[redacted]学生实验报告

开课学院及实验室：计算机科学与网络工程学院 513 实验室　　2023 年 11 月 20 日

学院	计算机科学与网络工程学院	年级/专业/班		姓名	[redacted]	学号	[redacted]
实验课程名称	数据挖掘与机器学习实验					成绩	
实验项目名称	逻辑回归与朴素贝叶斯分类					指导老师	[redacted]

实验二　逻辑回归与朴素贝叶斯分类

一、 实验目的

本实验课程是计算机、人工智能、软件工程等专业学生的一门专业课程，通过实验，帮助学生更好地掌握数据挖掘与机器学习相关概念、技术、原理、应用等；通过实验提高学生编写实验报告、总结实验结果的能力；使学生对机器学习模型、算法等有比较深入的认识。

要掌握的知识点如下：

1. 掌握机器学习中涉及的相关概念、模型、算法；
2. 熟悉机器学习模型训练、验证、测试的流程；
3. 熟悉常用的数据预处理方法；
4. 掌握逻辑回归、贝叶斯分类的表示、求解及编程。

二、基本要求

1. 实验前，复习《数据挖掘与机器学习》课程中的有关内容。
2. 准备好实验数据，编程完成实验内容，收集实验结果。
3. 独立完成实验报告。

三、实验软件

推荐使用 Python 编程语言（允许使用 numpy 库，需实现详细实验步骤，不允许直接调用 scikit-learn 中回归、分类等高层 API）。

四、实验内容：

基于 Adult 数据集，完成关于收入是否大于 50K 的逻辑回归分类、朴素贝叶斯模型训练、测试与评估。

1 准备数据集并认识数据

下载 Adult 数据集

http://archive.ics.uci.edu/ml/datasets/Adult

了解数据集各个维度特征及预测值的含义

2 探索数据并预处理数据

观察数据集各个维度特征及预测值的数值类型与分布

预处理各维度特征，参考：

https://blog.csdn.net/SanyHo/article/details/105304292

3 训练模型

编程实现训练数据集上逻辑回归模型的梯度下降参数求解、朴素贝叶斯参数统计

4 测试和评估模型

在测试数据集上计算所训练模型的准确率、AUC 等指标

五、实验报告

（1）简要介绍逻辑回归分类的原理

逻辑回归是一种广义上的线性回归分析模型，简单来看就是将线性回归的结果 y 通过一个映射函数 sigmoid 映射到 0-1 区间，将 sigmoid 函数输出当做概率值，当概率值大于 0.5 时分类为正类，反之反类。

Sigmoid：

$$g(z)=\frac{1}{1+e^{-z}}$$

模型：

$$h_\theta(x)=g(\theta^T X)=\frac{1}{1+e^{-\theta^T X}}$$

$$\hat{y}=\begin{cases}1, h_\theta(x)>0.5\\ 0, h_\theta(x)<0.5\end{cases}$$

$$令\begin{cases}P_\theta(y=1|x)=h_\theta(x)\\ P_\theta(y=0|x)=1-h_\theta(x)\end{cases}\xrightarrow{合并}P_\theta(y|x)=(h_\theta(x))^y(1-h_\theta(x))^{1-y}$$

$$令\begin{cases}h_\theta(x)=p\\ 1-h_\theta(x)=1-p\end{cases}\xrightarrow{原式变换为}P_\theta(y|x)=p^y\cdot(1-p)^{1-y}$$

这里实际上将sigmoid输出当做概率，我们目标追求正确分类的概率$P_\theta(y|x)$要越高越好，使用**最大似然估计**来求使$P_\theta(y|x)$达到最大的参数θ，故可以得到参数θ的似然函数如下：

似然函数：[1]

$$\begin{aligned}L(\theta)=P_\theta(y|x)&=\prod_{j=1}^{N}P_\theta(y^j|x^j)\\ &=\prod_{j=1}^{N}p^{y^j}(1-p)^{1-y^j}\\ &=\ln(\prod_{j=1}^{N}p^{y^j}(1-p)^{1-y^j})\\ &=\sum_{j=1}^{N}\ln(p^{y^j}(1-p)^{1-y^j})\\ &=\sum_{j=1}^{N}(y^j\ln p+(1-y^j)\ln(1-p))\end{aligned}$$

