

桂林电子科技大学 2022-2023 学年第 2 学期

机器学习实验报告

实验名称	实验一：线性回归与决策树
学院	计算机与信息安全学院
专业	计算机科学与技术
姓名	倪培洋
学号	2000300222
实验日期	2023 年 3 月 30 日

评语：

成绩：_____ 指导教师签名：_____

一. 实验目的

1. 掌握线性回归算法和 ID3 决策树算法的原理；
2. 学会线性回归算法和 ID3 决策树算法的实现和使用方法。

二. 实验内容

本次实验为第一次实验，要求完成本次实验所有内容。具体实验内容如下：

1. 一元线性回归模型实验

(1) 假设 `line-ext.csv` 是对变量 y 随变量 x 的变化情况的统计数据集。请根据教材的公式 (3.7, 3.8)，使用 Python 语言编程计算线性回归模型的系数，建立一个线性回归模型。

要求如下：

- 1) 计算出回归系数，输出模型表达式；绘制散点图和回归直线，输出均方误差。

参考代码：

- i) 数据预览

```
# 导入第三方模块
import statsmodels.api as sm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn import linear_model

# 导入数据集
income = pd.read_csv(r'line-ext.csv')
# 绘制散点图
sns.lmplot(x='YearsExperience', y='Salary', data=income, ci=None)
# 显示图形
plt.show()
```

