机器学习实验报告

题　　目：	逻辑回归问题
学生姓名：	张芮熙
学　　号：	22354188
指导教师：	马倩
专业班级：	22 智科 3 班

2021 年 5 月

本科生院制

目录

1. 实验目的、思路：

Iris 数据集是常用的分类实验数据集,也称鸢尾花卉数据集，是一类多重变量分析的数据集。我们实验选取数据集的部分内容，包含训练集中的 80 个数据样本和测试集的 20 个样本，分为 2 类，每个数据包含 2 个属性。可通过花萼长度(x1)，花萼宽度(x2)2 个属性预测鸢尾花卉属于（Setosa，Versicolour）二个种类中的哪一类。

核心代码实现与测试结果

首先，导入训练数据存放在 dataframe 中，测试数据放在 dataframe2 中，再打开 Excel 表可以发现，数据不是完整的，有缺失或者是很离谱的数据，在这里，我把缺失的(“NaN”)，不合理数据用所有有效数据的中位数代替，这样保证了总体数据的中位数没有改变，确保了样本数据的真实性，同时，最后一列 type 列中的元素是字符形“Iris-setosa”,“Iris-versicolor”用替换函数把值换成 0,1，实现数据的数字化。（0 为负类，1 为正类）

1.

1) 使用pandas库将训练数据集'flower_train.csv'与测试数据集'flower_test.csv'载入到Dataframe对象中，并判断训练集中每列数据是否有缺失值或者不合理的数值，如果有，请在不删除数据的前提下进行处理，而测试集为完好的数据集，不需要进行操作。由于花卉类型(type)为字符串类型，请将花卉类型转换为适合模型训练的类型

```
dataframe = pd.read_csv('flower_train.csv')
dataframe2 = pd.read_csv('flower_test.csv')
print(dataframe.isnull().sum())
print(dataframe)
dataframe[['x1','x2']]=\
dataframe[['x1','x2']].replace(0,np.NaN)
temp=dataframe[dataframe['x1'].notnull()]
temp=temp[['x1','type']].groupby(['type'])[['x1']].mean().reset_index()
dataframe.loc[(dataframe['type']=='Iris-setosa')&(dataframe['x1'].isnull()),'x1']=temp['x1'][0]
dataframe.loc[(dataframe['type']=='Iris-versicolor')&(dataframe['x1'].isnull()),'x1']=temp['x1'][1]
print("将第一列x1中的NaN变为数字后：")
print(dataframe)
temp=dataframe[dataframe['x2'].notnull()]
temp=temp[['x2','type']].groupby(['type'])[['x2']].median().reset_index()
#使用dataframe的loc函数将指定条件的height列和sex列筛选出来进行值替换操作
dataframe.loc[(dataframe['type']=='Iris-setosa')&(dataframe['x2'].isnull()),'x2']=temp['x2'][0]
dataframe.loc[(dataframe['type']=='Iris-versicolor')&(dataframe['x2'].isnull()),'x2']=temp['x2'][1]
print("将第二列x2中的NaN变为数字后：")
print(dataframe)
dataframe['type'] = np.where(dataframe['type'] == "Iris-setosa", 0, 1)
print(dataframe)
```

