



实 验 报 告

年 级 专 业： 20 级智能科学与技术

学 生 姓 名： 孙成

学 号： 20203101694

课 程 名 称： 机器学习

实 验 名 称： 线性回归与逻辑回归实验

任 课 老 师： 胡祝华

实验报告成绩	
任课教师签名	

海南大学 · 信息与通信工程学院

School of Information and Communication Engineering, Hainan University

实验室	学院 118	计算机号	B08
实验软硬件环境	1. 实验所用到的硬件环境 MacBook Pro (16-inch, 2019) 2.6GHz 六核 Intel Core i7 16 GB 2667 MHz DDR4 2. 实验所用到的软件环境 PyCharm 2021.3.2 (Professional Edition)		
实验目的或要求	1. 掌握线性回归 (linear regression) 的基本原理和实现方法, 掌握基本术语和概念: 序关系 (order)、均方差 (square error) 最小化、欧式距离 (Euclidean distance)、最小二乘法 (least square method)、参数估计 (parameter estimation)、多元线性回归 (multivariate linear regression)、广义线性回归 (generalized linear model)、对数线性回归 (log-linear regression); 2. 掌握逻辑回归 (logistic regression) 的基本原理和实现方法, 掌握基本术语和概念: 分类、Sigmoid 函数、对数几率 (log odds/logit)、极大似然法 (maximum likelihood method); 3. 熟悉 LDA 线性判别分析和多分类转二分类的方法。		
实验内容	Pycharm 仿真终端的使用介绍 1. 认识 IDE 各部分组成 2. 新建项目, 新建文件 3. 设置好解释器环境 4. 运行设置, 执行代码 线性回归实验 1. 导入 numpy 库, 并将当前项目根目录加入解释器的搜索路径中。 2. 伪造 100 个样本数据集 3. 把生成的样本数据集用散点图画出来, 同时画出真实的直线 4. 参数的求解 5. 根据训练得到的参数 w 进行可视化展示 6. 查看 loss 的变化。通过图直观的查看 loss 值是否随着迭代次数的增加趋于平稳? 注意: loss 是否趋于平稳是衡量一个模型是否已经收敛的重要指标。 7. 实验补充: 另一种尝试, 即利用求伪逆直接求闭式解。		

	<p>逻辑回归实验</p> <ol style="list-style-type: none">1. 导入实验所需的相关资源和模块2. 实验原理解释3. 与线性回归类似，模型训练的代码及封装如下所示。代码命名为 <code>logic_regression.py</code>，并放到 <code>my_models\linear_model</code> 目录下。4. 编写测试程序，命名为 <code>logic_regression_test.py</code>，并放到 <code>tests</code> 目录下。构建数据集，训练模型并进行测试。5. 与 <code>sklearn</code> 中的逻辑回归模型的对比。 <p>思考与提高</p> <ol style="list-style-type: none">1. 逻辑回归的损失函数为何不用 <code>mse</code>?
实 验 结 果	<p>1.线性回归实验</p> <pre>## 伪造 100 个样本数据集 import numpy as np import sys import os import matplotlib.pyplot as plt X = np.linspace (0, 100, 100) X = np.c_[X, np.ones (100)] w = np.asarray([3,2]) Y = X.dot(w) X= X.astype('float') Y= Y.astype('float') X[:,0]+=np.random.normal(size=(X[:, 0].shape))*3 Y = Y.reshape (100,1) print(Y) ## 结果如下: [[2. [5.03030303] [8.06060606] [11.09090909] [14.12121212] [17.15151515] [20.18181818] [23.21212121] [26.24242424] [29.27272727]</pre>

	[32.3030303]
	[35.33333333]
	[38.36363636]
	[41.39393939]
	[44.42424242]
	[47.45454545]
	[50.48484848]
	[53.51515152]
	[56.54545455]
	[59.57575758]
	[62.60606061]
	[65.63636364]
	[68.66666667]
	[71.6969697]
	[74.72727273]
	[77.75757576]
	[80.78787879]
	[83.81818182]
	[86.84848485]
	[89.87878788]
	[92.90909091]
	[95.93939394]
	[98.96969697]
	[102.]
	[105.03030303]
	[108.06060606]
	[111.09090909]
	[114.12121212]
	[117.15151515]
	[120.18181818]
	[123.21212121]
	[126.24242424]
	[129.27272727]
	[132.3030303]
	[135.33333333]
	[138.36363636]
	[141.39393939]
	[144.42424242]