[1] 似然函数是衡量样本成为观察到数据的概率

损失函数：

$$J(\theta) = -\frac{1}{m}L(\theta) = -\frac{1}{m}\sum_{j=1}^{N}(y^j \ln p + (1-y^j)\ln(1-p))$$

目标：

$$\min J(\theta) = \max L(\theta)$$
$$\hat{\theta} = \arg\min J(\theta) = \arg\max L(\theta)$$

梯度下降求解：

对 sigmoid 函数 进行求导：

$$\begin{aligned}\frac{\partial}{\partial z}g(z) &= \frac{\partial}{\partial z}\frac{1}{1+e^{-z}} \\ &= \frac{\partial}{\partial z}(1+e^{-z})^{-1} \\ &= -(1+e^{-z})^{-2}(1+e^{-z})' \\ &= -(1+e^{-z})^{-2}(-e^{-z}) \\ &= (1+e^{-z})^{-2}(e^{-z})\end{aligned}$$

对 1-sigmoid 函数 进行求导：

$$\begin{aligned}\frac{\partial}{\partial z}(1-g(z)) &= (-\frac{1}{1+e^{-z}})' \\ &= -(1+e^{-z})^{-2}(e^{-z})\end{aligned}$$

对 对数 sigmoid 函数 进行求导：

$$\begin{aligned}\frac{\partial}{\partial z}\ln g(z) &= \frac{\partial}{\partial z}\ln(\frac{1}{1+e^{-z}}) \\ &= (1+e^{-z})(1+e^{-z})^{-2}(e^{-z}) \\ &= \frac{e^{-z}}{1+e^{-z}} \\ &= 1-\frac{1}{1+e^{-z}} \\ &= 1-g(z)\end{aligned}$$

对 对数 1-sigmoid 函数 进行求导：

$$\begin{aligned}\frac{\partial}{\partial z}\ln(1-g(z)) &= \frac{\partial}{\partial z}\ln(1-\frac{1}{1+e^{-z}}) \\ &= -\frac{1}{1-\frac{1}{1+e^{-z}}}(1+e^{-z})^{-2}(e^{-z}) \\ &= -\frac{1+e^{-z}}{e^{-z}}(1+e^{-z})^{-2}(e^{-z}) \\ &= -g(z)\end{aligned}$$

对$J(\theta)$函数进行求导：

$$\begin{aligned}\frac{\partial}{\partial\theta_i}J(\theta)&=\frac{\partial}{\partial\theta_i}-\frac{1}{m}\sum_{j=1}^{N}(y^j\ln p+(1-y^j)\ln(1-p))\\&=\frac{\partial}{\partial\theta_i}-\frac{1}{m}\sum_{j=1}^{N}(y^j\ln g(\theta^TX^j)+(1-y^j)\ln(1-g(\theta^TX^j)))\\&=-\frac{1}{m}\sum_{j=1}^{N}(y^j(\ln g(\theta^TX^j))'+(1-y^j)(\ln(1-g(\theta^TX^j)))')\\&=-\frac{1}{m}\sum_{j=1}^{N}(y^j(1-\mathrm{g}(\theta^TX^j))x_i^{\,j})+(y^j-1)g(\theta^TX^j)x_i^{\,j}\\&=-\frac{1}{m}\sum_{j=1}^{N}x_i^{\,j}y^j-x_i^{\,j}y^j\mathrm{g}(\theta^TX^j)-g(\theta^TX^j)x_i^{\,j}+x_i^{\,j}y^jg(\theta^TX^j)\\&=-\frac{1}{m}\sum_{j=1}^{N}(y^j-g(\theta^TX^j))x_i^{\,j}\end{aligned}$$