运行结果：

```
x1      5
x2      6
type    0
dtype: int64
     x1   x2            type
0   NaN  3.5     Iris-setosa
1   4.9  3.0     Iris-setosa
2   4.7  3.2     Iris-setosa
3   4.6  3.1     Iris-setosa
4   5.0  3.6     Iris-setosa
..  ...  ...             ...
75  5.7  NaN  Iris-versicolor
76  5.7  2.9  Iris-versicolor
77  6.2  2.9  Iris-versicolor
78  5.1  NaN  Iris-versicolor
79  5.7  2.8  Iris-versicolor

[80 rows x 3 columns]
将第一列x1中的NaN变为数字后：
          x1   x2            type
0   5.032432  3.5     Iris-setosa
1   4.900000  3.0     Iris-setosa
2   4.700000  3.2     Iris-setosa
3   4.600000  3.1     Iris-setosa
4   5.000000  3.6     Iris-setosa
..       ...  ...             ...
75  5.700000  NaN  Iris-versicolor
76  5.700000  2.9  Iris-versicolor
77  6.200000  2.9  Iris-versicolor
78  5.100000  NaN  Iris-versicolor
79  5.700000  2.8  Iris-versicolor

[80 rows x 3 columns]
将第二列x2中的NaN变为数字后：
          x1   x2            type
0   5.032432  3.5     Iris-setosa
1   4.900000  3.0     Iris-setosa
2   4.700000  3.2     Iris-setosa
3   4.600000  3.1     Iris-setosa
4   5.000000  3.6     Iris-setosa
..       ...  ...             ...
75  5.700000  2.8  Iris-versicolor
76  5.700000  2.9  Iris-versicolor
77  6.200000  2.9  Iris-versicolor
78  5.100000  2.8  Iris-versicolor
79  5.700000  2.8  Iris-versicolor

[80 rows x 3 columns]
          x1   x2  type
0   5.032432  3.5     0
1   4.900000  3.0     0
2   4.700000  3.2     0
3   4.600000  3.1     0
4   5.000000  3.6     0
..       ...  ...   ...
75  5.700000  2.8     1
76  5.700000  2.9     1
77  6.200000  2.9     1
78  5.100000  2.8     1
79  5.700000  2.8     1

[80 rows x 3 columns]
```

最后的输出是一个 80 行，3 列，元素均为数字的矩阵。

2.

把矩阵 matrix 最后一列全都设成 1。

2)在之前的线性回归实验中，我们的模型为$\hat{y} = \omega^T x + b$，为方便实验，该实验中我们将偏置量b划入模型参数中，则对应的模型变为$\hat{y} = \omega^T x$，请进行相应的转换
tips:上一次实验中的矩阵求解析解的方法中将某一列全设置为1，即将偏置量b算入模型参数中，特征值中加入一列全1的特征量

```
matrix=np.array(dataframe)
m=len(dataframe)
x1=matrix[:,0]
x2=matrix[:,1]
y=np.array(matrix[:,2])
matrix[:,2].fill(1)
print(matrix)
```

运行结果：

```
[[5.03243243 3.5        1.        ]
 [4.9        3.         1.        ]
 [4.7        3.2        1.        ]
 [4.6        3.1        1.        ]
 [5.         3.6        1.        ]
 [5.4        3.9        1.        ]
 [4.6        3.4        1.        ]
 [5.         3.4        1.        ]
 [4.4        2.9        1.        ]
 [4.9        3.1        1.        ]
 [5.4        3.7        1.        ]
 [4.8        3.4        1.        ]
 [5.03243243 3.         1.        ]
 [4.3        3.         1.        ]
 [5.8        4.         1.        ]
 [5.7        4.4        1.        ]
 [5.4        3.9        1.        ]
 [5.1        3.5        1.        ]
 [5.7        3.4        1.        ]
 [5.1        3.8        1.        ]
 [5.4        3.4        1.        ]
 [5.1        3.7        1.        ]
 [4.6        3.6        1.        ]
 [5.1        3.3        1.        ]
 [4.8        3.4        1.        ]
 [5.         3.         1.        ]
 [5.         3.4        1.        ]
 [5.2        3.5        1.        ]
 [5.2        3.4        1.        ]
 [4.7        3.2        1.        ]
 [4.8        3.1        1.        ]
 [5.03243243 3.4        1.        ]
 [5.2        4.1        1.        ]
 [5.5        4.2        1.        ]
 [4.9        3.1        1.        ]
 [5.         3.2        1.        ]
 [5.5        3.5        1.        ]
 [4.9        3.1        1.        ]
```

```
[4.9         3.1         1.          ]
[5.          3.2         1.          ]
[5.5         3.5         1.          ]
[4.9         3.1         1.          ]
[4.4         3.4         1.          ]
[5.1         3.4         1.          ]
[5.          2.          1.          ]
[5.9         3.          1.          ]
[6.          2.2         1.          ]
[5.91315789 2.9         1.          ]
[5.6         2.9         1.          ]
[6.7         3.1         1.          ]
[5.6         3.          1.          ]
[5.8         2.7         1.          ]
[6.2         2.2         1.          ]
[5.6         2.5         1.          ]
[5.9         3.2         1.          ]
[6.1         2.8         1.          ]
[6.3         2.5         1.          ]
[6.1         2.8         1.          ]
[6.4         2.9         1.          ]
[6.6         3.          1.          ]
[6.8         2.8         1.          ]
[6.7         3.          1.          ]
[6.          2.9         1.          ]
[5.7         2.8         1.          ]
[5.5         2.4         1.          ]
[5.5         2.4         1.          ]
[5.8         2.7         1.          ]
[6.          2.7         1.          ]
[5.4         3.          1.          ]
[6.          3.4         1.          ]
[6.7         3.1         1.          ]
[6.3         2.3         1.          ]
[5.6         3.          1.          ]
[5.5         2.5         1.          ]
[5.5         2.6         1.          ]
[6.1         3.          1.          ]
[5.8         2.6         1.          ]
[5.91315789 2.3         1.          ]
[5.6         2.7         1.          ]
[5.7         2.8         1.          ]
[5.7         2.9         1.          ]
[6.2         2.9         1.          ]
[5.1         2.8         1.          ]
[5.7         2.8         1.          ]]
```