上述代码的结果截图为：

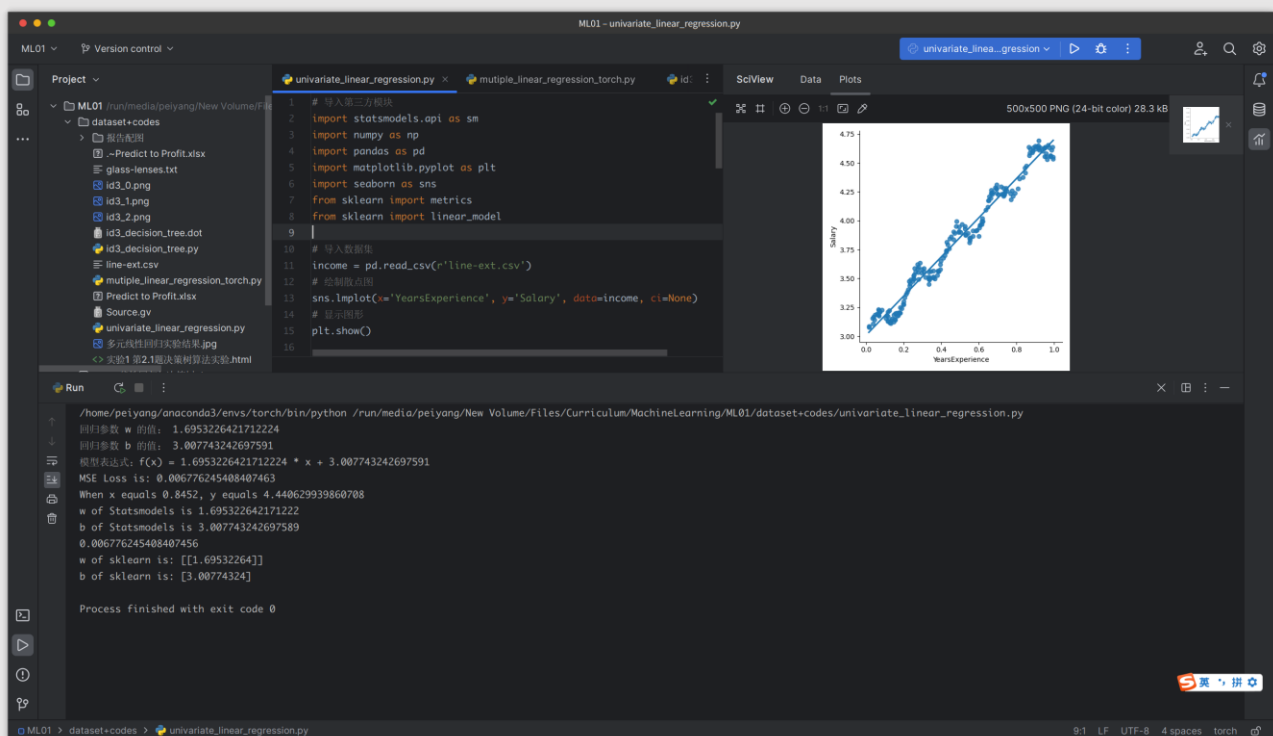


图 1

ii) 简单线性回归模型的参数 w, b 求解

```

# 样本量
n = income.shape[0]
inputs = income.YearsExperience
labels = income.Salary
# 计算自变量、因变量、自变量平方、自变量与因变量乘积的和
sum_x = income.YearsExperience.sum()
sum_y = income.Salary.sum()
sum_x2 = income.YearsExperience.pow(2).sum()
xy = income.YearsExperience * income.Salary
sum_xy = xy.sum()
# 根据公式计算回归模型的参数
w = (sum_xy - sum_x * sum_y / n) / (sum_x2 - sum_x ** 2 / n)
b = income.Salary.mean() - w * income.YearsExperience.mean()
# 打印出计算结果
print('回归参数 w 的值: ', w)
print('回归参数 b 的值: ', b)
print('模型表达式: f(x) = {0} * x + {1}'.format(w, b))

```

补全代码:

```

# 打印均方误差
predicted_y = income.YearsExperience * w + b
test_y = income.Salary
mse_loss = metrics.mean_squared_error(test_y, predicted_y)
print("MSE Loss is: {0}".format(mse_loss))

```

2) 请给出当自变量 $x=0.8452$ 时, 因变量 y 的预测值。

补全代码:

```

# 请给出当自变量 x =0.8452 时, 因变量 y 的预测值
x = 0.8452
y = x * w + b
print("When x equals 0.8452, y equals {0}".format(y))

```

给出以上代码 (第 “ii)” 部分和 “2)” 部分) 的结果截图:

```

28 w = (sum_xy - sum_x * sum_y / n) / (sum_x2 - sum_x ** 2 / n)
29 b = income.Salary.mean() - w * income.YearsExperience.mean()
30 # 打印出计算结果
31 print('回归参数 w 的值: ', w)
32 print('回归参数 b 的值: ', b)
33 print('模型表达式: f(x) = {0} * x + {1}'.format(w, b))
34 # 打印均方差
35 predicted_y = income.YearsExperience * w + b
36 test_y = income.Salary
37 mse_loss = metrics.mean_squared_error(test_y, predicted_y)
38 print("MSE Loss is: {0}".format(mse_loss))
39 |
40 # 给出当自变量 x = 0.8452 时, 因变量 y 的预测值
41 x = 0.8452
42 y = x * w + b
43 print("When x equals 0.8452, y equals {0}".format(y))

```

```

/home/peiyang/anaconda3/envs/torch/bin/python /run/media/peiyang/New Volume/Files/Curriculum/MachineLearning/ML01/dataset-codes/univariate_linear_regression.py
回归参数 w 的值: 1.6953226421712224
回归参数 b 的值: 3.007743242697591
模型表达式: f(x) = 1.6953226421712224 * x + 3.007743242697591
MSE Loss is: 0.006776245408407463
When x equals 0.8452, y equals 4.440629939860708
w of Statsmodels is 1.695322642171222
b of Statsmodels is 3.007743242697589
0.006776245408407456
w of sklearn is: [[1.69532264]]
b of sklearn is: [3.00774324]

Process finished with exit code 0

```

图 2

由图可知，程序输出：

When x equals 0.8452, y equals 4.440629939860708

即当自变量为 0.8452 时，预测的 y 等于 4.4406。

(2) 对于上面的数据集 line-ext.csv，可以使用第三方模块 statsmodels 中的函数 ols() 来计算线性回归模型的系数，建立线性回归模型，并验证上面计算结果及预测结果。也可以使用第三方模块 sklearn 模块中的类 LinearRegression 来完成这个任务，请分别给出 Python 代码。

参考代码：

1) 调用第三方模块 statsmodels 计算

```

# 请给出代码
# 导入第三方模块
import statsmodels.api as sm

# 利用收入数据集，构建回归模型
fit = sm.formula.ols("Salary ~ YearsExperience", data=income).fit()
# 返回模型的参数值
fit.params

