实验三食用指南

试鸢

一、实验准备

1. 创建一个合适的虚拟环境

```
conda create -n zju-yee python=3.10.16
```

2. 安装该实验需要的包

```
conda activate zju-yee
pip install scikit-learn matplotlib numpy conda ipykernel
```

3. 配置 **Jupyter Notebook** 使用我们的虚拟环境

以 vscode 为例，在 `文件>打开文件夹` 中打开你想放置代码的文件夹， 然后你应该能看到你的资源管理器，大概如下图所示

图 1 vscode 资源管理器

在资源管理器的空白处点击右键选择“新建文件”，命名为 `lab3.ipynb`，它应该会自动打开，如果没有请双击。最后如下图

(这个应该是 IPython Notebook 的缩写,你可以这样记住文件的后缀名)

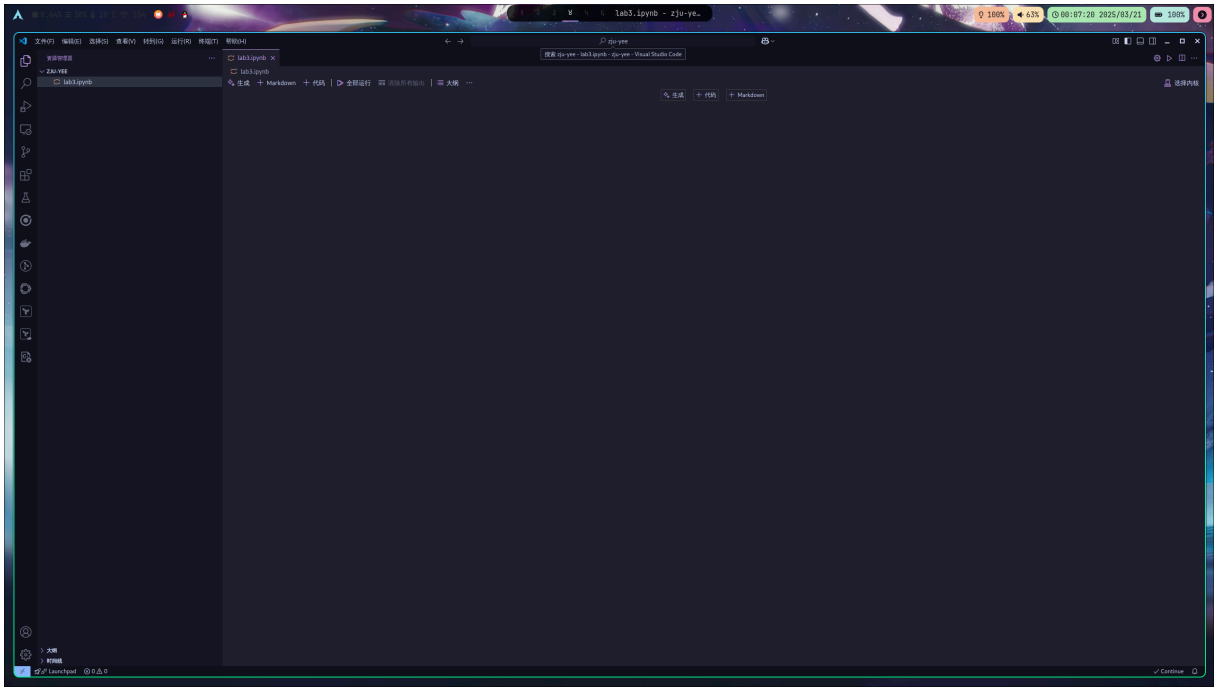
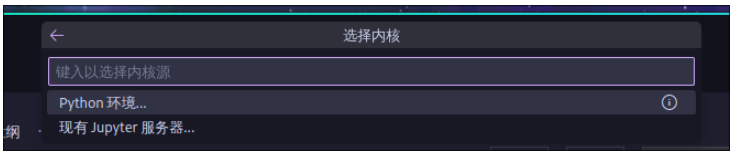
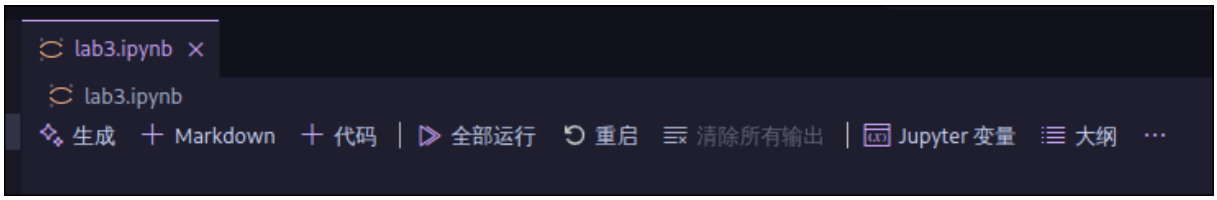