计算细节：

$$(\ln g(\theta^TX))'=\frac{\partial}{\partial\theta_i}\ln g(\theta^TX)$$

$$\begin{aligned}\frac{\partial\ln g(z)}{\partial z}\frac{\partial z}{\partial\theta_i}&=(1-\mathrm{g}(z))\frac{\partial}{\partial\theta_i}\theta^TX\\&=(1-\mathrm{g}(z))x_i\\&=(1-\mathrm{g}(\theta^TX))x_i\end{aligned}$$

$$(\ln(1-g(\theta^TX)))'=\frac{\partial}{\partial\theta_i}\ln(1-g(\theta^TX))$$

$$\begin{aligned}\frac{\partial\ln(1-g(z))}{\partial z}\frac{\partial z}{\partial\theta_i}&=-g(z)\frac{\partial}{\partial\theta_i}\theta^TX\\&=-g(\theta^TX)x_i\end{aligned}$$

说明：$\begin{cases}\theta\in\mathbb{R}^{d\times1}\\X\in\mathbb{R}^{N\times d}\\Y\in\mathbb{R}^{N\times1}\end{cases}$

梯度下降更新公式： $\theta_i:=\theta_i-\alpha\frac{1}{m}\sum_{j=1}^{N}(g(\theta^TX^j)-y^j)x_i^{\,j}$

（2）简要介绍朴素贝叶斯分类的原理

朴素贝叶斯分类是基于贝叶斯定理与待分类项的各个属性独立同分布假设下的生成式模型。朴素贝叶斯分类通过学习样本的属性分布和先验概率，利用贝叶斯定理计算后验概率，然后选择具有最大后验概率的类别作为样本的分类结果。

贝叶斯定理：

$$P(Y|X)=\frac{P(X|Y)P(Y)}{P(X)}$$

重要概念：

先验概率：$P(Y)$

后验概率：$P(Y|X)$

似然：$P(X|Y)$

证据：$P(X)$

似然 $P(X|Y)$ 是关于各个特征维度的联合分布，若无样本各个特征独立同分布的假设，高维联合分布计算非常非常困难，故假设样本各个特征独立同分布，得：

$$P(X=x|Y=c_k)=\prod_{i=1}^{d}P(X_i=x_i|Y=c_k)$$

贝叶斯定理原式变换为：

$$P(Y=c_k|X=x)=\frac{\prod_{i=1}^{d}P(X_i=x_i|Y=c_k)P(Y=c_k)}{\prod_{i=1}^{d}P(X_i=x_i)}$$

对于给定的训练数据集来说，$\prod_{i=1}^{d}P(X_i=x_i)$ 不会改变，可以去除该分母，变为：

$$P(Y=c_k|X=x)=\prod_{i=1}^{d}P(X_i=x_i|Y=c_k)P(Y=c_k)$$

为使分类准确率达到最高，使用极大似然估计来估计 $P(Y=c_k)$：

$$P(Y=c_k)=\frac{\sum_{j=1}^{N}I(y^j=c_k)}{N},k=1,2,...,K$$

设第 i 个特征 x_i 可能取值集合为 $\{a_{i1},a_{i2},...,a_{iS_i}\}$，$P(X_i=x_i|Y=c_k)$ 极大似然估计为：

$$P(X_i = x_i \mid Y = c_k) = \frac{P(X_i = x_i, Y = c_k)}{P(Y = c_k)} = \frac{\sum_{j=1}^{N} I(x_i^j = a_{il}, y^j = c_k)}{\sum_{j=1}^{N} I(y^j = c_k)}$$

$i = 1, 2, ..., d$

$j = 1, 2, ..., N$

$l = 1, 2, ..., S_i$

给定一个输入 $x = (x_1, x_2, ..., x_d)$，其预测类别为：

$$\hat{y} = \arg\max_{c_k} \prod_{i=1}^{d} P(X_i = x_i \mid Y = c_k) P(Y = c_k)$$

（3）程序清单（包含详细求解步骤）

0.导包

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

%matplotlib inline
%config InlineBackend.figure_format = 'svg'
```

1. 数据处理

1）导入数据

```
# 导入数据
df = pd.read_csv('./adult.data',
                 names=['age', 'workclass', 'fnlwgt', 'education',
'education-num', 'marital-status', 'occupation',
                        'relationship', 'race', 'sex', 'capital-gain',
'capital-loss', 'hours-per-week',
                        'native-country', 'income'])