	[147.45454545]
	[150.48484848]
	[153.51515152]
	[156.54545455]
	[159.57575758]
	[162.60606061]
	[165.63636364]
	[168.66666667]
	[171.6969697]
	[174.72727273]
	[177.75757576]
	[180.78787879]
	[183.81818182]
	[186.84848485]
	[189.87878788]
	[192.90909091]
	[195.93939394]
	[198.96969697]
	[202.]
	[205.03030303]
	[208.06060606]
	[211.09090909]
	[214.12121212]
	[217.15151515]
	[220.18181818]
	[223.21212121]
	[226.24242424]
	[229.27272727]
	[232.3030303]
	[235.33333333]
	[238.36363636]
	[241.39393939]
	[244.42424242]
	[247.45454545]
	[250.48484848]
	[253.51515152]
	[256.54545455]
	[259.57575758]

```
[262.60606061]  
[265.63636364]  
[268.66666667]  
[271.6969697 ]  
[274.72727273]  
[277.75757576]  
[280.78787879]  
[283.81818182]  
[286.84848485]  
[289.87878788]  
[292.90909091]  
[295.93939394]  
[298.96969697]  
[302.          ]]
```

把生成的样本数据集用散点图画出来，同时画出真实的直线

```
plt.scatter(X[:,0],Y)  
plt. plot(np.arange(0, 100).reshape((100,1)),Y, 'r')  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.show()  
## 结果如下:
```

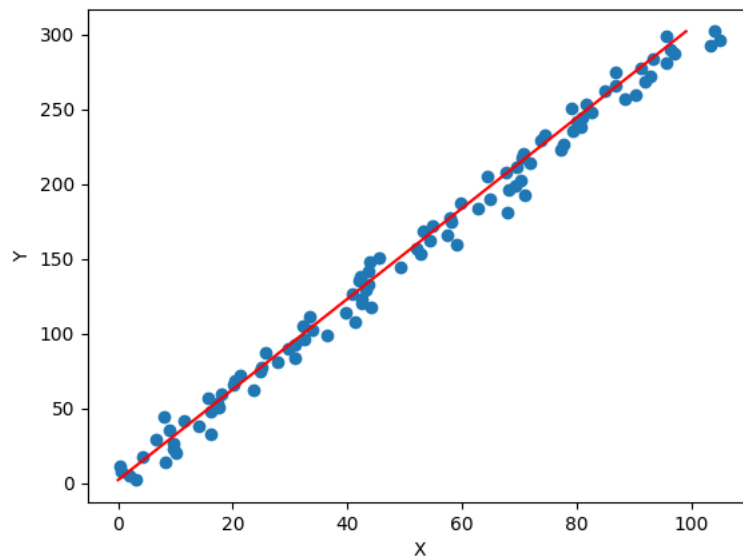


图 1 散点图

```
## 参数的求解
w=np.random.random(size=(2,1))
epochs=100
eta=0.0000001
losses=[]
for _ in range(epochs):
    dw=-2*X.T.dot(Y-X.dot(w))
    w = w - eta * dw
    losses.append((Y-X.dot(w)).T.dot(Y-X.dot(w)).reshape(-1))
print(losses)
## 结果如下
/Users/suncheng/opt/anaconda3/envs/SS/bin/python /Users/suncheng/PycharmProjects/machineLearning/
[array([1380117.30643279]), array([1200472.11819044]), array([1044355.30802675]), array([908685.3
进程已结束,退出代码0
```

图 2 loss 结果

```
## 根据训练得到的参数 w 进行可视化展示
plt.scatter (X[:,0],Y)
plt.plot(X[:, 0], X.dot (w), 'r')
plt.xlabel('x')
```

```
plt.ylabel('Y')
plt.show()
## 结果如下
```

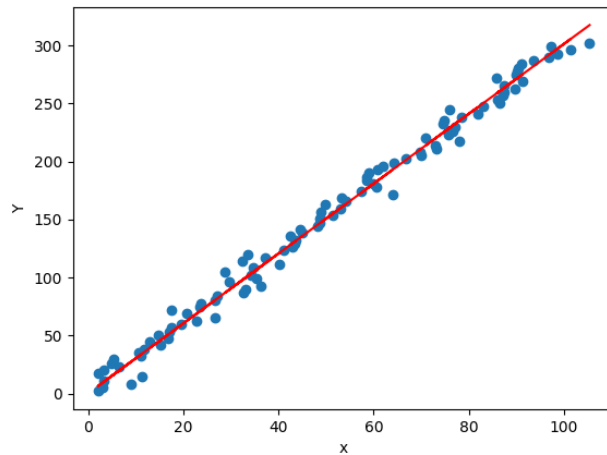


图 3 可视化结果

```
## loss 可视化
plt.plot(losses)
plt.xlabel('iterations')
plt.ylabel('loss')
plt.show()
```

```
## 结果如下
```

