] == 1) and (self.testY[i] == 0)):
                self.FP = -~self.FP
            elif ((prediction[i] == 0) and (self.testY[i] == 0)):
                self.TN = -~self.TN

    def Score(self, delta: float=1) -> None:
        print('Logistic Regression:\n')
        print(f'TP: {self.TP}')
        print(f'FN: {self.FN}')
```

```
        print(f'FP: {self.FP}')
        print(f'TN: {self.TN}')
        print('')
        self.precision = self.TP/(self.TP+self.FP)
        self.recall = self.TP/(self.TP+self.FN)
        self.accuracy = (self.TP+self.TN)/ (self.TP+self.TN+self.
            FP+self.FN)
        self.FPrate = self.FP / len(self.testY)
        self.F1 = (1+delta**2) * (self.precision*self.recall)/((
            delta**2)*self.precision+self.recall)
        self.GMean = sqrt((self.TP+self.TN) / ((self.TP+self.FN) *
            (self.TN+self.FP)))
        self.MCC = (self.TP*self.TN-self.FP*self.FN) / sqrt((self.
            TP+self.TN)*(self.TP+self.FN)*(self.TN+self.FP)*(self.
            TN+self.FN))
        print('Precision:  {:.3f} %'.format(100*self.precision))
        print('Recall:     {:.3f} %'.format(100*self.recall))
        print('Accuracy:   {:.3f} %'.format(100*self.accuracy))
        print('FPrate:     {:.3f} %'.format(100*self.FPrate))
        print('F1:         {:.3f} %'.format(100*self.F1))
        print('G-Mean:     {:.3f} %'.format(100*self.GMean))
        print('MCC:        {:.3f} %'.format(100*self.MCC))
        print('')
```

## Code6: DecisionTree.py

```
'''
# System --> Windows & Python3.8.0
# File ----> DecisionTree.py
# Author --> Illusionna
# Create --> 2023/09/14 04:09:16
'''
```

```
# -*- encoding: UTF-8 -*-


import os
import pydotplus
from numpy import sqrt
from sklearn import tree
from six import StringIO
from typing import Literal

def cls() -> None:
    os.system('cls')
    os.environ["PATH"] += os.pathsep + './Graphviz/bin'
cls()

class DECISION:
    def __init__(self, trainData: list, trainLabels: list, testData:
        list, testLabels: list) -> None:
        self.trainX = trainData
        self.trainY = trainLabels
        self.testX = testData
        self.testY = testLabels

    CRITERION = Literal['gini', 'entropy', 'log_loss']
    SPLITTER = Literal['best', 'random']

    def Decision(self, criterion:CRITERION='gini', splitter:
        SPLITTER='best', maxDepth:int=4) -> None:
        clf = tree.DecisionTreeClassifier(
            criterion = criterion,
            splitter = splitter,
            max_depth = maxDepth
```

```
        )
        self.clf = clf.fit(self.trainX, self.trainY)
        prediction = self.clf.predict(self.testX)
        self.TP = 0
        self.FN = 0
        self.FP = 0
        self.TN = 0
        for i in range(0, len(prediction), 1):
            if ((prediction[i] == 1) and (self.testY[i] == 1)):
                self.TP = -~self.TP
            elif ((prediction[i] == 0) and (self.testY[i] == 1)):
                self.FN = -~self.FN
            elif ((prediction[i] == 1) and (self.testY[i] == 0)):
                self.FP = -~self.FP
            elif ((prediction[i] == 0) and (self.testY[i] == 0)):
                self.TN = -~self.TN

    def Score(self, delta:float=1) -> None:
        print('Decision Tree:\n')
        print(f'TP: {self.TP}')
        print(f'FN: {self.FN}')
        print(f'FP: {self.FP}')
        print(f'TN: {self.TN}')
        print('')
        self.precision = self.TP/(self.TP+self.FP)
        self.recall = self.TP/(self.TP+self.FN)
        self.accuracy = (self.TP+self.TN)/ (self.TP+self.TN+self.
            FP+self.FN)
        self.FPrate = self.FP / len(self.testY)
        self.F1 = (1+delta**2) * (self.precision*self.recall)/((
            delta**2)*self.precision+self.recall)
        self.GMean = sqrt((self.TP+self.TN) / ((self.TP+self.FN) *
```

```
            (self.TN+self.FP)))
        self.MCC = (self.TP*self.TN-self.FP*self.FN) / sqrt((self.
            TP+self.TN)*(self.TP+self.FN)*(self.TN+self.FP)*(self.
            TN+self.FN))
        print('Precision:  {:.3f} %'.format(100*self.precision))
        print('Recall:     {:.3f} %'.format(100*self.recall))
        print('Accuracy:   {:.3f} %'.format(100*self.accuracy))
        print('FPrate:     {:.3f} %'.format(100*self.FPrate))
        print('F1:         {:.3f} %'.format(100*self.F1))
        print('G-Mean:     {:.3f} %'.format(100*self.GMean))
        print('MCC:        {:.3f} %'.format(100*self.MCC))
        print('')

    def Chart(self, attribute: list) -> None:
        image = StringIO()
        del attribute[-1]
        status = ['Alive', 'Dead']
        tree.export_graphviz(
            self.clf,
            out_file = image,
            feature_names = attribute,
            class_names = status,
            filled = True,
            rounded = True,
            special_characters = True
        )
        chart = pydotplus.graph_from_dot_data(image.getvalue())
        chart.write_pdf('DecisionTree.pdf')
```