3.矩阵初始化

思路如下：初始化一个 w 矩阵，包括 w1,w2,w0,再利用公式迭代，进行计算，循环 1000 次之后得出最适合的 w 矩阵。

3) 由于逻辑回归的原理是用逻辑函数把线性回归的结果(-∞,∞)映射到(0,1)所以逻辑回归适合于二分类问题。我们使用sigmoid函数$g(z)=\frac{1}{1+e^{-z}}$将把线性回归的结果从(-∞,∞)映射到(0,1)。
假设模型为线性回归模型$\hat{y}=\omega_0+\omega_1 x_1+\omega_2 x_2+\ldots+\omega_n x_n=\omega^T x$，则任意样本所对应发生的概率值函数即为$g(\hat{y})=\frac{1}{1+e^{-y}}$，这样事情发生(定义为标签为1)的概率为

$$P(y=1|x)=\frac{1}{1+e^{-\omega^T x}}$$

对应于任意一个样本(x_i,y_i),其中x_i为特征值，y_i为实际结果值,在参数ω下，该样本发生的概率为

$$P(y_i|x_i,\omega)=y_i P(y_i=1|x_i)+(1-y_i)P(y_i=0|x_i)$$

将每个样本发生概率相乘，得到似然函数:

$$\Pi_{i=1}^{m} P(y_i|x_i,\omega)$$

为了计算方便，一般取对数得到对数似然函数:

$$L(\omega)=\sum_{i=1}^{m} lnP(y_i|x_i,\omega)$$

我们总是希望出现预测正确的概率的可能性最大，即想要得到极大化似然函数对应的参数ω。这样最大化似然函数就转变为最小化似然函数的负数，取负的平均对数似然函数为损失函数,通过这样构建的损失函数

$$J(\omega)=-\frac{1}{m}\sum_{i=1}^{m} lnP(y_i|x_i,\omega)=-\frac{1}{m}\sum_{i=1}^{m} ln(y_i\frac{1}{1+e^{-\omega^T x_i}}+(1-y_i)\frac{e^{-\omega^T x_i}}{1+e^{-\omega^T x_i}})$$

