机器学习讲课

作业3

Exercise 3.14

通过选读链接 https://scikitlearn.com.cn/ 1.1广义线性模型的部分内容，解决下面问题。

使用regress_data1的数据，分别采用批量梯度下降法、L_1正则化，L_2正则化 实现线性回归模型。

原理

公式4-1：线性回归模型预测

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

公式4-3：线性回归模型的MSE成本函数

$$\mathrm{MSE} = (\boldsymbol{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}^{(i)} - y^{(i)})^2$$

公式4-5：成本函数的偏导数

$$\frac{\partial}{\partial\theta_j}\mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}^{(i)} - y^{(i)})x_j^{(i)}$$

公式4-6：成本函数的梯度向量

$$\nabla_{\theta}\mathrm{MSE}(\boldsymbol{\theta})=\begin{pmatrix}\frac{\partial}{\partial\theta_0}\mathrm{MSE}(\boldsymbol{\theta})\\ \frac{\partial}{\partial\theta_1}\mathrm{MSE}(\boldsymbol{\theta})\\ \vdots\\ \frac{\partial}{\partial\theta_n}\mathrm{MSE}(\boldsymbol{\theta})\end{pmatrix}=\frac{2}{m}\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{\theta}-\boldsymbol{y})$$

公式4-7：梯度下降步骤

$$\boldsymbol{\theta}^{(\text{下一步})}=\boldsymbol{\theta}-\eta\nabla_{\theta}\mathrm{MSE}(\boldsymbol{\theta})$$

本题采用

$$\frac{\partial}{\partial\theta_j}\mathrm{MSE}(\boldsymbol{\theta})=\frac{2}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}^{(i)}-y^{(i)})x_j^{(i)}$$

代码逐段讲解

导入库

```
# 导入岭回归线性模型
from sklearn.linear_model import Ridge
# 导入Lasso回归线性模型
from sklearn.linear_model import Lasso
# 导入Pandas库
import pandas as pd
# 导入Matplotlib库
import matplotlib.pyplot as plt
# 导入Numpy库
import numpy as np
```

- Matplotlib 数据可视化库，提供大量专业数据制作工具

读取csv文件数据

```
# 1.读取regress_data1.csv文件数据，将第一列数据作为x，第二列数据作为y 第一行为名称
# 采用相对路径读取
data = pd.read_csv('regress_data1.csv')
dataMat = np.array(data)
x = dataMat[:, 0:1]
y = dataMat[:, 1]
```

- `pd.read_csv() # 可以读取纯文本文件`
- Pandas 数据结构和数据分析库。包含高级数据结构和类SQL语句，让数据处理变得快速、简单

  ```
  data = pd.read_csv('regress_data1.csv')
  ```

  此处 data 可看作数据库中一张表
- Numpy 科学计算基础库，提供高效的N维数组和向量运算

  ```
  dataMat = np.array(data)
  ```

  根据pandas对象生成数组
  - 生成的数组舍弃了第一行不符合格式的文本即[人口 收益]

使用批量梯度下降法获取线性回归模型

```
# 2. 使用批量梯度下降法获取线性回归模型
data.insert(0, 'ones', 1)
cols = data.shape[1]
x1 = data.iloc[:, 0:cols - 1]
y1 = data.iloc[:, cols - 1:cols]
```

- `data.insert(0, 'ones', 1)`

  pandas表data在第0列插入列索引(名称为one)，列的值为1

  ```
      ones       人口        收益
  0      1   6.1101  17.59200
  1      1   5.5277   9.13020
  2      1   8.5186  13.66200
  3      1   7.0032  11.85400
  4      1   5.8598   6.82330
  ..   ...      ...       ...
  92     1   5.8707   7.20290
  93     1   5.3054   1.98690
  ```

- `cols = data.shape[1]`

  显示data表列数

- ```
  x1 = data.iloc[:, 0:cols - 1]
  y1 = data.iloc[:, cols - 1:cols]
  ```

  切割data表，此时x1只包含原data表第1，2列数据

  y1包含原data表第3列数据

  - 表x1

    ```
    ones        人口
    0      1   6.1101
    1      1   5.5277
    2      1   8.5186
    3      1   7.0032
    4      1   5.8598
    ..   ...      ...
    92     1   5.8707
    93     1   5.3054
    94     1   8.2934
    95     1  13.3940
    96     1   5.4369

    [97 rows x 2 columns]
    ```

  - 表y1

    ```
            收益
    0   17.59200
    1    9.13020
    2   13.66200
    3   11.85400
    4    6.82330
    ..       ...
    92   7.20290
    93   1.98690
    94   0.14454
    95   9.05510
    96   0.61705

    [97 rows x 1 columns]
    ```

- ```
  x1 = np.matrix(x1.values)
  y1 = np.matrix(y1.values)
  ```