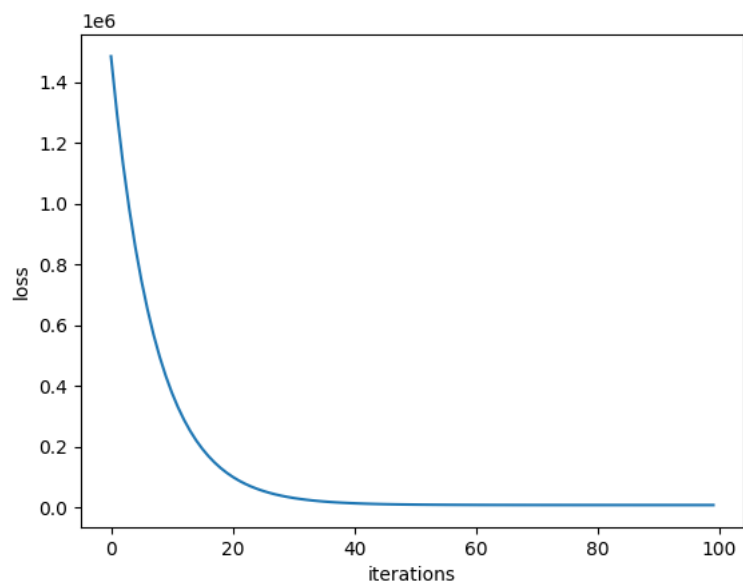



图 4 loss 可视化

```
## 利用求伪逆直接求闭式解
w=np.linalg.pinv(X).dot (Y)
plt.scatter (X[:, 0], Y)
plt.plot(X[:, 0], X. dot (w), 'r')
plt.plot(np. arange (0, 100).reshape((100, 1)), Y, c='b',
linestyle='--')
plt.xlabel ('X')
plt.ylabel ('Y')
plt.show()
```

结果如下图

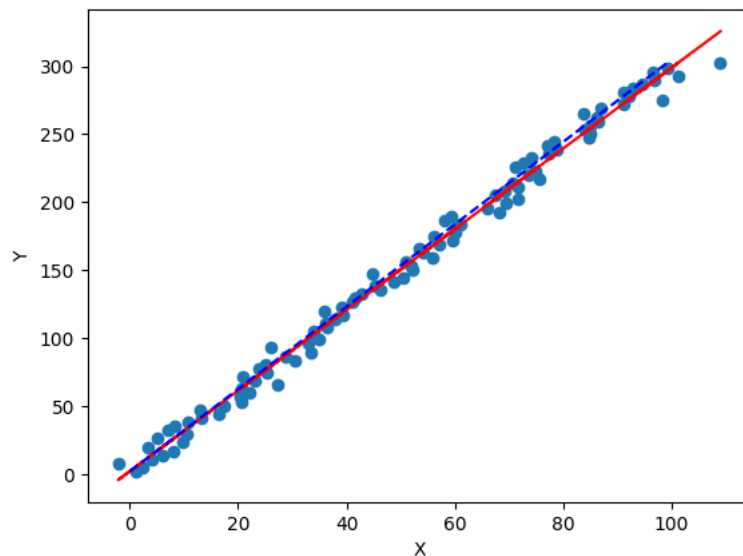


图 5 求伪例直接求闭式解

2. 基于封装模块的测试

1) 直接调用封装对象中的 SGD 方法进行训练和预测

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import numpy as np
import matplotlib.pyplot as plt

### 将根目录添加到 sys.path, 解决在命令行下执行时找不到模块的问题。
import sys
import os
curPath = os.path.abspath(os.path.dirname(__file__))
rootPath = os.path.split(curPath)[0]
sys.path.append(rootPath)

# 造伪样本
X = np.linspace(0, 100, 100)
X = np.c_[X, np.ones(100)] # 考虑到偏置 b
w = np.asarray([3, 2]) # 参数
Y = X.dot(w)
```

```
X = X.astype('float')
Y = Y.astype('float')
X[:, 0] += np.random.normal(size=(X[:, 0].shape)) * 3 # 添加
0,1 高斯噪声