手动实现梯度下降法(不使用机器学习框架，如PyTorch、TensorFlow等)来进行模型的训练。

算法步骤如下：①初始化模型参数ω的值；②在负梯度的方向上更新参数(由于该实验涉及样本数量较小，建议使用批量梯度下降)，并不断迭代这一步骤。

其中梯度的下降偏导公式为

$$\frac{\partial J}{\partial \omega_j}=\frac{1}{m}\sum_{i=1}^{m} x_{ij}(\frac{e^{\omega^T x_i}}{1+e^{\omega^T x_i}}-y_i)$$

参数更新的公式为

$$\omega_j=\omega_j-\eta\frac{\partial J}{\partial w_j}$$

其中η表示学习率，m则表示批量中的样本数量，x_{ij}代表着第i个样本的第j个特征值,y_i代表着第i个样本的真实值

代码及结果：

```
w0=5
w1=5
w2=5
n=0.3
matrixy=[w0,w1,w2]
matrixy1=np.mat(matrixy)
trans=(matrixy1.T)
print(matrixy1)
for i in range(1000):
    s=[1-1/(1+np.exp(w0+w1*x1[i]+w2*x2[i]))-y[i] for i in range (m) ]
    w0=w0-n*np.sum(s)/m
    w1=w1-n*np.sum(np.dot(x1,s))/m
    w2=w2-n*np.sum(np.dot(x2,s))/m
print(w0,w1,w2)
```

```
[[5 5 5]]
2.4864525976446514 4.226037696071281 -8.159666051971293
```

4. 误差计算

思路：

4)在模型训练完成后得到所训练的模型参数ω，在测试集上进行所训练模型的测试并使用之前所介绍的损失函数计算loss值

代码及结果：

```
#your code here------

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

epsilon = 1e-15

# 计算所有样本的预测概率
z = w0 + w1 * x1 + w2 * x2
probabilities = sigmoid(z)

# 确保概率不为0或1，防止log函数计算时出错
probabilities = np.clip(probabilities, epsilon, 1 - epsilon)

# 计算损失函数
loss_values = -y * np.log(probabilities) - (1 - y) * np.log(1 - probabilities)
loss = np.mean(loss_values)

print(f"Loss: {loss}")
```

```
Loss: 0.05727989621092957
```