  将表x1，y1的数值部分转换为矩阵

  x1为第一列全为1的矩阵

- ```
  # 特征权重向量初始化
  theta = np.random.randn(2, 1)
  # 超参数 eta 设置 (学习率)
  eta = 0.01
  # 代表迭代次数
  n_iterations = 1000
  # m 代表 训练集样本数
  m = len(x1)
  ```
  ```

成本函数、成本函数偏导数及梯度下降(特征权重转移方程)参数设定

- ```
  # 不断迭代的特征向量矩阵
  temp = np.matrix(np.zeros(theta.shape))

  for iteration in range(n_iterations):
      error = x1 * theta.T - y1
      for theta_num in range(theta.shape[1]):
          term = np.multiply(error, x1[:, j])
          temp[0, j] = theta[0, j] - ((2 * eta)/m)*np.sum(term)
      theta = temp
  ```

  - ```
    for iteration in range(n_iterations):
    ```

    模型迭代次数 1000

  - ```
    error = x1 * theta.T - y1
    ```

    当前特征权重下，各个样本预测与样本标签误差

  - ```
    for theta_num in range(theta.shape[1])
    ```

    梯度下降，更新梯度权重向量中各个特征权重

    theta_num指代当前特征权重的内部子权重

    theta.shape表明权重向量的权重参数个数,shape函数指出矩阵第1维度个数(即列数)

  - ```
    term = np.multiply(error, x1[:, j])
    ```

    代表 $(\theta^{\mathrm{T}}\boldsymbol{x}^{(i)}-y^{(i)})x_j^{(i)}$

    求出各个样本下，预测实际误差与当前特征向量参数的乘积组成的向量

  - ```
    term[0, j] = theta[0, j] - ((2 * eta)/m)*np.sum(term)
    ```

    代表 $\frac{2}{m}\sum_{i=1}^{m}$

    和 $\theta^{(\text{下一步})}=\theta-\eta\nabla_{\theta}\mathrm{MSE}(\theta)$

    单单更新当前特征向量中的某(第j个)特征参数

使用Lasso实现L1正则化

```
# 设置L1正则化超参数
reg_l1 = Lasso(alpha=0.1)
reg_l1.fit(x,y)
```

公式4-10：Lasso回归成本函数

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i|$$

s使用Ridge实现L2正则化

```
reg_l2 = Ridge(alpha=0.1)
reg_l2.fit(x,y)
```

公式4-8：岭回归成本函数

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

绘制图表

Matplotlib数据可视化库

```
pyplot.figure(figsize=(10, 10))
```

figure可以看作指定一个容纳坐标轴、图形、文字、标签的容器(图形实例)

figsize=() 指定figure的宽和高

```
plt.scatter(x, y, c = 'b', label = 'data')
plt.plot(x, theta[0, 0] + theta[0, 1] * x, c = 'r', label = 'Batch gradient
descent')
plt.plot(x, reg_l1.predict(x), c = 'g', labels='L1 regularization')
plt.plot(x, reg_l2.predict(x), c = 'y', labels='L2 regularization')
# 获取label图例信息
plt.legend()
# 显示图形
plt.show()
```

- plt.scatter(x,y,c,label) 绘制散点图

  x,y输入点列的数组

  c 点的颜色

  label 标签

- plt.plot(x,y,c,label) 绘制折线图

  同上

- reg_l1.predict(x) 模型预测值y帽（Lasso对象.predict())
- reg_l2.predict(x) 模型预测值y帽（Ridge对象.predict())

代码

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


# 1. 读取regress_data1.csv文件数据，将第一列数据作为x，第二列数据作为y 第一行为名称
data = pd.read_csv('regress_data1.csv')
dataMat = np.array(data)
x = dataMat[:, 0:1]
y = dataMat[:, 1]

# 2. 使用批量梯度下降法获取线性回归模型
data.insert(0, 'ones', 1)
cols = data.shape[1]
x1 = data.iloc[:, 0:cols - 1]
y1 = data.iloc[:, cols - 1:cols]
x1 = np.matrix(x1.values)
y1 = np.matrix(y1.values)

# 特征权重向量初始化矩阵 1行2列
theta = np.matrix(np.array([0, 0]))
# 超参数 eta 设置 (学习率)
eta = 0.01
# 代表迭代次数
n_iterations = 100
# m 代表 训练集样本数
m = len(x1)
```

```
temp = np.matrix(np.zeros(theta.shape))

for iteration in range(n_iterations):
    error = x1 * theta.T - y1
    for theta_num in range(theta.shape[1]):
        term = np.multiply(error, x1[:, theta_num])
        temp[0, theta_num] = theta[0, theta_num] - ((2 * eta)/m)*np.sum(term)
    theta = temp