```

2) 调用第三方模块 sklearn 模块中的类 LinearRegression 计算

补全代码:

```

# Implementation of Scikit-learn
model = linear_model.LinearRegression()
inputs = np.array(inputs)
labels = np.array(labels)
inputs = np.reshape(inputs, (-1, 1))
labels = np.reshape(labels, (-1, 1))
# print(inputs, labels)
model.fit(inputs, labels)
mse_loss_hat = metrics.mean_squared_error(model.predict(inputs), labels)
print(mse_loss_hat)
print("w of sklearn is: {}".format(model.coef_))
print("b of sklearn is: {}".format(model.intercept_))

```

运行结果截图:

```

/home/peiyang/anaconda3/envs/torch/bin/python /run/media/peiyang/New Volume/Files/Current/MachineLearning/ML01/dataset+codes/univariate_linear_regression.py
回归参数 w 的值: 1.6953226421712224
回归参数 b 的值: 3.007743242697591
模型表达式: f(x) = 1.6953226421712224 * x + 3.007743242697591
MSE Loss is: 0.006776245408407456
When x equals 0.8452, y equals 4.448629939868708
w of Statsmodels is 1.6953226421712224
b of Statsmodels is 3.007743242697591
0.006776245408407456
w of sklearn is: [[1.69532264]]
b of sklearn is: [3.00774324]

Process finished with exit code 0

```

图 3

由图可知, 程序输出

0.006776245408407456**w of sklearn is: [[1.69532264]]****b of sklearn is: [3.00774324]**

使用 sklearn 模型得到的均方误差与第一小问得到的均方误差近似,且模型的参数也近似。

2. 决策树算法实验

(1) 隐形眼镜数据集 `glass-lenses.txt` 是著名的数据集。它包含了很多患者眼部状况的观察条件以及医生推荐的隐形眼镜类型。

数据集的属性信息如下:

-- 4 Attributes

1. age of the patient: (1) young, (2) pre-presbyopic (for short, pre), (3) presbyopic
2. spectacle prescription: (1) myope, (2) hypermetrope (for short, hyper)
3. astigmatic: (1) no, (2) yes
4. tear production rate: (1) reduced, (2) normal

-- 3 Classes

- 1 : the patient should be fitted with hard contact lenses,
- 2 : the patient should be fitted with soft contact lenses,
- 3 : the patient should not be fitted with contact lenses.

请使用 Python 语言建立决策树模型 ID3, 划分 25%的数据集作为测试数据集, 并尝试进行后剪枝操作。要求 :

1) 使用 Python 语言建立决策树模型 ID3, 划分 25%的数据集作为测试数据集。使用 Graphviz 工具, 将此决策树绘制出来。此小题代码由如下, 请给出运行结果。

运行结果截图:

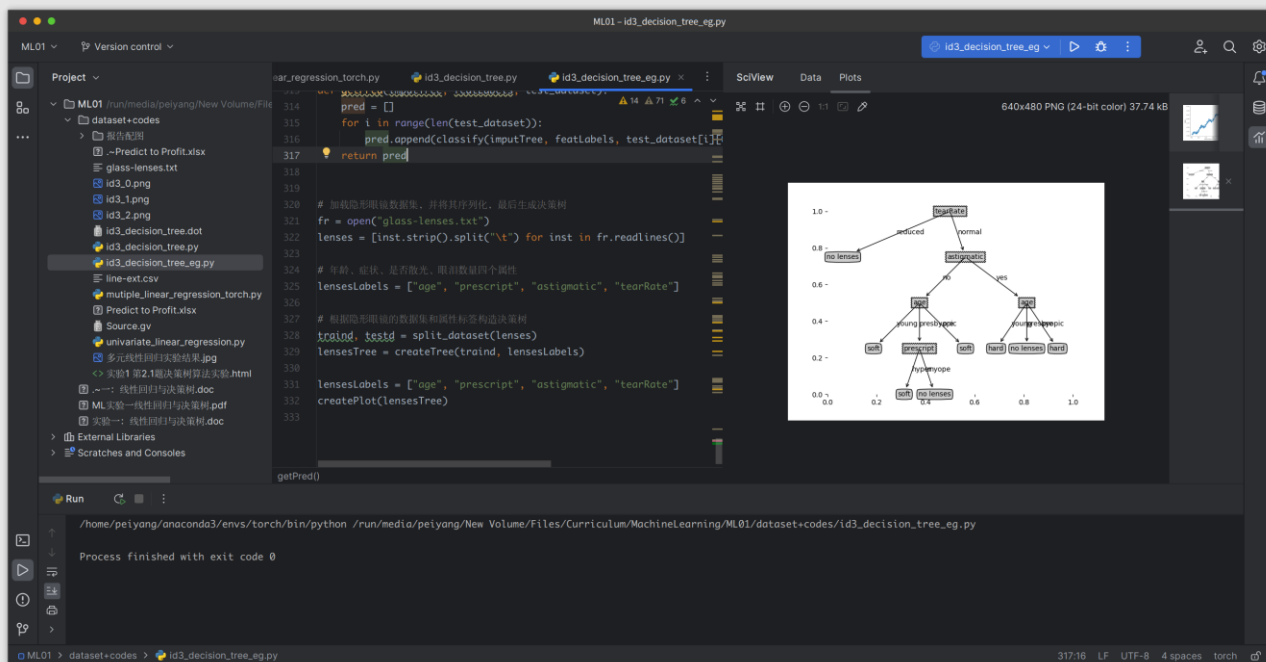


图 4

2) 请使用测试数据集进行测试, 输出测试准确率。

补全代码:

(2) 对于数据集 `glass-lenses.txt`, 使用 `sklearn` 模块中的类 `DecisionTreeClassifier` (其中, `criterion='entropy'`) 来建立决策树模型 ID3, 重复 2 (1) 中的操作, 验证上面计算结果。

1) 数据预览

```
# 导入第三方包
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import ensemble, metrics, model_selection, tree

# 读入数据
fr = open("glass-lenses.txt")
lenses = [inst.strip().split("\t") for inst in fr.readlines()]
lensesLabels = ["age", "prescript", "astigmatic", "tearRate", "type"]
lens = pd.DataFrame.from_records(lenses, columns=lensesLabels)
lens
```

2) 数据预处理

```
# 哑变量处理
dummy = pd.get_dummies(lens[["age", "prescript", "astigmatic", "tearRate"]])
# 水平合并数据集和哑变量的数据集
lens = pd.concat([lens, dummy], axis=1)
# 删除原始的 age, prescript, astigmatic 和 tearRate 变量
lens.drop(["age", "prescript", "astigmatic", "tearRate"], inplace=True,
axis=1)
lens.head()
```

3) 将数据集拆分为训练集和测试集, 且测试集的比例为 25%

```
X_train, X_test, y_train, y_test =
model_selection.train_test_split(lens.loc[:, 'age_pre': 'tearRate_reduced'],
lens.type, test_size=0.25, random_state=1234)
```

4) 构建分类决策树, 请给出代码

补全代码:

决策树为:

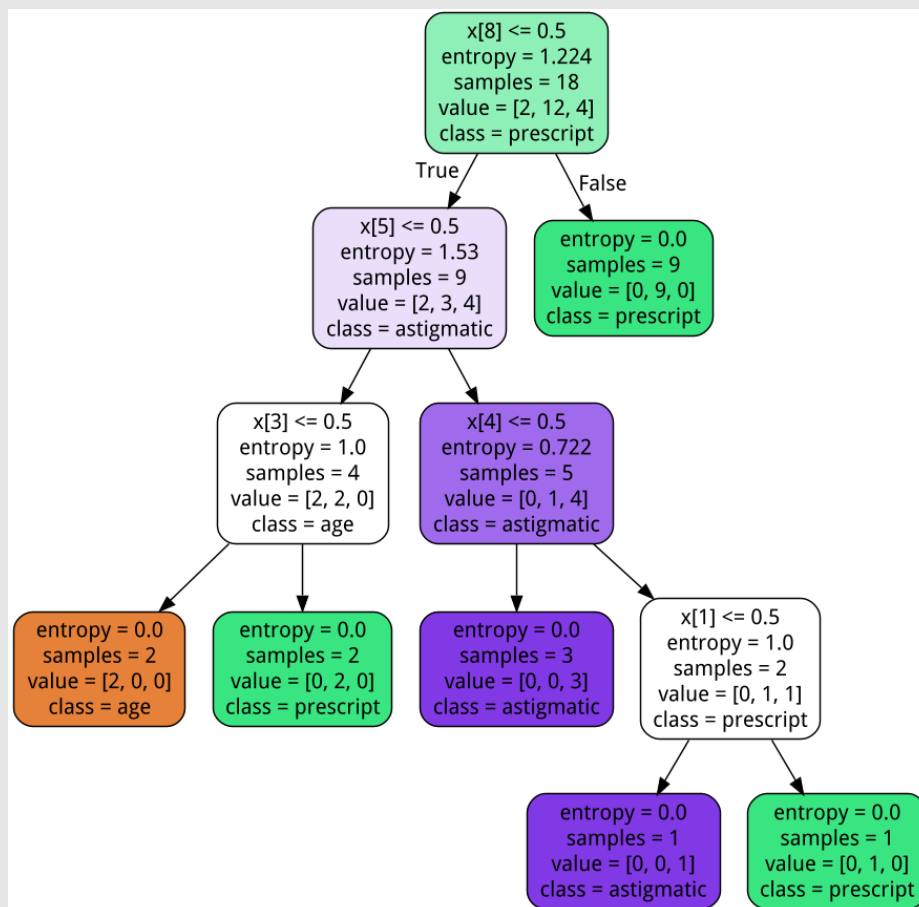


图 5

5) 输出预测准确率, 请给出代码

使用 `sklearn.metrics` 中的 `accuracy_score()` 实现准确率的计算。

补全代码:

```
# 对test集进行预测
predictedY = clf.predict(X_test)

# 预测结果
acc = metrics.accuracy_score(y_test, predictedY)
print("accuracy is: {}".format(acc))
```

预测准确率为:

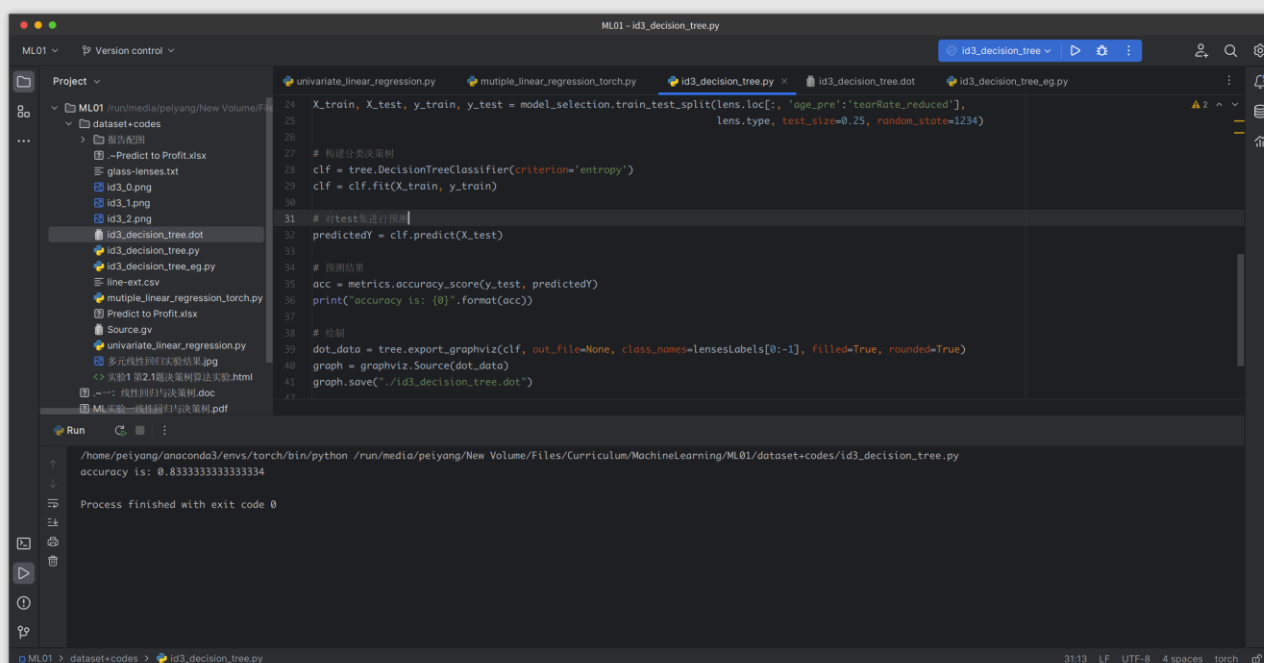


图 6

决策树的准确度为 0.8334.