计算出的 loss 值在误差范围以内，有效。

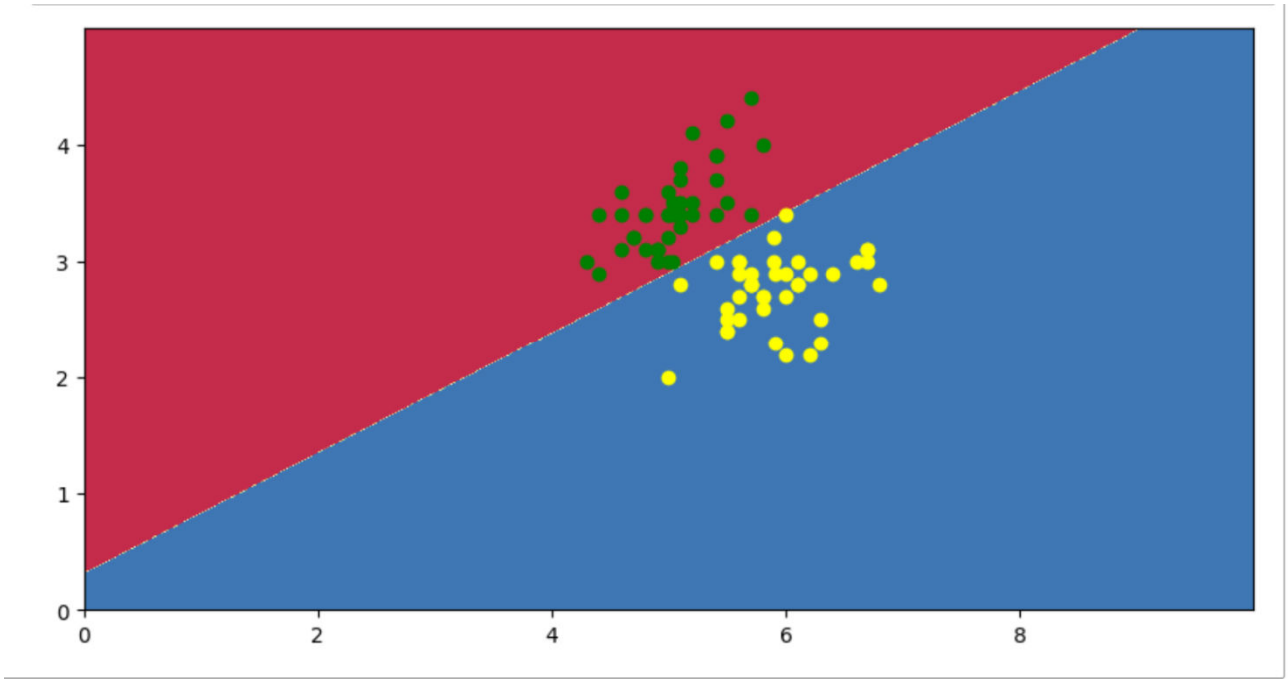
5. 绘图

5)使用训练后的逻辑回归模型对测试数据集'flower_test.csv'进行预测，输出可视化结果（比如用seaborn或者matplotlib等可视化库来画出测试数据的散点图以及训练好的模型函数图像），要求如下:
1.将所得到的逻辑回归模型所得到的决策边界绘制出来
2.测试集的所有点在同一幅图中进行绘制
3.需要给不同类别的测试点不同颜色，方便通过颜色的区别直观看到预测正确和错误的样本

代码及结果：

In [33]:

```
#确定图画边界和大小
plt.figure(figsize=(10,5))
x_min, x_max = 0,10
y_min, y_max = 0,5
#使用numpy中的meshgrid生成网格矩阵，方便进行之后的描点
boundary_x, boundary_y = np.meshgrid(np.arange(x_min, x_max, 0.01),np.arange(y_min, y_max, 0.01))
grid = np.c_[boundary_x.ravel(), boundary_y.ravel()]
#加入偏置对应的一列
e=np.ones((len(grid),1))
grid=np.c_[e,grid]
#假定下列的模型参数
w=np.array([[w0],[w1],[w2]])
#计算出网格点中每个点对应的逻辑回归预测值
z=grid.dot(w)
for i in range(len(z)):
    z[i][0]=(1/(1+np.exp(-z[i])))
    if(z[i][0]<0.5):z[i][0]=0
    else:z[i][0]=1
#转换shape以作出决策边界
z=z.reshape(boundary_x.shape)
plt.contourf(boundary_x, boundary_y, z, cmap=plt.cm.Spectral, zorder=1)
class_1=dataframe[dataframe['type']==1]
class_0=dataframe[dataframe['type']==0]
plt.scatter(class_1['x1'],class_1['x2'],c='yellow')
plt.scatter(class_0['x1'],class_0['x2'],c='green')
plt.show()
```



可见绝大多数点都被直线区分开来，说明有效。

心得体会

1. 学习率和迭代次数会对实验数据造成很大的影响，首先，如果迭代次数太多，就有可能造成过拟合，误差 w 反倒升高了（比如执行 1000 次就优于执行 10000 次）同时也要给予足够的迭代次数，否则欠拟合，w 同样很大。

2. 这里的损失函数计算有很多种方法，assignment3 给了一种，但我认为误差比较大，尤其是当概率趋近于 0,1 时，log 值会趋于无穷和 0，会造成误差，所以我 define 了一个极小值，避免造成 log 趋于负无穷的状态。又利用了 sigmoid 函数进行求解。

第二题：泰坦尼克号

题目概述：该数据集(train_titanic.csv 和 test_titanic.csv)同样为分类数据集，为泰坦尼克号的乘客信息以及最后是否生还。包括了七个特征值以及一个类别特征(即为 Survived 类型,代表是否生还),特征信息分别为 Passengerid(乘客 id)，Age(乘客年龄)，Fare(船票价格),Sex(性别)，sibsp(堂兄弟妹个数)，Parch(父母与小孩的个数)，Pclass(乘客等级)