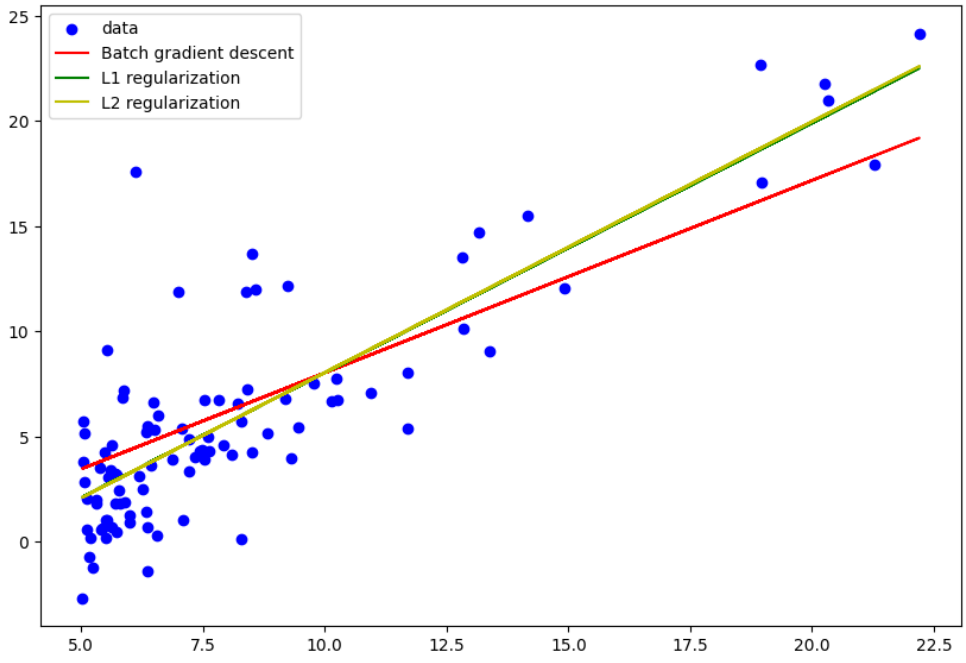
print(theta)

# 3. 使用Lasso实现L1正则化
# 设置L1正则化超参数
reg_l1 = Lasso(alpha=0.1)
reg_l1.fit(x,y)

# 4. 使用Ridge实现L2正则化
reg_l2 = Ridge(alpha=0.1)
reg_l2.fit(x,y)

# 5. 画出原始数据散点图和拟合的直线图
plt.figure(figsize=(10, 10))
plt.scatter(x, y, c = 'b', label = 'data')
plt.plot(x, theta[0, 0] + theta[0, 1] * x, c = 'r', label = 'Batch gradient
descent')
plt.plot(x, reg_l1.predict(x), c = 'g', label = 'L1 regularization')
plt.plot(x, reg_l2.predict(x), c = 'y', label = 'L2 regularization')
plt.legend()
plt.show()
```

结果:



作业5

Exercise 5.4

使用scikit-learn库，运用逻辑回归方法实现下面文件(ex2data1)里的分类任务

原理

逻辑回归（Logistic回归，也称为Logit回归）被广泛用于估算一个实例属于某个特定类别的概率。（比如，这封电子邮件属于垃圾邮件的概率是多少？）如果预估概率超过50%，则模型预测该实例属于该类别（称为正类，标记为“1”），反之，则预测不是（称为负类，标记为“0”）。这样它就成了一个二元分类器。

公式4-13：逻辑回归模型的估计概率（向量化形式）

$$\hat{p} = h_{\theta}(\boldsymbol{x}) = \sigma(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta})$$

逻辑记为σ（•），是一个sigmoid函数（即S型函数），输出一个介于0和1之间的数字。其定义如公式4-14和图4-21所示。

公式4-14：逻辑函数

$$\sigma(t) = \frac{1}{1+\exp(-t)}$$

- Sigmoid函数图形

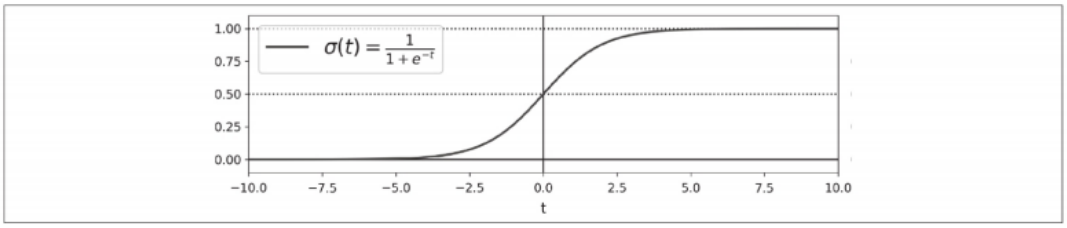


图4-21：逻辑函数

公式4-15：逻辑回归模型预测

$$\hat{y} = \begin{cases} 0, & \text{如果} \hat{p} < 0.5 \\ 1, & \text{如果} \hat{p} \geqslant 0.5 \end{cases}$$

注意，当t<0时，σ（t）<0.5；当t≥0时，σ（t）≥0.5。所以如果$x^T \theta$是正类，逻辑回归模型预测结果是1，如果是负类，则预测为0。

- 逻辑回归成本函数
- 逻辑回归成本函数偏导数---用于梯度下降优化

代码

逐段

导入库、读取文件

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

path = 'ex2data1.txt'
data = pd.read_csv(path, header=None, names=['data1','data2','Accpted'])
print(data.head())
```