3. 多元线性回归模型实验 (选做, 加分)

以某产品的销售利润数据为例, 该数据集 (Predict to Profit.xlsx) 包含 5 个变量, 分别是产品的研发成本、管理成本、市场营销成本、销售市场和销售利润。数据集的部分截图如下所示。请根据此数据集建立一个预测利润的多元线性模型。

(1) 使用 `torch` 编写数据加载器 `data_loader` 函数

```
def data_loader(data_array, batch_size, is_train=True):
    data_set = torch.utils.data.TensorDataset(*data_array)
    return torch.utils.data.DataLoader(data_set, batch_size, shuffle=is_train)
```

(2) 数据特征中具有离散型变量，使用独热编码 One-Hot Encoding 进行处理

```
# 对离散型特征做独热编码处理
label = sklearn.preprocessing.LabelEncoder()
state_label = label.fit_transform(data["State"])
# print(state_label)
state_label = state_label.reshape(len(state_label), 1)
state_encoder = sklearn.preprocessing.OneHotEncoder(sparse_output=False)
state_encoded = state_encoder.fit_transform(state_label)
# print("one-hot encoding of state is:\n{0}".format(state_encoded))
```

(3) 以 7:3 的比例划分训练集和验证集

```
# 以7:3的比例划分训练集和验证集
index = len(x) // 10 * 7
train_x = x[0:index].to(torch.float32)
train_y = y[0:index].to(torch.float32)
eval_x = x[index:].to(torch.float32)
eval_y = y[index:].to(torch.float32)
original_train_x = train_x.clone().detach()
original_train_y = train_y.clone().detach()
```

(4) 对特征进行归一化处理

```
# 对数据进行线性归一化处理
for i in range(train_x.shape[1]):
    train_x[:, i] = normalization(train_x[:, i])
train_y[:, 0] = normalization(train_y[:, 0])
for i in range(eval_x.shape[1]):
    eval_x[:, i] = normalization(eval_x[:, i])
eval_y[:, 0] = normalization(eval_y[:, 0])
print("\nThe shape of train dataset:\nx: {0}, y: {1}".format(train_x.shape,
train_y.shape))
print("\nThe shape of eval dataset:\nx: {0}, y: {1}".format(eval_x.shape,
eval_y.shape))
```