题目思路：该数据集存在七个特征值，我们需要选择其中和生还率关联最大的几个特征值，其中很显然，乘客 ID,堂兄弟妹个数，父母与小孩个数这三个特征和生存没有明显的关

联，而乘客年龄，船票价格，乘客性别，乘客等级更加重要，同时，乘客的船票价格和船票等级上也可能存在关联关系，比如价格越高，等级越高。

选择完之后，我们要对模型进行训练，由于样本数据有限（1000 个），将其中 90%的数据用作训练集，10%用作测试集，称为十次十折。这里我在网上特意查到了 kfold 函数，学习了它的一些简单用法。利用自带的函数进行模型选择，十折交叉验证，再计算查准率和查全率，用 matplotlib 画出 roc 和 pr 曲线。

核心代码实现：

一. 准备工作：

1. 导入数据，输出训练集和测试集。

2. 导入梯度函数和十次十折函数。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
```

```
# 使用pandas库载入数据
train_data = pd.read_csv('train_titanic.csv')
test_data = pd.read_csv('test_titanic.csv')
print(train_data)
print(test_data)
#your code here------
from sklearn.model_selection import KFold
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
#定义函数，求查准率，查全率，roc曲线
import numpy as np
import matplotlib.pyplot as plt
```

运行结果：

	Passengerid	Age	Fare	Sex	sibsp	Parch	Pclass	Survived
0	1	22.0	7.2500	0	1	0	3	0
1	2	38.0	71.2833	1	1	0	1	1
2	3	26.0	7.9250	1	0	0	3	1
3	4	35.0	53.1000	1	1	0	1	1
4	5	35.0	8.0500	0	0	0	3	0
...	...	...	...	...	...	...	...	...
1004	1305	28.0	8.0500	0	0	0	3	0
1005	1306	39.0	108.9000	1	0	0	1	0
1006	1307	38.5	7.2500	0	0	0	3	0
1007	1308	28.0	8.0500	0	0	0	3	0
1008	1309	28.0	22.3583	0	1	1	3	0

[1009 rows x 8 columns]

	Passengerid	Age	Fare	Sex	sibsp	Parch	Pclass	Survived
0	471	28.0	7.2500	0	0	0	3	0
1	472	38.0	8.6625	0	0	0	3	0
2	473	33.0	27.7500	1	1	2	2	1
3	474	23.0	13.7917	1	0	0	2	1
4	475	22.0	9.8375	1	0	0	3	0
..	...	...	...	...	...	...	...	...
295	766	51.0	77.9583	1	1	0	1	1
296	767	28.0	39.6000	0	0	0	1	0
297	768	30.5	7.7500	1	0	0	3	0
298	769	28.0	24.1500	0	1	0	3	0
299	770	32.0	8.3625	0	0	0	3	0

[300 rows x 8 columns]

二.

选择最优的四个特征（'age','fare','sex','pclass'）带入模型，利用自带的 KFold 函数实现十次十折（把 X_train 和 y_train 按照 9:1 分配 train：val=9:1），计算准确分数（预测正确，则准确分数为 1，预测错误，准确分数为 0）将结果导入到列表中，再求列表的平均值，也就是求准确分数的平均值，得到平均准确率。

In [45]:
```
# 选择四个特征
features = ['Age', 'Fare', 'Sex', 'Pclass']
X_train = train_data[features].values
y_train = train_data['Survived'].values

# 使用SGD和交叉验证进行模型选择
kf = KFold(n_splits=10, shuffle=True, random_state=42)
model = SGDClassifier(loss='log_loss', max_iter=1000, tol=1e-3)