- 回顾

  ```
  data = pd.read_csv(path, header=None, names=['data1','data2','Accpted'])
  ```

  header 和 names

  这两个功能相辅相成，header 用来指定列名，例如header =0，则指定第一行为列名；若header =1 则指定第二行为列名；有时，我们的数据里没有列名，只有数据，这时候就需要names=[], 来指定列名；

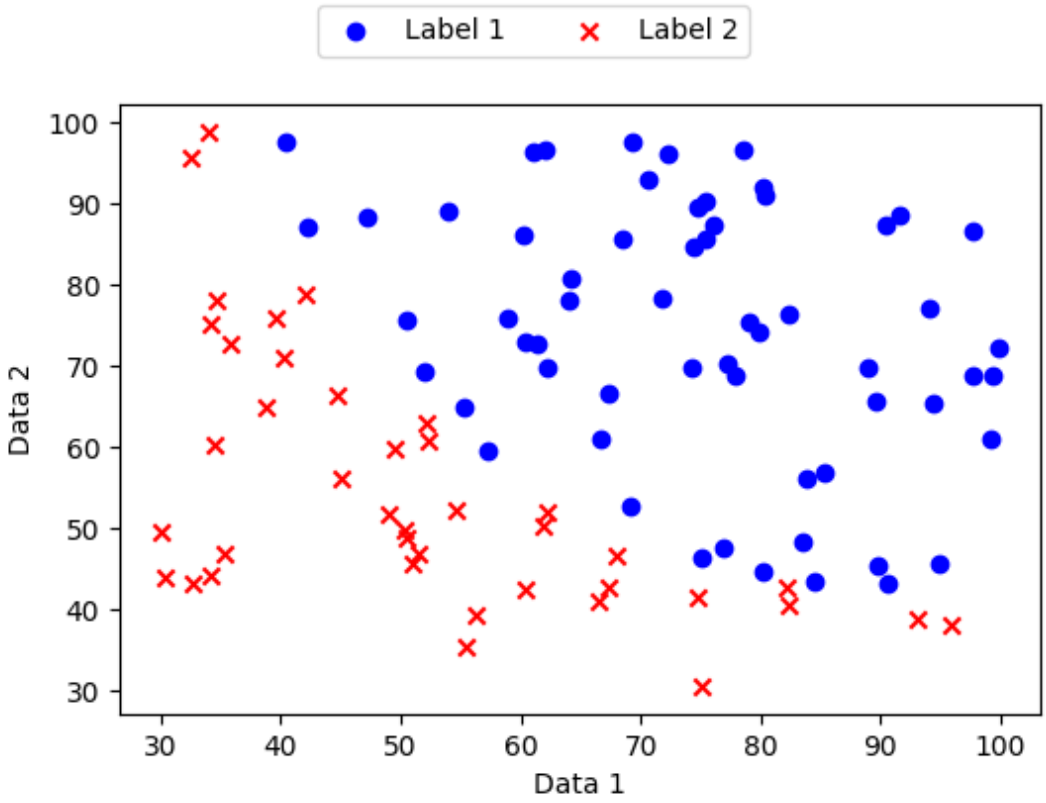
- 数据展示

```
      data1      data2  Accpted
0  34.623660  78.024693        0
1  30.286711  43.894998        0
2  35.847409  72.902198        0
3  60.182599  86.308552        1
4  79.032736  75.344376        1

Process finished with exit code 0
```

绘制散点图

```
# 绘制散点图
# 1
positive = data[data['Accpted'].isin([1])]
# 1
negative = data[data['Accpted'].isin([0])]
# 设置图例的长度
fig, ax = plt.subplots(figsize=(6, 5))
# 设置散点样式
ax.scatter(positive['data1'],positive['data2'], c='b', label='Label 1')
ax.scatter(negative['data1'],negative['data2'], c='r', marker='x', label='Label
2')
# 设置图例显示在图的上方
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width, box.height*0.8])
ax.legend(loc='center left', bbox_to_anchor=(0.2, 1.12), ncol=2)
# 设置坐标轴
ax.set_xlabel('Data 1')
ax.set_ylabel('Data 2')
plt.show()
```



线性回归

- 定义函数取得样本X，以及对应标签y纯数据

```
# 获取纯粹数据 定义一个获取原始数据的函数
def get_Xy(data):
    data.insert(0, 'one', 1)
    X = data.iloc[:,0:-1]
    X = X.values
    y_ = data.iloc[:,-1]
    y = y_.values.reshape(len(y_),1)
    return X,y
```