# fnlwgt 是编号
df.drop('fnlwgt', axis=1, inplace=True)

df
```

```
df.info()
```

2）缺失值处理

```
# 去除所有列中字符串两侧的空格
df_ = df.copy()

for col in df_.columns:
    if df_[col].dtype == object:  # 只处理字符串列
        df_[col] = df_[col].str.strip()

# 将 "?" 替换为 NaN
df_.replace("?", np.nan, inplace=True)

# 删除缺失值样本
df_.dropna(axis=0, inplace=True)

# 打印数据框信息
df_.info()
```

2. 逻辑回归求解

1）数据归一化（方便梯度下降）

```
def normalize(data):
    for col in data.columns:
        if np.issubdtype(data[col].values.dtype, np.number):  # Check if
the column contains numerical values
            sigma = np.std(data[col], axis=0)
            mu = np.mean(data[col], axis=0)
            data[col] = (data[col] - mu) / sigma

    return data


# 归一化
norm_df = normalize(df_)
norm_df
```

2）编码

```
encode_df = norm_df.copy()  # 存放编码后的数据
encode_list = {}  # 编码器，方便后续的反编码

for col in encode_df.columns:
```

```
    if encode_df[col].dtype == object:  # 字符型数据
        encode_list[col] = LabelEncoder()
        encode_df[col] = encode_list[col].fit_transform(encode_df[col])

encode_df
```

3）划分数据集

```
# 拆分数据集
feature = encode_df.iloc[:, 0:-1]
label = encode_df.iloc[:, -1]

Xtrain, Xtest, Ytrain, Ytest = train_test_split(feature, label,
test_size=0.1, random_state=1314)

X = np.array(Xtrain).reshape(-1,13)
y = np.array(Ytrain).reshape(-1, 1)
Xtest = Xtest.values
Ytest = Ytest.values
```

4）求解

```
# 编写一个类将逻辑回归封装起来
class LogisticRegression:
    def __init__(self):
        self.theta = None
        self.epochs_loss = []
        self.fit_epoch = -1

    # Sigmoid 函数
    def _sigmoid(self, X):
        return 1 / (1 + np.exp(-(X @ self.theta)))

    # 损失函数
    def _loss(self, y, y_pred):
        m = len(y)
        epsilon = 1e-5
        # 计算损失
        J = - (1 / m) * (np.sum(y * np.log(y_pred + epsilon) + (1 - y) *
np.log(1 - y_pred + epsilon)))
        return J

    # 训练函数
    def fit(self, X, y, lr=0.01, epochs=1500):
        m, n = X.shape
```

```
        self.fit_epoch = epochs

        # 初始化参数为标准正态分布
        self.theta = np.random.randn(n, 1)

        for epoch in range(epochs):
            # 计算激活值
            h = self._sigmoid(X)
            # 计算梯度
            gradient = X.T @ (h - y) / m
            # 计算损失并保存
            J = self._loss(y, h)
            self.epochs_loss.append(J)
            # 更新参数
            self.theta -= lr * gradient

    # 预测函数
    def predict(self, X):
        # 异常处理
        if self.theta is None:
            raise ValueError("请先调用 fit 进行训练！")
        # 进行预测
        h = self._sigmoid(X)
        return (h >= 0.5).astype(int)

    # 绘图函数:绘制损失函数梯度下降曲线
    def drawloss(self):
        sns.lineplot(x=np.arange(self.fit_epoch), y=self.epochs_loss,
label='Loss Curve')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.title('Gradient Descent Loss Curve')
        plt.savefig("1.png",dpi=800)
        plt.show()

    # 获取训练后的 theta
    def report(self):
        # 创建 DataFrame 以查看结果
        result = pd.DataFrame({"ColumnName": list(feature.columns),
"Theta": self.theta.flatten()})
        return result

    # 计算准确率
    def score(self, label_true, label_predict):
```

```
        # 确保 label_true 和 label_predict 的长度相同
        if len(label_true) != len(label_predict):
            raise ValueError("label_true 和 label_predict 的长度不一致")