(5) 使用 torch 进行梯度下降的发放进行训练，定义模型、损失函数以及训练相关参数

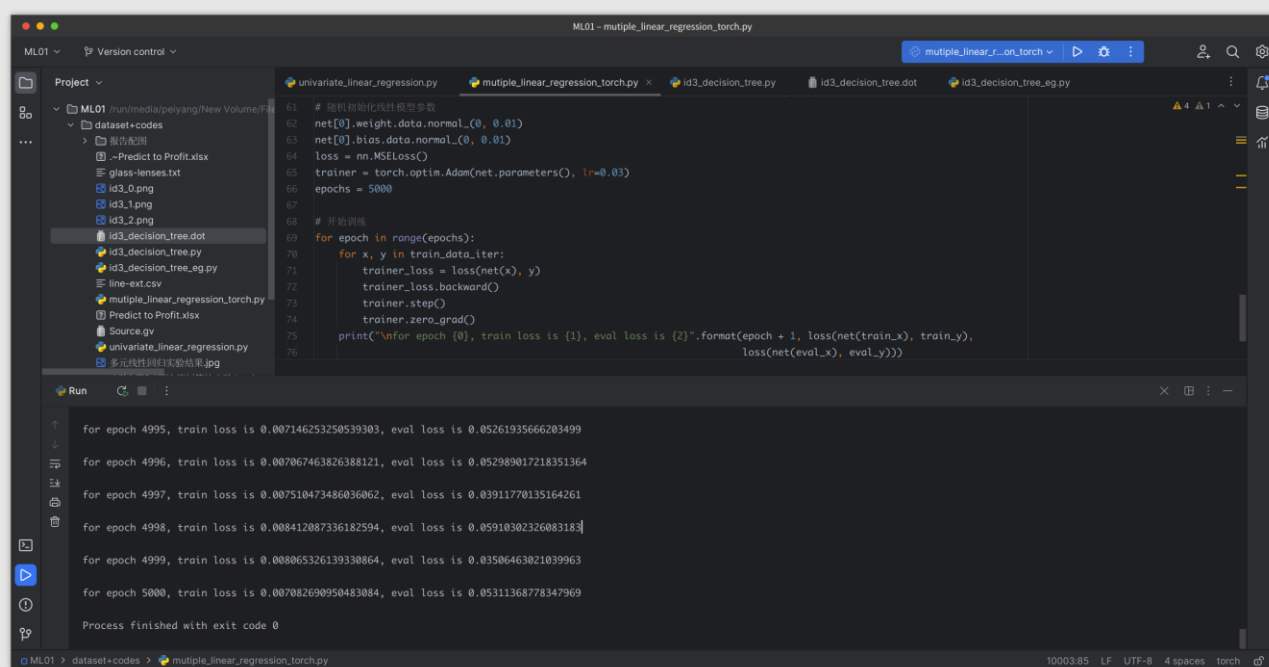
```
# 使用dataset和dataloader定义数据枚举器
batch_size = 5
train_data_iter = data_loader((train_x, train_y), batch_size=batch_size)

# 定义线性模型
net = nn.Sequential(nn.Linear(in_features=train_x.shape[1],
out_features=train_y.shape[1]))
# 随机初始化线性模型参数
net[0].weight.data.normal_(0, 0.01)
net[0].bias.data.normal_(0, 0.01)
loss = nn.MSELoss()
trainer = torch.optim.Adam(net.parameters(), lr=0.03)
epochs = 5000
```

(6) 训练过程中, 每个 epoch 分别打印在整个训练集上的均方差损失以及验证集上的均方差损失

```
# 开始训练
for epoch in range(epochs):
    for x, y in train_data_iter:
        trainer_loss = loss(net(x), y)
        trainer_loss.backward()
        trainer.step()
        trainer.zero_grad()
    print("\nfor epoch {0}, train loss is {1}, eval loss is {2}".format(epoch
+ 1, loss(net(train_x), train_y), loss(net(eval_x), eval_y)))
```

训练结果截图:



```
61 # 随机初始化线性模型参数
62 net[0].weight.data.normal_(0, 0.01)
63 net[0].bias.data.normal_(0, 0.01)
64 loss = nn.MSELoss()
65 trainer = torch.optim.Adam(net.parameters(), lr=0.03)
66 epochs = 5000
67
68 # 开始训练
69 for epoch in range(epochs):
70     for x, y in train_data_iter:
71         trainer_loss = loss(net(x), y)
72         trainer_loss.backward()
73         trainer.step()
74         trainer.zero_grad()
75     print("for epoch {0}, train loss is {1}, eval loss is {2}".format(epoch + 1, loss(net(train_x), train_y),
76                                                                     loss(net(eval_x), eval_y)))
```

for epoch 4995, train loss is 0.007146253250539303, eval loss is 0.05261935666203499

for epoch 4996, train loss is 0.007067463826388121, eval loss is 0.052989017218351364

for epoch 4997, train loss is 0.007510473486036062, eval loss is 0.03911770135164261

for epoch 4998, train loss is 0.008412087336182594, eval loss is 0.05910302326083183

for epoch 4999, train loss is 0.008065326139330864, eval loss is 0.03506463021039963

for epoch 5000, train loss is 0.007082690950483084, eval loss is 0.05311368778347969

Process finished with exit code 0

图 7

由图可知,在 `batch_size` 为 5,学习率为 0.03,优化器为 Adam 的条件下,训练 5000epochs 后,模型在训练集上的均方误差为 0.007,在验证集上的均方误差为 0.05.