- 定义Sigmoid函数和cost function

```
    y_ = data.iloc[:,-1]
    y = y_.values.reshape(len(y_),1)
    return X,y

# 定义一个Sigmoid函数和cost function
# np.exp()函数是求e^x的值的函数
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def costFunction(X, y, theta):
    A = sigmoid(X@theta)
    first = y * np.log(A)
    second = (1-y) * np.log(1-A)
```

```
    return -np.sum(first+second)/len(X)
```

- ```
  A = sigmoid(X@theta)
  ```

$$x^{\mathrm{T}}\theta$$

- ```
  first = y * np.log(A)
  second = (1-y) * np.log(1-A)
  return -np.sum(first+second)/len(X)
  ```

$$J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{p}^{(i)})+(1-y^{(i)})\log(1-\hat{p}^{(i)})]$$

优化 梯度下降

```
# 实现梯度下降算法的向量化，给定固定学习率0.004和迭代参数 20000
def gradientDescent(X, y, theta, iters, alpha):
    m = len(X)
    costs = []
    for i range(iters):
        h = sigmoid(X@theta)
        theta = theta - (alpha/m)*X.T@(h-y)
        cost = costFunction(X, y, theta)
        cost.append(cost)
        if i % 1000 == 0:
            print(cost)
    return costs,theta

alpha = 0.004
iters = 20000
costs, final_theta = gradientDescent(X, y, theta, iters, alpha)
```

- ```
  h = sigmoid(X@theta)
  theta = theta - (alpha/m)*X.T@(h-y)
  ```

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(\sigma(\theta^{\mathrm{T}}x^{(i)})-y^{(i)})x_j^{(i)}$$
  ```

绘图

```
# 绘制决策边界
x = np.linspace(2,100,100)
coff11 = float(final_theta[0][0])
coff21 = float(final_theta[1][0])
coff22 = float(final_theta[2][0])
coff1 = coff11/coff22 * -1
coff2 = (coff21 / coff22) * -1
f = coff1 + coff2 * x
fig, ax = plt.subplots()
ax.scatter(data[data['Accpted']==0]['data1'],data[data['Accpted']==0]
['data2'],c='r',marker='x',label='y=0')
ax.scatter(data[data['Accpted']==1]['data1'],data[data['Accpted']==1]
['data2'],c='r',marker='o',label='y=1')
ax.legend()
ax.plot(x, f, c='g')
plt.show()
```

代码

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

path = 'ex2data1.txt'
data = pd.read_csv(path, header=None, names=['data1','data2','Accpted'])
# print(data.head())

# # 绘制散点图
# # 1
# positive = data[data['Accpted'].isin([1])]
# # 1
# negative = data[data['Accpted'].isin([0])]
# # 设置图例的长度
# fig, ax = plt.subplots(figsize=(6, 5))
# ax.scatter(positive['data1'],positive['data2'], c='b', label='Label 1')
# ax.scatter(negative['data1'],negative['data2'], c='r', marker='x', label='Label
2')
# #设置图例显示在图的上方
# box = ax.get_position()
# ax.set_position([box.x0, box.y0, box.width, box.height*0.8])
# ax.legend(loc='center left', bbox_to_anchor=(0.2, 1.12), ncol=2)
# #设置坐标轴
# ax.set_xlabel('Data 1')
# ax.set_ylabel('Data 2')
# # plt.show()

# 获取纯粹数据 定义一个获取原始数据的函数
def get_Xy(data):
    data.insert(0, 'one', 1)
    X = data.iloc[:,0:-1]
    X = X.values
    y_ = data.iloc[:,-1]
    y = y_.values.reshape(len(y_),1)
    return np.matrix(X),np.matrix(y)
```

```
# 定义一个Sigmoid函数和cost function
# np.exp()函数是求e^x的值的函数
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# 实现梯度下降算法的向量化，给定固定学习率0.004和迭代参数 200000
def gradientDescent(X, y, theta, iters, alpha):
    m = len(X)
    costs = []
    for i in range(iters):
        h = sigmoid(X@theta)
        theta = theta - (alpha/m)*X.T@(h-y)
    return theta

alpha = 0.004
iters = 200000
X, y = get_Xy(data)
# X(m*n), X是m行n列的矩阵,theta是n*1的向量
theta = np.zeros(data.shape[1]-1)
theta = np.matrix(theta).reshape(3,1)
print(X.shape)
print(y.shape)
print(theta.shape)

# costs, final_theta = gradientDescent(X, y, theta, iters, alpha)
final_theta = gradientDescent(X, y, theta, iters, alpha)

print(final_theta)

# 绘制决策边界
x = np.linspace(2,100,100)
coff11 = float(final_theta[0][0])
coff21 = float(final_theta[1][0])
coff22 = float(final_theta[2][0])
coff1 = coff11/coff22 * -1
coff2 = (coff21 / coff22) * -1
f = coff1 + coff2 * x
fig, ax = plt.subplots()
ax.scatter(data[data['Accpted']==0]['data1'],data[data['Accpted']==0]
['data2'],c='r',marker='x',label='y=0')
ax.scatter(data[data['Accpted']==1]['data1'],data[data['Accpted']==1]
['data2'],c='r',marker='o',label='y=1')
ax.legend()
ax.plot(x, f, c='g')
plt.show()
```