Y = Y.reshape(100, 1)

from my_models.linear_model import *
#测试
lr=LinearRegression(solver='sgd')
lr.fit(X[:, :-1], Y) #[:, :-1] 表示除掉最后一列
predict=lr.predict(X[:, :-1])
#查看w
print('w', lr.get_params())
#查看标准差, 如果标准差小的话则认为真实值与预测值相符合。
print(np.std(Y-predict))

#可视化结果
lr.plot_fit_boundary(X[:, :-1], Y) # 预测的拟合直线
plt.plot(np.arange(0, 100).reshape((100, 1)), Y, c='b',
linestyle='--') #真实直线
plt.show() # 可视化显示
## 结果如下
```

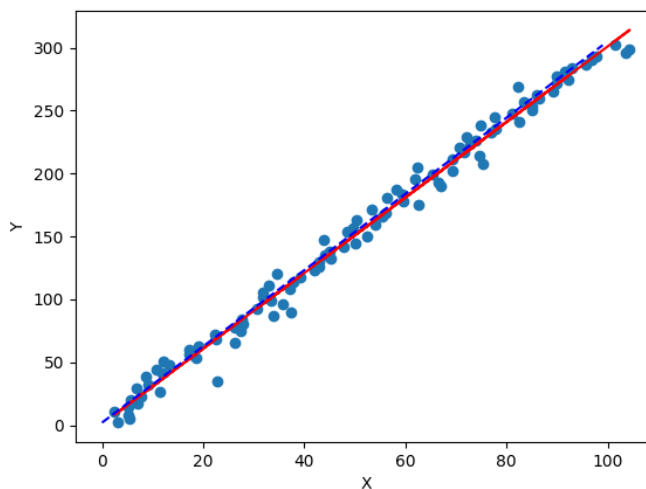


图 6 测试 solver='sgd'

2) 使用闭式求解参数的方法进行测试

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
```

```
import numpy as np
import matplotlib.pyplot as plt
```

###将根目录添加到 sys.path, 解决在命令行下执行时找不到模块的问题。

```
import sys
import os
curPath = os.path.abspath(os.path.dirname(__file__))
rootPath = os.path.split(curPath)[0]
sys.path.append(rootPath)
```

造伪样本

```
X = np.linspace(0, 100, 100)
X = np.c_[X, np.ones(100)] #考虑到偏置 b
w = np.asarray([3, 2]) #参数
Y = X.dot(w)
X = X.astype('float')
Y = Y.astype('float')
X[:, 0] += np.random.normal(size=(X[:, 0].shape)) * 3 # 添加
0,1 高斯噪声
```

```
Y = Y.reshape(100, 1)

from my_models.linear_model import *

#测试
#lr=LinearRegression(solver='sgd')
lr=LinearRegression(solver='closed form')
lr.fit(X[:, :-1], Y) #[:, :-1] 表示除掉最后一列
predict=lr.predict(X[:, :-1])
#查看 w
print('w', lr.get_params())
#查看标准差, 如果标准差小的话则认为真实值与预测值相符合。
print(np.std(Y-predict))

#可视化结果
lr.plot_fit_boundary(X[:, :-1], Y) # 预测的拟合直线
plt.plot(np.arange(0, 100).reshape((100, 1)), Y, c='b',
         linestyle='--') #真实直线
plt.show() # 可视化显示
## 结果如下
```

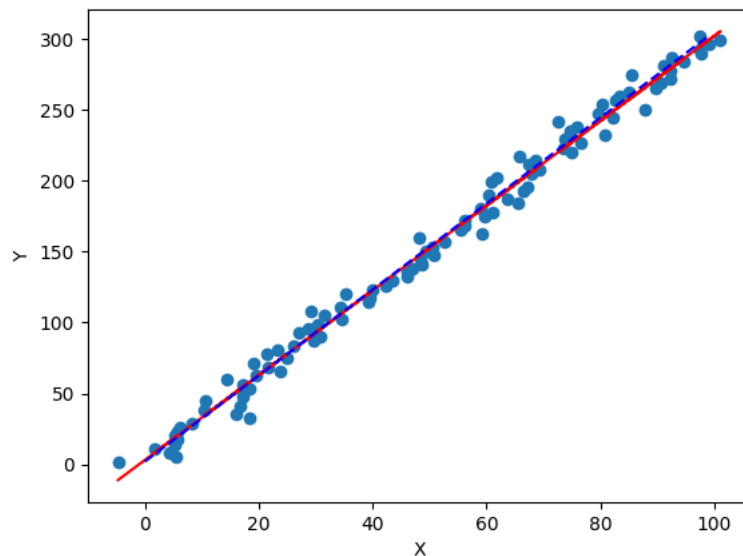


图 7 测试 solver='closed form'