        # 计算准确率并返回
        accuracy = sum(label_true.reshape(-1, 1) ==
label_predict.reshape(-1, 1)) / len(label_predict)
        return accuracy

# 实例化
log_reg = LogisticRegression()
# 训练模型
log_reg.fit(X, y)
# 查看训练报告
log_reg.report()
# 绘图
log_reg.drawloss()
```

3. 贝叶斯回归求解

0）查看数据分布

```
# 创建一个 2x3 的子图布局
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

# 绘制第一个直方图：'age'
sns.histplot(data=df_['age'], kde=True, bins=25, ax=axes[0, 0])
axes[0, 0].set_title('Age')

# 绘制第二个直方图：'education-num'
sns.histplot(data=df_['education-num'], kde=True, bins=10, ax=axes[0, 1])
axes[0, 1].set_title('Education Num')

# 绘制第三个直方图：'native-country'
sns.histplot(data=df_['native-country'], kde=True, bins=10, ax=axes[0,
2])
axes[0, 2].set_title('Native Country')
axes[0, 2].set_xticklabels(axes[0, 2].get_xticklabels(), rotation=90)

# 绘制第四个直方图：'hours-per-week'
sns.histplot(data=df_['hours-per-week'], kde=True, bins=10, ax=axes[1,
0])
axes[1, 0].set_title('Hours per Week')

# 绘制第五个直方图：'capital-gain'
```

```
sns.histplot(data=df_['capital-gain'], kde=True, bins=10, ax=axes[1, 1])
axes[1, 1].set_title('Capital Gain')

# 绘制第六个直方图：'capital-loss'
sns.histplot(data=df_['capital-loss'], kde=True, bins=10, ax=axes[1, 2])
axes[1, 2].set_title('Capital Loss')

# 调整子图之间的间距
plt.tight_layout()
plt.savefig("1.png",dpi=800)
plt.show()
```

1）连续型变量离散化

```
encode_df_ = df_.copy()

# 划分'age'属性为'0-25', '25-45', '45-65', '65-100'四类
encode_df_['age'] = pd.cut(encode_df_['age'], bins=[-1, 25, 45, 65, 100],
labels=['0-25', '25-45', '45-65', '65-100'],
                           include_lowest=True)

# 划分'education-num'属性为'<5', '5-10', '>10'三类
encode_df_['education-num'] = pd.cut(encode_df_['education-num'],
bins=[-1, 5, 10, float('inf')],
                                     labels=['<5', '5-10', '>10'],
include_lowest=True)

# 划分'native-country'属性为'USA'和'not USA'两类
encode_df_['native-country'] = encode_df_['native-country'].apply(lambda
x: 1 if x == 'United-States' else 0)

# 划分'hours-per-week'属性为'<40', '=40', '>40'三类
encode_df_['hours-per-week'] = pd.cut(encode_df_['hours-per-week'],
bins=[-1, 40, 40.1, float('inf')],
                                      labels=['<40', '=40', '>40'],
include_lowest=True)

# 划分'capital-gain'属性为'=0'和'>0'两类
encode_df_['capital-gain'] = pd.cut(encode_df_['capital-gain'],
bins=[float('-inf'), 0, float('inf')],
                                    labels=['=0', '>0'],
include_lowest=True)

# 划分'capital-loss'属性为'>0'和'=0'两类
```

```
encode_df_['capital-loss'] = pd.cut(encode_df_['capital-loss'],
bins=[float('-inf'), 0, float('inf')],
                                    labels=['>0', '=0'],
include_lowest=True)

encode_df_
```

2）编码

```
# 使用LabelEncoder对分类变量进行编码
encode_list_ = {}
for col in encode_df_.columns:
    encode_list_[col] = LabelEncoder()
    encode_df_[col] = encode_list_[col].fit_transform(encode_df_[col])

encode_df_
```

3）划分数据集

```
# 拆分数据集
feature_ = encode_df_.iloc[:, 0:-1]
label_ = encode_df_.iloc[:, -1]

Xtrain_, Xtest_, Ytrain_, Ytest_ = train_test_split(feature_, label_,
test_size=0.2, random_state=1314)