作业6

Exercise 6.17（实践题）

在MNIST数据集上训练SVM分类器。由于SVM分类器是个二元分类 器，所以你需要使用一对多来为10个数字进行分类。你可能还需要使 用小型验证集来调整超参数以加快进度。最后看看达到的准确率是多少？

代码逐段

导入库

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_openml
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier
from sklearn.utils import shuffle
import numpy as np
```

加载并清洗数据

```
# 加载并清洗数据
mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False)
X, y = mnist["data"], mnist["target"].astype(np.uint8)
X, y = shuffle(X, y, random_state=42)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
(70000, 784)
[8 4 8 ... 1 0 0]
(70000,)

Process finished with exit code 0
```

数据预处理

```
# 数据正则化
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

- Scaler.fit_transform是一种机器学习中常用的预处理方式，它可以对数据进行归一化处理，使得数据处理过后的均值为0，方差为1。这种处理方式有助于提高机器学习模型的准确度和迭代速度。
- **Scaler.fit_transform的原理**

Scaler.fit_transform的原理是将数据按照特定的方式进行处理，将数据的均值转换为0，方差转换为1。具体来说，Scaler.fit_transform分为两个步骤：

（1）在fit步骤中，Scaler会计算出数据集的均值和标准差。

（2）在transform步骤中，Scaler会对数据集中的每一个值进行归一化处理。

transform操作会使用均值和标准差来归一化每一个数据点，具体的公式如下：

X_scaled = (X - mean) / std

其中，X_scaled是处理过后的数据，X是原始数据，mean是原始数据的均值，std是原始数据的标准差。

选取部分数据，使用小型验证集来调整超参数以加快进度

```
# 选取部分数据，使用小型验证集来调整超参数以加快进度
sub_size = 5000
X_subset, _, y_subset, _ = train_test_split(X, y, train_size=sub_size,
random_state=42)


param_grid = {
    'C' : [0.1, 1, 10]
    'gamma' : [0.001, 0.01, 0.1, 1]
    'kernel' : ['linear', 'rbf']
}

clf = SVC(random_state=42)
grid_search = GridSearchCV(clf, param_grid, cv=3, verbose=2, n_jobs=-1)
grid_search.fit(X_subset, y_subset)
```

- 1

```
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=None,
train_size=None, random_state=None, shuffle=True, stratify=None)
```

参数说明：

参数	含义
X	待划分的样本特征集
y	待划分的样本标签
test_size	默认值为none，值为0.0-1.0时表示测试集占总样本比例；值为整数时表示测试集数量
train_size	默认值为none，值为0.0-1.0时表示训练集占总样本比例；值为整数时表示训练集数量
random_state	默认值none, 随机数种子（下面详细介绍）
shuffle	默认值True, 表示是否在拆分前打乱数据, 若为False则stratify必须置为none
stratify	默认值none，如果不是none，则以分层方式拆分数据，并将其用作类标签

返回值说明：

名称	含义
X_train	训练数据集
X_test	测试数据集
y_train	训练标签集
y_test	测试标签集

- 2

```
clf = SVC(random_state=42)
```

随机森林分类器

- 3

```
grid_search = GridSearchCV(clf, param_grid, cv=3, verbose=2, n_jobs=-1)
```

网格搜索调参GridSearchCV()