3) 用 sklearn 中的 linear_model 模块中的 LinearRegression 类测试。

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import os

curPath = os.path.abspath(os.path.dirname(__file__))
rootPath = os.path.split(curPath)[0]
sys.path.append(rootPath)

# 造伪样本
X = np.linspace(0, 100, 100)
X = np.c_[X, np.ones(100)] # 考虑到偏置 b
w = np.asarray([3, 2]) # 参数
Y = X.dot(w)
X = X.astype('float')
Y = Y.astype('float')
X[:, 0] += np.random.normal(size=(X[:, 0].shape)) * 3 # 添加 0,1 高斯噪声
Y = Y.reshape(100, 1)

# 与 sklearn 对比
```

```

from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X[:, :-1], Y)
predict=lr.predict(X[:, :-1])
#查看 w,b
print('w:', lr.coef_, 'b:', lr.intercept_)
#查看标准差
print(np.std(Y-predict))
#可视化结果
plt.scatter(X[:, 0], Y)
plt.plot(X[:, 0], predict, 'r')
plt.plot(np.arange(0,100).reshape((100,1)), Y, c='b',
linestyle='--')
plt.show()
## 结果如下

```

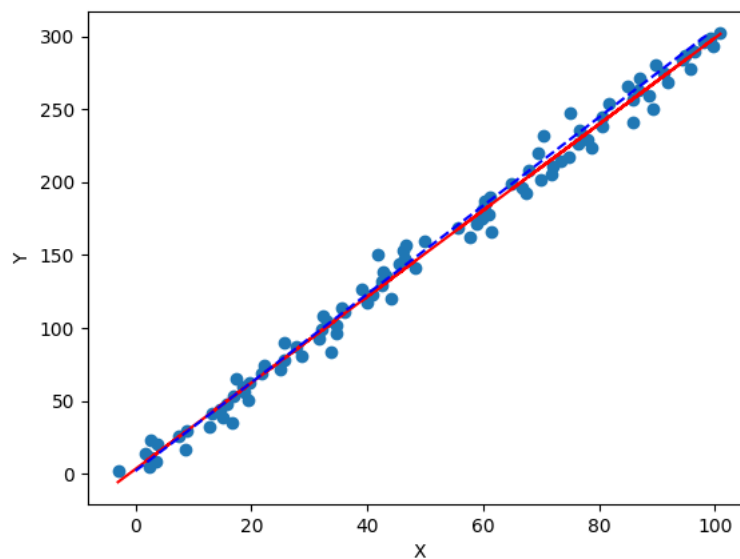


图 8 LinearRegression 类测试

3.逻辑回归实验

绘制 sigmoid 函数

```

t=np.arange(-8, 8, 0.5)
d_t=1/(1+np.exp(-t))
plt.plot (t, d_t)
plt.show()

```

结果如下

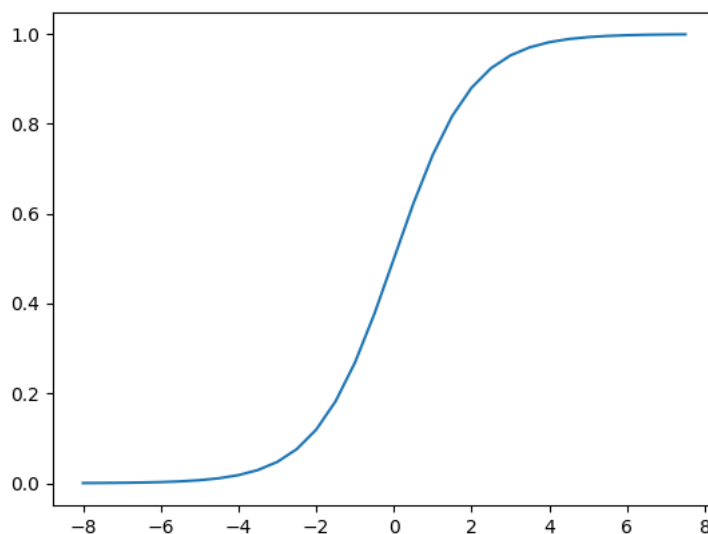


图 9 Sigmoid 函数图

将 logic regression.py,并放到 my models\ linear model 目录下

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import numpy as np
import os
os.chdir('../..') # 用于改变当前工作目录到指定的路径。
from my_models import utils
import matplotlib.pyplot as plt

class LogisticRegression(object):
    def __init__(self, fit_intercept=True, solver='sgd',
if_standard=True, epochs=10, eta=None, batch_size=16):

        self.w = None
        self.fit_intercept = fit_intercept
        self.solver = solver
        self.if_standard = if_standard
        if if_standard:
```



```

        self.feature_mean = None
        self.feature_std = None
    self.epochs = epochs
    self.eta = eta
    self.batch_size = batch_size
    # 注册 sign 函数
    self.sign_func = np.vectorize(utils.sign)
    # 记录 losses
    self.losses = []

def init_params(self, n_features):
    """
    初始化参数
    :return:
    """
    self.w = np.random.random(size=(n_features, 1))

def _fit_sgd(self, x, y):
    """
    随机梯度下降求解
    :param x:
    :param y:
    :return:
    """
    x_y = np.c_[x, y]
    count = 0
    for _ in range(self.epochs):
        np.random.shuffle(x_y)
        for index in range(x_y.shape[0] //
self.batch_size):
            count += 1
            batch_x_y = x_y[self.batch_size *
index:self.batch_size * (index + 1)]
            batch_x = batch_x_y[:, :-1]
            batch_y = batch_x_y[:, -1:]

            dw = -1 * (batch_y -
utils.sigmoid(batch_x.dot(self.w))).T.dot(batch_x) /