X = Xtrain_
y = Ytrain_
```

4）求解

```
class NaiveBayesClassifier:
    def __init__(self):
        self.class_probs = {}  # 存储 P(Y)
        self.feature_probs = {}  # 存储 P(X|Y)
        self.epsilon = 1e-10  # 防止log(0)

    # 训练函数
    def fit(self, X, y):
        # 获取样本数量和特征数量
        num_samples, num_features = X.shape

        # 获取所有唯一的类别（目标变量的不同取值）
        self.classes = np.unique(y)
```

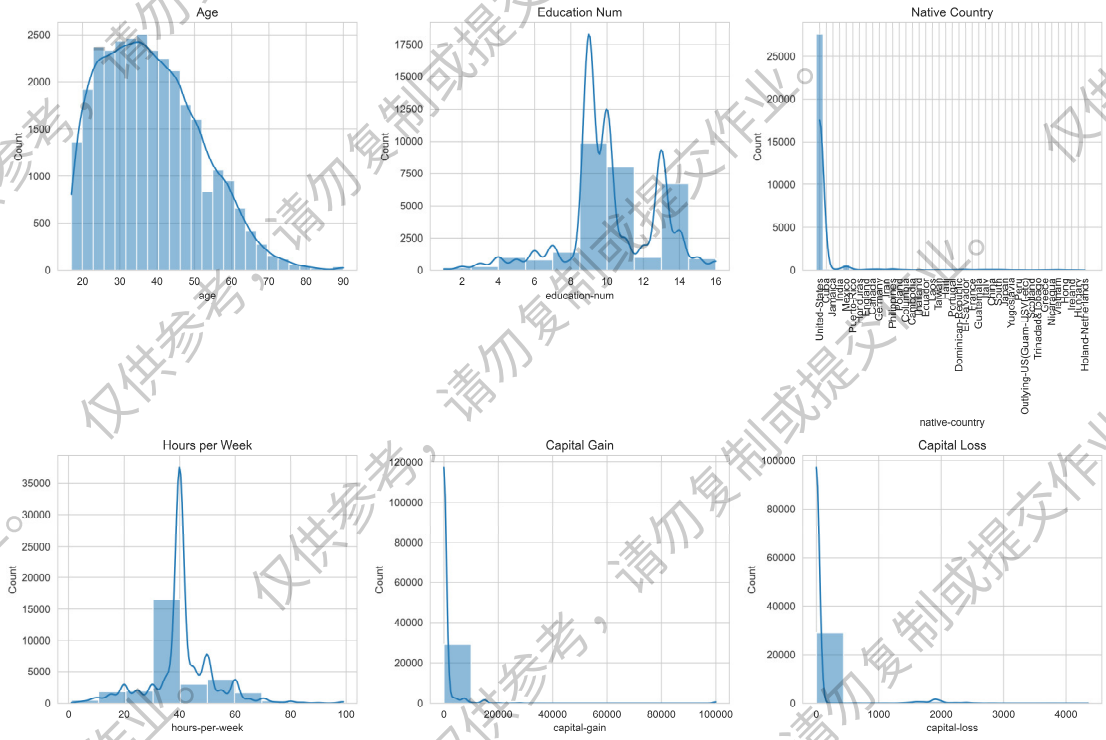
```
        # 计算主循环，遍历每个类别
        for c in self.classes:
            # 计算 P(Y)
            self.class_probs[c] = np.sum(y == c) / num_samples

            # 计算对于每个特征 P(X_i|Y)
            self.feature_probs[c] = []
            # 遍历每个特征列
            for feature_name in X.columns:
                # 获取当前特征列的唯一取值
                unique_values = np.unique(X[feature_name])

                # 存储当前特征在给定类别下的概率
                feature_prob = {}
                # 遍历当前特征的所有可能取值
                for value in unique_values:
                    # Laplace 平滑以处理未出现的值，计算 P(X_i = value | Y = c)
                    feature_prob[value] = (np.sum((X[feature_name] ==
value) & (y == c)) + 1) / (
                            np.sum(y == c) + len(unique_values))
                # 将当前特征在给定类别下的概率存储起来
                self.feature_probs[c].append(feature_prob)