# 10次10折交叉验证
accuracies = []

for train_index, val_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]

    model.fit(X_train_fold, y_train_fold)
    predictions = model.predict(X_val_fold)
    accuracies.append(accuracy_score(y_val_fold, predictions))

print("Mean Accuracy:", np.mean(accuracies))
```

```
Mean Accuracy: 0.7433366336633662
```

三．计算查准率和查全率

利用 sklearn 当中的得分函数（如果正类，得到值为 1，反之为 0）求解查准率 p，查全率 r，代码以及运行结果：

```
#your code here------
X_test = test_data[features].values
y_test = test_data['Survived'].values

# 使用训练好的模型进行预测
y_pred = model.predict(X_test)

# 计算评估指标
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Accuracy: 0.6366666666666667
Precision: 1.0
Recall: 0.043859649122807015
```

四．画出 P-R 曲线和 ROC 曲线

在上面的第三步已经求出了查准率 p,查全率 r,把它们用曲线连接并展示出来。运行代码和运行结果

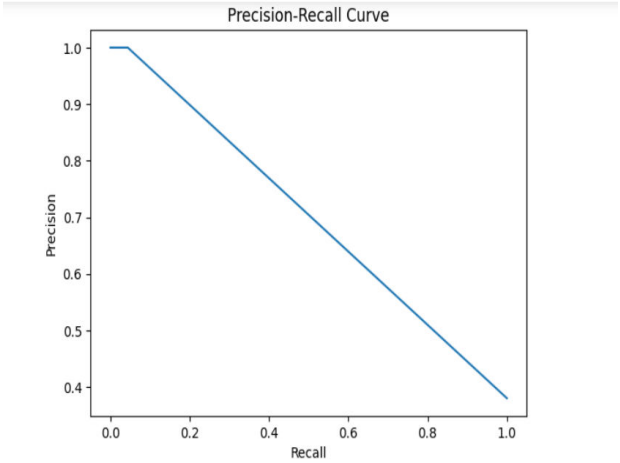
In [69]:

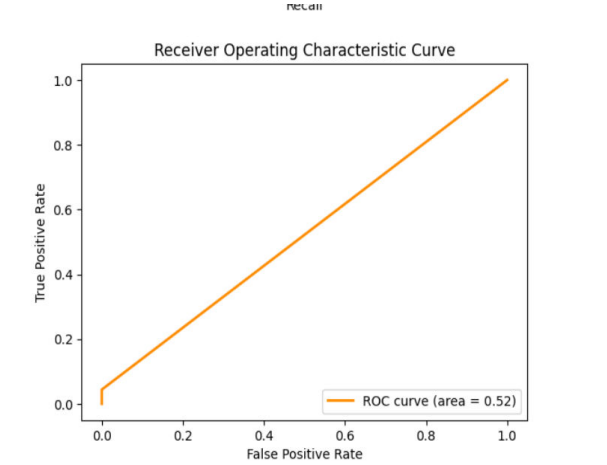
```
# P-R曲线
from sklearn.metrics import precision_recall_curve

precision, recall, _ = precision_recall_curve(y_test, y_pred)
plt.plot(recall, precision, label='P-R Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()

# ROC曲线
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve')
plt.legend(loc="lower right")
plt.show()
```





心得体会：

1.求查准率和查全率时，要知道查准率和查全率是负相关的，在一定范围内，查准率越高，查全率越低，反之亦然。就像宁可错杀一千，绝不放过一个，查全率提高了，但查准率降低，可能会滥杀无辜，反之查全率如果过低也会影响效果。如果查全率和查准率都很高或者都很低，则一定存在问题。

2.使用已有的 sklearn,kfold 函数代替自主编程（手动梯度下降），可以大大节省时间，提升效率。

3.P-R 曲线越接近右上角，效果越好，ROC 曲线越接近左上角，效果越好。

4.在优化过程中，我从网上查得可以利用特征缩放的方法，就是对同一个特征值进行等比例放大或缩小，从而使梯度下降速度变快，加快收敛速度。