```

```

self.batch_size
    dw = dw.T

    self.w = self.w - self.eta * dw

    # 计算 losses
    cost = -1 * np.sum(
        np.multiply(y,
np.log(utils.sigmoid(x.dot(self.w))) + np.multiply(1 - y,
np.log(
        1 - utils.sigmoid(x.dot(self.w))))
    self.losses.append(cost)

def fit(self, x, y):
    """
    :param x: ndarray 格式数据: m x n
    :param y: ndarray 格式数据: m x 1
    :return:
    """
    y = y.reshape(x.shape[0], 1)
    # 是否归一化 feature
    if self.if_standard:
        self.feature_mean = np.mean(x, axis=0)
        self.feature_std = np.std(x, axis=0) + 1e-8
        x = (x - self.feature_mean) / self.feature_std
    # 是否训练 bias
    if self.fit_intercept:
        x = np.c_[x, np.ones_like(y)]
    # 初始化参数
    self.init_params(x.shape[1])
    # 更新 eta
    if self.eta is None:
        self.eta = self.batch_size / np.sqrt(x.shape[0])

    if self.solver == 'sgd':
        self._fit_sgd(x, y)

def get_params(self):

```

```

"""
输出原始的系数
:return: w,b
"""

if self.fit_intercept:
    w = self.w[:-1]
    b = self.w[-1]
else:
    w = self.w
    b = 0
if self.if_standard:
    w = w / self.feature_std.reshape(-1, 1)
    b = b - w.T.dot(self.feature_mean.reshape(-1, 1))
return w.reshape(-1), b

def predict_proba(self, x):
    """
    预测为y=1 的概率
    :param x:ndarray 格式数据: m x n
    :return: m x 1
    """
    if self.if_standard:
        x = (x - self.feature_mean) / self.feature_std
    if self.fit_intercept:
        x = np.c_[x, np.ones(x.shape[0])]
    return utils.sigmoid(x.dot(self.w))

def predict(self, x):
    """
    预测类别, 默认大于0.5 的为1, 小于0.5 的为0
    :param x:
    :return:
    """
    proba = self.predict_proba(x)
    return (proba > 0.5).astype(int)

def plot_decision_boundary(self, x, y):
    """

```

绘制前两个维度的决策边界

```
:param x:
:param y:
:return:
"""

y = y.reshape(-1)
weights, bias = self.get_params()
w1 = weights[0]
w2 = weights[1]
bias = bias[0][0]
x1 = np.arange(np.min(x), np.max(x), 0.1)
x2 = -w1 / w2 * x1 - bias / w2
plt.scatter(x[:, 0], x[:, 1], c=y, s=50)
plt.plot(x1, x2, 'r')
plt.show()

def plot_losses(self):
    plt.plot(range(0, len(self.losses)), self.losses)
    plt.show()
```

编写测试程序，命名为 logic regression test.py

```
#!/usr/bin/python
```

```
# -*- coding: UTF-8 -*-
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

###将根目录添加到 sys.path，解决在命令行下执行时找不到模块的问题。