三. 总结 (实验中遇到的问题和解决问题的方法、实验收获)

1. 在解决附加题多元线性回归时,遇到了离散特征,采用了独热编码进行处理,将特征向量扩展为[1, 6]的维度。

2. 由于独热编码的数值为 0 或 1,而其他特征值具有较大值,特征值之间存在量纲不同的问题,故需要进行归一化处理。

由于特征分布不一定符合正态分布的规律,故不适合 z 型归一化,最终使用最大值-最小值方法进行归一化。

3. 训练采用 torch 框架的随机梯度下降算法,在训练时每个 epoch 输出训练集的均方差损失以及验证集的均方差损失,方便观察过拟合等现象,以便调整 batch size、优化器、学习率等参数。

四. 选做题代码附录

```
1. import sklearn.preprocessing
2. import torch
3. import pandas as pd
```

```
4. import torch.nn as nn
5. import sklearn
6.
7.
8. def data_loader(data_array, batch_size, is_train=True):
9.     data_set = torch.utils.data.TensorDataset(*data_array)
10.    return torch.utils.data.DataLoader(data_set, batch_size, shuffle=is_train)
11.
12.
13. def normalization(data_array):
14.    return (data_array - torch.min(data_array)) / (torch.max(data_array) - torch.min(data_array))
15.
16.
17. data = pd.read_excel("Predict to Profit.xlsx")
18. # print("original data is:\n{0}".format(data))
19.
20. # 对离散型特征做独热编码处理
21. label = sklearn.preprocessing.LabelEncoder()
22. state_label = label.fit_transform(data["State"])
23. # print(state_label)
24. state_label = state_label.reshape(len(state_label), 1)
25. state_encoder = sklearn.preprocessing.OneHotEncoder(sparse_output=False)
26. state_encoded = state_encoder.fit_transform(state_label)
27. # print("one-hot encoding of state is:\n{0}".format(state_encoded))
28.
29. # 特征数据处理
30. x = torch.concat((torch.tensor([list(data["RD_Spend"])]).reshape(-1, 1),
31.                  torch.tensor([list(data["Administration"])]).reshape(-1, 1)), -1)
32. x = torch.concat((x, torch.tensor([list(data["Marketing_Spend"])]).reshape(-1, 1)), -1)
33. x = torch.concat((x, torch.tensor(state_encoded)), -1)
34. y = torch.tensor(list(data["Profit"])).reshape(-1, 1)
35.
36. # 以 7:3 的比例划分训练集和验证集
37. index = len(x) // 10 * 7
38. train_x = x[0:index].to(torch.float32)
39. train_y = y[0:index].to(torch.float32)
40. eval_x = x[index:].to(torch.float32)
41. eval_y = y[index:].to(torch.float32)
42. original_train_x = train_x.clone().detach()
43. original_train_y = train_y.clone().detach()
```

```
44.
45. # 对数据进行线性归一化处理
46. for i in range(train_x.shape[1]):
47.     train_x[:, i] = normalization(train_x[:, i])
48. train_y[:, 0] = normalization(train_y[:, 0])
49. for i in range(eval_x.shape[1]):
50.     eval_x[:, i] = normalization(eval_x[:, i])
51. eval_y[:, 0] = normalization(eval_y[:, 0])
52. print("\nThe shape of train dataset:\nx: {0}, y: {1}".format(train_x.shape, train_y.shape))
53. print("\nThe shape of eval dataset:\nx: {0}, y: {1}".format(eval_x.shape, eval_y.shape))
54.
55. # 使用 dataset 和 dataLoader 定义数据枚举器
56. batch_size = 5
57. train_data_iter = data_loader((train_x, train_y), batch_size=batch_size)
58.
59. # 定义线性模型
60. net = nn.Sequential(nn.Linear(in_features=train_x.shape[1], out_features=train_y.shape[1]))
61. # 随机初始化线性模型参数
62. net[0].weight.data.normal_(0, 0.01)
63. net[0].bias.data.normal_(0, 0.01)
64. loss = nn.MSELoss()
65. trainer = torch.optim.Adam(net.parameters(), lr=0.03)
66. epochs = 5000
67.
68. # 开始训练
69. for epoch in range(epochs):
70.     for x, y in train_data_iter:
71.         trainer_loss = loss(net(x), y)
72.         trainer_loss.backward()
73.         trainer.step()
74.         trainer.zero_grad()
75.     print("\nfor epoch {0}, train loss is {1}, eval loss is {2}".format(epoch + 1, loss(net(train_x), train_y), loss(net(eval_x), eval_y)))
```