    # 预测函数
    def predict(self, X):
        # 初始化一个列表来存储预测结果
        predictions = []

        # 遍历样本数据集中的每一行
        for sample in X.values:
            # 初始化最大概率和预测类别
            max_prob = -1
            predicted_class = None

            # 遍历每个类别，计算 P(Y|X)
            for c in self.classes:
                # 计算 P(Y|X) = P(Y) * P(X1|Y) * P(X2|Y) * ...
                prob = np.log(self.class_probs[c])

                # 遍历当前样本的每个特征值
                for feature_index, value in enumerate(sample):
                    # 使用取对数的方式累加特征条件概率，避免数值下溢
```

```
                    prob +=
np.log(self.feature_probs[c][feature_index].get(value,
self.epsilon))  # 避免 log(0) 添加一个小值

                # 选择概率最高的类别
                if prob > max_prob or predicted_class is None:
                    max_prob = prob
                    predicted_class = c

            # 将预测的类别添加到结果列表中
            predictions.append(predicted_class)

        # 返回最终的预测结果
        return predictions

    # 获取训练结果
    def report(self):
        # P(Y)
        print("P(Y): ")
        for c, prob in zip(self.classes, self.class_probs.values()):
            print(f"Class {c}: {prob}")

        print(f"-----------------------------------------------------------
-\n")

        # P(X|Y)
        print("\nP(X|Y): ")
        for c, feature_probs_list in self.feature_probs.items():
            print(f"\nClass {c}:")
            for feature_name, feature_prob_dict in zip(encode_df_.columns,
feature_probs_list):
                print(f"\nFeature {feature_name}:")
                for value, prob in feature_prob_dict.items():
                    print(f"Value {value}: {prob}")

    # 计算准确率
    def score(self, label_true, label_predict):
        # 确保 label_true 和 label_predict 的长度相同
        if len(label_true) != len(label_predict):
            raise ValueError("label_true 和 label_predict 的长度不一致")

        # 计算准确率并返回
        accuracy = sum(label_true == label_predict) / len(label_predict)
        return accuracy
```

```
# 实例化
nb = NaiveBayesClassifier()
# 训练
nb.fit(Xtrain_, Ytrain_)
# 查看模型参数
nb.report()
# 计算准确率
print(f"准确率: {nb.score(Ytest_, nb.predict(Xtest_))}")
```

（4）展示实验结果

1. 数据集特征分布



可以从中看到特征 Age 分布主要集中在 50 的左侧，区间为 20-90，可按 25 为步长在 0-100 区间内进行离散化；特征 Education-num 分布区间为 0-16，其中<5 与>10 的人数较少，5-10 的人数较多，离散化为<5、5-10、>10 三个区间；特征 Native Country 分布几乎全部集中在 USA，可离散化为 USA 与 Not USA 两个区间；特征 Capital Gain 与特征 Capital Loss 同理，划分为=0 与>0 两类。

2. 逻辑回归结果

准确率：[0.79946967]

13 行 13 行 × 2 列

	ColumnName	Theta
0	age	0.473629
1	workclass	0.058775
2	education	0.017858
3	education-num	0.526521
4	marital-status	-0.212680
5	occupation	0.008861
6	relationship	-0.414001
7	race	-0.074303
8	sex	-0.691151
9	capital-gain	1.901309
10	capital-loss	0.246293
11	hours-per-week	0.592127
12	native-country	-0.012703

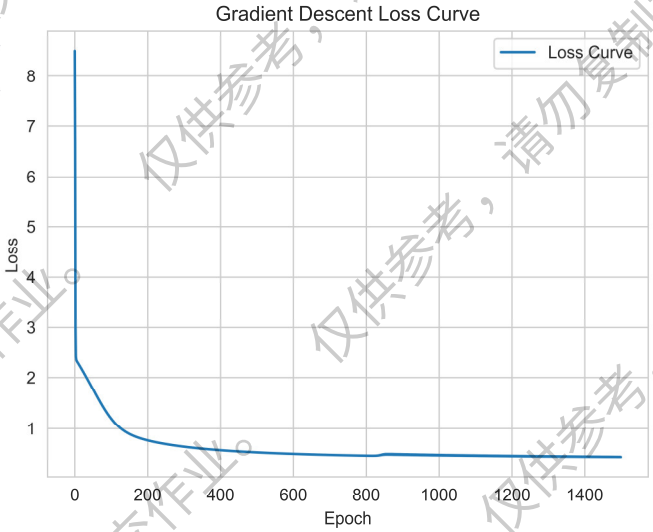


图 1 逻辑回归实验结果

3. 朴素贝叶斯结果