```
import sys
```

```
import os
```

```
curPath = os.path.abspath(os.path.dirname(__file__))
```

```
rootPath = os.path.split(curPath)[0]
```

```
sys.path.append(rootPath)
```

#构造伪分类数据并可视化

```
from sklearn.datasets import make_classification
```

```
from my_models.linear_model import LogisticRegression
```

```
data, target=make_classification(n_samples=100,  
n_features=2,n_classes=2,n_informative=1,n_redundant=0,n_rep  
eated=0,n_clusters_per_class=1)  
print(data.shape)  
print(target.shape)  
plt.scatter(data[:, 0], data[:, 1], c=target, s=50)  
plt.show()  
## 结果如下
```

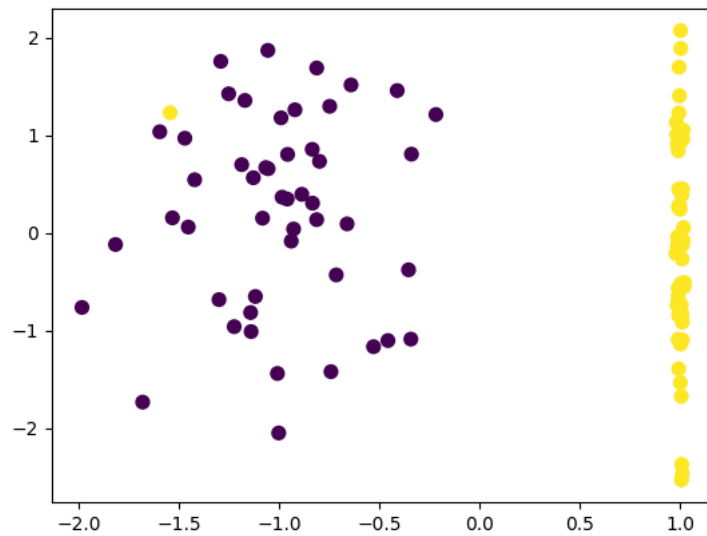


图 10 数据可视化

```
#训练模型  
lr = LogisticRegressionO  
lr.fit(data, target)  
#查看 loss 值的变化，交叉熵损失  
lr.plot lossesO  
结果如下：
```

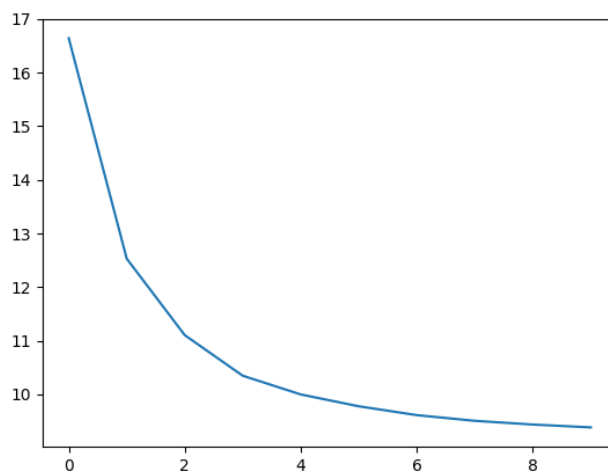


图 11 loss 损失

#绘制决策边界

```
lr.plot_decision_boundary(data, target)
```

#计算 F1

```
from sklearn.metrics import f1_score  
f1_score(target, lr.predict(data))
```

结果如下;

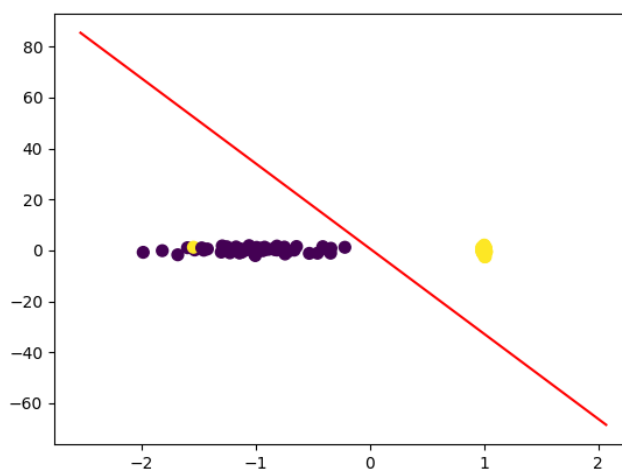


图 12 分类结果