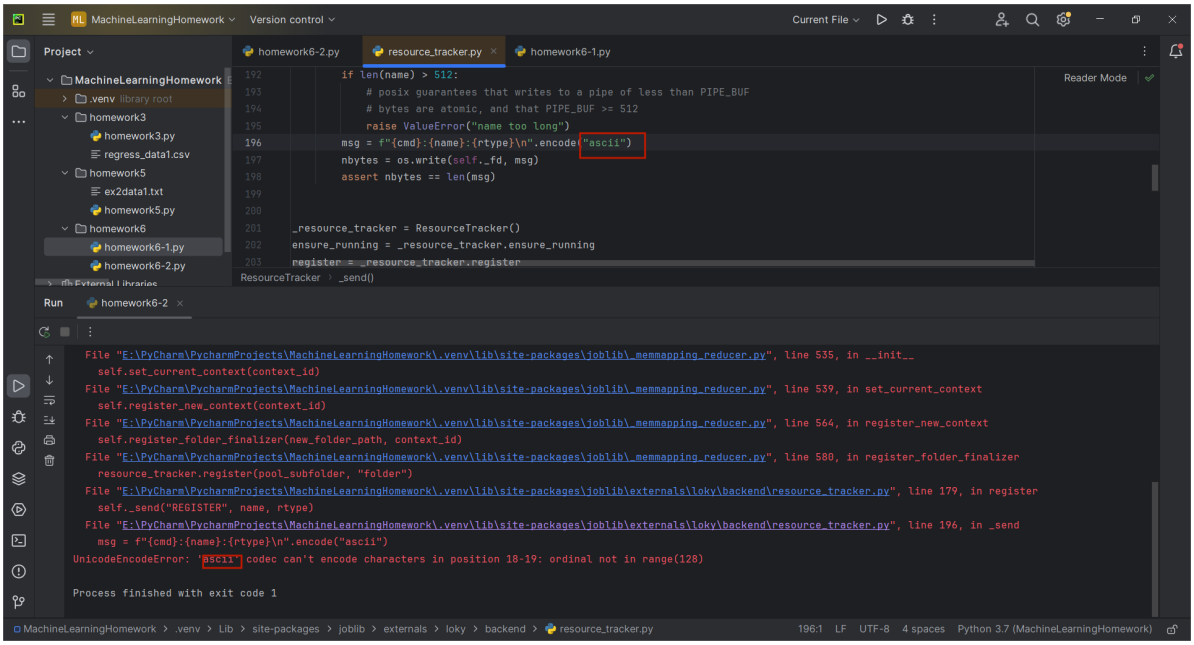
参数

GridSearchCV方法的参数：

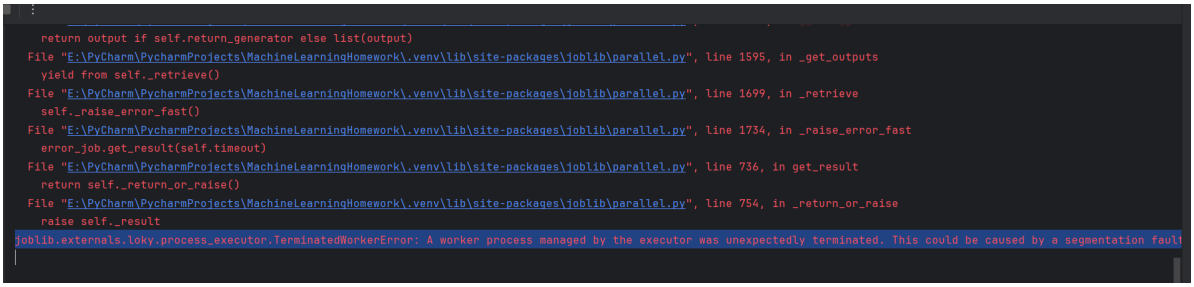
estimator	需要搜索的模型
param_grid	需要搜索的参数空间，可以是一个字典或一个列表，每个元素都是一个参数字典
cv	交叉验证的折数
scoring	模型评估指标
n_jobs	并行处理的进程数
verbose	详细程度（0、1、2）默认为0，即不输出如何过程

其中scoring的参数除accuracy，其他方法在多分类需要进行调整，它的几种参数如下

1. accuracy（准确率）：计算预测值和真实值相同的样本数除以总样本数来衡量分类模型性能。
2. precision（精确度）：在所有预测为正例的样本中，实际为正例的样本数量占比。
3. recall（召回率）：在所有实际为正例的样本中，预测为正例的样本数量占比。
4. f1：通过计算准确率和召回率的加权平均来衡量分类模型性能。
5. roc_auc（Area Under the Receiver Operating Characteristic curve）：通过计算正例样本的真阳性率和负例样本的假阳性率来衡量分类模型性能。



将这几行代码中的encode('ascii')改为encode('utf-8')即可，改完后程序正常运行。



joblic版本过高PyCharm不支持

```
pip install joblib==1.2.0
```

- 解出最优参数

```
[CV] END .....................C=10, gamma=0.01, kernel=rbf; total time=  55.5s
[CV] END .....................C=10, gamma=0.01, kernel=rbf; total time=  55.3s
[CV] END ......................C=10, gamma=0.1, kernel=rbf; total time=  50.5s
[CV] END ......................C=10, gamma=0.1, kernel=rbf; total time=  50.9s
[CV] END ......................C=10, gamma=0.1, kernel=rbf; total time=  48.1s
[CV] END ........................C=10, gamma=1, kernel=rbf; total time=  34.8s
[CV] END ........................C=10, gamma=1, kernel=rbf; total time=  35.5s
[CV] END ........................C=10, gamma=1, kernel=rbf; total time=  34.2s
Best hyperparameters: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}

Process finished with exit code 0
```

模型训练

```
# 数据量过大，sklearn只使用CPU，选取数据集中12000个数据作为训练-测试数据
X_train, y_train = X[:10000], y[:10000]
X_text, y_test = X[10000:12000], y[10000:12000]

clf = OneVsRestClassifier(SVC(kernel='rbf', C=10, gamma=0.001,
probability=False, random_state=42))
clf.fit(X_train, y_train)

y_pred = clf.predict(X_text)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on test set:{accuracy:.4f}")
```

- `OneVsRestClassfier()`

  多分类器OneVsRestClassfier

  使用OvR可以更好的获取每一个类别的相关信息

```
E:\PyCharm\PycharmProjects\MachineL
Accuracy on test set:0.9605

Process finished with exit code 0
```

代码

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_openml
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier
from sklearn.utils import shuffle
import numpy as np

# 加载并清洗数据
mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False)
X, y = mnist["data"], mnist["target"].astype(np.uint8)
X, y = shuffle(X, y, random_state=42)