```
P(Y):
Class 0: 0.7517924489203862
Class 1: 0.24820755107961373
--------------------------------

P(X|Y):

Class 0:

Feature age:
Value 0: 0.7562010803660015
Value 1: 0.24379891963399847

Feature workclass:
Value 0: 0.025513859040061718
Value 1: 0.06342646167410591
Value 2: 0.7678955199206481
Value 3: 0.020554361602468728
Value 4: 0.07995811979941587
Value 5: 0.04199041163828732
Value 6: 0.0006612663250123987

Feature education:
Value 0: 0.03387309980171844
Value 1: 0.04406256884776383
Value 2: 0.015587133729896454
```

```
Class 1:

Feature age:
Value 0: 0.986145885494909
Value 1: 0.01385411450509097

Feature workclass:
Value 0: 0.05086724482988659
Value 1: 0.08155436957971982
Value 2: 0.6467645096731154
Value 3: 0.07921947965310207
Value 4: 0.0943962641761174
Value 5: 0.04703135423615744
Value 6: 0.0001667778519012675

Feature education:
Value 0: 0.007160699417152373
Value 1: 0.007993338884263113
Value 2: 0.00466278101582015
Value 3: 0.0006661115736885929
Value 4: 0.0016652789342214821
Value 5: 0.00466278101582015
Value 6: 0.003164029975020816
Value 7: 0.03363863447127394
Value 8: 0.046960865945045796
Value 9: 0.28043297252289756
Value 10: 0.03763530391340549
Value 11: 0.21565362198168192
Value 12: 0.12223147377185678
```

准确率: 0.7903199071771921

图 2 朴素贝叶斯实验结果

（5）讨论实验结果，分析各个特征与目标预测类别的正负相关性

1. 实验结果讨论

逻辑回归

在逻辑回归的实验中，需要先对连续性数据进行标准化处理，否则梯度下降时可能会因为数据各个维度的差异过大造成计算异常，然后需要对分类变量进行编码，在本实验中我采用标签编码的方式进行编码。梯度下降算法设定学习率 0.01，迭代次数 1500 次，得到分类准确率为 79.95%。

朴素贝叶斯

在逻辑回归的实验中，需要先对连续性数据进行离散化处理，详细处理请见 实验结果展示的 1.数据集特征分布 部分，然后根据公式计算先验概率与似然，计算完毕即训练完毕，生成式模型无需迭代优化。得到分类准确率为 79.03%，准确率较逻辑回归略低一点点，但因为朴素贝叶斯无需迭代优化，所以训练速度比逻辑回归快得多。

2. 分析各个特征与目标预测类别的正负相关性

	ColumnName	Theta
0	age	0.473629
1	workclass	0.058775
2	education	0.017858
3	education-num	0.526521
4	marital-status	-0.212680
5	occupation	0.008861
6	relationship	-0.414001
7	race	-0.074303
8	sex	-0.691151
9	capital-gain	1.901309
10	capital-loss	0.246293
11	hours-per-week	0.592127
12	native-country	-0.012703

逻辑回归的系数如左图所示，从图中可以看到各个特征的正负相关性，根据 sigmoid 函数的图像可以知道，输入 $\theta^T X$ 越大，分类器越趋向于正类，故系数为正的特征如 age 对分类为正类起正相关性；同理输入 $\theta^T X$ 越小，分类器越趋向于负类，故系数为负的特征如 sex 对分类为负类起正相关性、对分类为正类起负相关性。