```
## 与 sklearn 中的逻辑回归模型的对比。
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
data,target=make_classification(n_samples=100,
n_features=2,n_classes=2,n_informative=1,n_redundant=0,n_repeated=0,n_clusters_per_class=1)
lr.fit(data, target)
w1=lr.coef_[0][0]
w2=lr.coef_[0][1]
bias=lr.intercept_[0]
print(w1)
print(w2)
print(bias)
#画决策边界
x1=np.arange(np.min(data),np.max(data),0.1)
x2=-w1/w2*x1-bias/w2
plt.scatter(data[:, 0], data[:, 1], c=target,s=50)
plt.plot(x1 ,x2,'r')
plt.show()
#计算 F1
f1_score(target,lr.predict( data))
## 结果如下
```

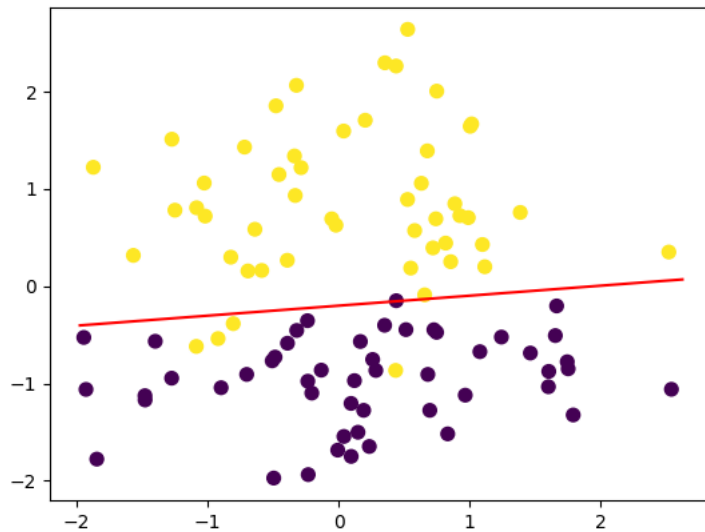


图 13 结果图

```

/Users/suncheng/opt/anaconda3/envs/SS/bin/python
-0.33268150468684354
3.2376437036034127
0.6500785678402703

```

图 14 w_1 & w_2 & bias 结果

逻辑回归的损失函数为何不用 mse?

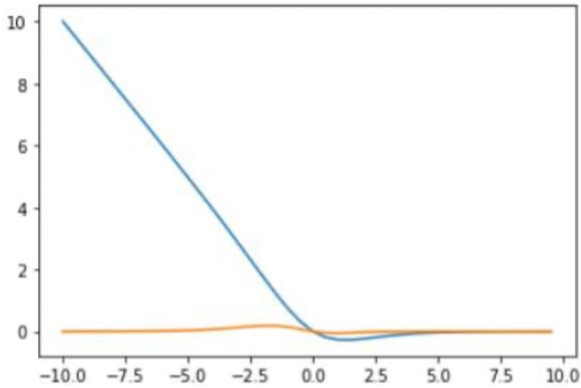
在前面线性回归模型中使用 mse 作为损失函数，并取得不错的效果，而逻辑回归中使用的是交叉熵损失函数；这是因为如果使用 mse 作为损失函数，梯度下降将会比较困难，在 $f(x')$ 与 y' 相差较大或者较小时梯度值都会很小

我们绘图对比一下两者梯度变化的情况，假设在 $y=1$, $x \in (-10, 10)$, $w=1$, $b=0$ 的情况下：

```

import matplotlib.pyplot as plt
import numpy as np
y=1
x0=np.arange(-10, 10, 0.5)
#交叉熵
x1=np.multiply (utils.sigmoid(x0)-y, x0)
#mse

```


	<pre>x2=np.multiply(utils.sigmoid(x0)-y, utils.sigmoid(x0)) x2=np.multiply(x2, 1-utils.sigmoid(x0)) x2=np.multiply (x2, x0) plt.plot (x0,x1) plt.plot (x0,x2) plt.show()</pre> <p>结果如下</p>  <p>图 15 结果图</p> <p>可见在错分的那一部分 ($x < 0$), mse 的梯度值基本停留在 0 附近, 而交叉熵会让越“错”情况具有越大的梯度值。</p>
心得体会	<p>通过本次试验, 我学会了给 python 配置环境, 以及如何定义 my_models, 在验证上述代码的正确性后, 我收获匪浅, 后续将继续深入学习, 力争在大学时间学习更多的机器学习的知识。</p>