# print(X)
# print(X.shape)
# print(y)
# print(y.shape)

# 数据正则化
scaler = StandardScaler()
X = scaler.fit_transform(X)

# # 选取部分数据，使用小型验证集来调整超参数以加快进度
# sub_size = 5000
# X_subset, _, y_subset, _ = train_test_split(X, y, train_size=sub_size,
random_state=42)
#
# param_grid = {
#     'C' : [0.1, 1, 10],
#     'gamma' : [0.001, 0.01, 0.1, 1],
#     'kernel' : ['linear', 'rbf']
# }
#
# clf = SVC(random_state=42)
# grid_search = GridSearchCV(clf, param_grid, cv=3, verbose=2, n_jobs=-1)
# grid_search.fit(X_subset, y_subset)
#
# print("Best hyperparameters:", grid_search.best_params_)

# Best hyperparameters: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
# 数据量过大，sklearn只使用CPU，选取数据集中12000个数据作为训练-测试数据
X_train, y_train = X[:10000], y[:10000]
X_text, y_test = X[10000:12000], y[10000:12000]

clf = OneVsRestClassifier(SVC(kernel='rbf', C=10, gamma=0.001, probability=False,
random_state=42))
clf.fit(X_train, y_train)

y_pred = clf.predict(X_text)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on test set:{accuracy:.4f}")
```

```
E:\PyCharm\PycharmProjects\MachineL
Accuracy on test set:0.9605

Process finished with exit code 0
```

作业7

Exercise 7.10

利用CART决策树方法，根据职业和年龄来预测月薪

职业	年龄	月薪
程序员	22	20000
程序员	23	26000
程序员	29	30000
教师	23	12000
教师	25	14000

原理

回归决策树

训练样本集为：$D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

样本的某个特征$\mathbf{x}^{(j)}$的取值b作为切分点，则训练子集划分为：

$R_1(j,b) = \{\mathbf{x}|\mathbf{x}^{(j)} \leq b\}, R_2(j,b) = \{\mathbf{x}|\mathbf{x}^{(j)} > b\}$

记 $\hat{y}_l = \frac{1}{N_l}\sum_{\mathbf{x}\in R_l} y$, N_l表示R_l区域的样本数。

决策树f在这两个区域上整体误差平方和为：

$$\sum_{\mathbf{x}\in R_1}(y-\hat{y}_1)^2 + \sum_{\mathbf{x}\in R_2}(y-\hat{y}_2)^2$$

- 分裂特征选择

样本x：[1, 0, 22], [1, 0, 23], [1, 0, 29], [0, 1, 23], [0, 1, 25] （注：我们对职业特征进行了one-hot升维）

样本y：[20000, 26000, 30000, 12000, 14000]

分裂规则：找到使得按照feature<=阈值，和feature>阈值分成的两个分枝。

以职业为切分点：

程序员均值c1=(20000+26000+30000)/3=25333.33　　教师均值c2=13000

以职业平方误差=(20000-25333.33)^2+(26000-25333.33)^2+(30000-25333.33)^2 + (12000-13000)^2+(14000-13000)^2=51 666 666+2 000 000 = 53 666 666

以年龄为划分：

以年龄22切分点：

c左-均值 = 20000　　c右-均值= (26000+30000+12000+14000)/4=20500

以年龄-22的平方误差= (20000-20000)^2 +(26000-20500)^2+(30000-20500)^2+(12000-20500)^2+(14000-20500)^2= 235 000 000

以年龄23切分点：

c左-均值 = (20000+26000+12000)/3=19333.33　　c右-均值=(30000+14000)/2=22000

以年龄-23的平方误差= (20000-19333.33)^2+(26000-19333.33)^2+(30000-22000)^2+(12000-22000)^2+(14000-22000)^2=272 888 937.77

以年龄25切分点：

c左-均值 = (20000+26000+12000+16000)/4=18500　　c右-均值=30000

以年龄-25的平方误差= (20000-18500)^2+(26000-18500)^2+(30000-18500)^2+(12000-18500)^2+(30000-30000)^2=233 000 000

同理可得每个年龄的划分情况

	职业_程序员	职业_教师	年龄_22	年龄_23	年龄_25
平方误差	53 666 666	53 666 666	235 000 000	272 888 937.77	233 000 000

故第一次分裂选择职业。左子树为程序员的三个样本，柚子树为教师的两个样本

代码

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

list = [[1, 0, 22], [1, 0, 23], [1, 0, 29], [0, 1, 23], [0, 1, 25]]
x = np.array(list)
y = np.array([20000, 26000, 30000, 12000, 14000])
model = DecisionTreeRegressor(max_depth=1)
model.fit(x, y)
print(model.tree_.value)  # 各节点的输出值
print(model.predict([[1, 0, 24]]))
```

总结：

1.CART回归树的损失函数平方误差（SE）

2.CART回归树的某叶子节点的输出值为该节点所有样本的均值。

预测结果

```
[[[20400.        ]]

 [[25333.33333333]]

 [[13000.        ]]]
[25333.33333333]

Process finished with exit code 0
```