图 2 lab3.ipynb

更换 notebook 的环境，使其使用我们的虚拟环境。
点击右上角的选择内核，在中间的下拉框中点击 Python 环境...，选择我们刚刚创建的虚拟环境 zju-yee



接着点击 +代码,尝试输入以下代码并运行。




```python
import sys,sklearn,matplotlib
 # 输出python的路径，你应该能在路径中看到你的虚拟环境名称，如 zju-yee
print(sys.executable)
# 输出示例
# /home/yee/conda/envs/zju-yee/bin/python
```

至此实验准备完成。

二、sklearn 和 Linear Regression(线性回归)入门

1. 回归

回归是一类典型的监督学习问题，旨在通过输入特征预测一个连续的数值。这类问题的数据集通常由两部分组成：**特征（自变量）** 和 **标签（因变量）**。特征是用于预测的输入数据，标签是想要预测的输出数据。

与分类问题不同，分类问题的输出是离散的（例如预测一张图片是猫还是狗），而回归问题的输出是连续的（例如预测某地区的房价）。

线性回归所做的,就是将所有标签 y 预测为一个关于特征的函数 x。

$$h_{\theta(x)} = \hat{y}_i = \theta_0 + \theta_1 x_i$$

其中：

- θ_0 是截距（偏置项），
- θ_1 是斜率（权重），
- x_i 是第 i 个样本的特征值，
- $\hat{y}_i$ 是对应的预测值。

2. 损失函数

为了找到最优的函数，我们需要定义一个损失函数来度量预测值与真实值之间的差距。对于线性回归，可以定义损失函数为：

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - h_{\theta(x_i)} \right)^2$$

其中：

- y_i 是第 i 个样本的真实标签，
- n 是样本数量。

我们的目标是通过最小化损失函数 $J(\theta_0, \theta_1)$ 来找到最优的参数 θ_0 和 θ_1。

3. 梯度下降法

为了最小化损失函数，我们通常使用 **梯度下降法**。梯度下降法的核心思想是通过迭代更新参数，逐步逼近损失函数的最小值。

下面是梯度下降法的更新规则：

$$temp_0 := \theta_0 - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0}$$

$$temp_1 := \theta_1 - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1}$$

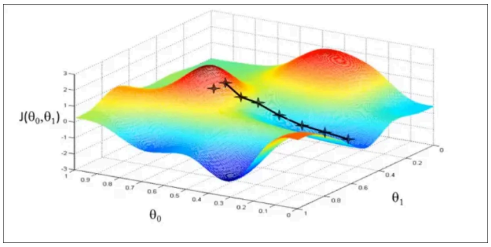
$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$

其中，α 是学习率，控制每次更新的步长。

重复上述步骤，直到损失函数收敛。

这是一张很经典的梯度下降示意图，



，

以上就是线性回归的基本原理，接下来我们将使用 **sklearn** 来实现一个简单的线性回归。

```python
import numpy as np
from sklearn.model_selection import train_test_split

# 设置随机种子以确保结果可复现
np.random.seed(67656)

# 生成模拟数据
X = 2 * np.random.rand(100, 1)  # 生成100个随机点
y = 3 * X.flatten() + 2 + np.random.randn(100) * 0.5 # y = 3x + 2 加入噪声

# 将数据分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```python
from sklearn.linear_model import LinearRegression

# 创建线性回归模型
model = LinearRegression()

# 训练模型
model.fit(X_train, y_train)
```

```python
# 进行预测
y_pred = model.predict(X_test)
```

```python
from sklearn.metrics import mean_squared_error, r2_score

# 计算性能指标
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'均方误差 (MSE): {mse}')
print(f'决定系数 (R^2): {r2}')
```

```python
import matplotlib.pyplot as plt

# 原始数据与预测值的可视化
plt.scatter(X, y, color='blue', label='target', s=10)
X_line = np.linspace(0, 2, 100).reshape(-1, 1)  # 创建用于绘制回归线的X值
y_line = model.predict(X_line)
plt.plot(X_line, y_line, color='red', linewidth=2, label='prediction')
plt.xlabel('X')
plt.ylabel('y')
plt.title(f"y = {model.coef_[0]:.2f}x + {model.intercept_:.2f}")
plt.legend()
plt.grid()
plt.show()
```

然后，下面是加州房价数据集的线性回归

```
from sklearn.datasets import fetch_california_housing
# import pandas as pd

cal = fetch_california_housing() # 加载数据集

# 得到样本特征与样本目标值
# X = pd.DataFrame(cal.data, columns=cal.feature_names)
# y = pd.Series(cal.target, name='target')
X = cal.data
y = cal.target
```

```
from sklearn.model_selection import train_test_split
# 检查缺失值
# print(X.isnull().sum())  这个是pandas的函数，事实上该数据集没有缺失值

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train, y_train)
```

```
from sklearn.metrics import mean_squared_error, r2_score

y_pred = model.predict(X_test)

print(f'均方误差 (MSE): {mean_squared_error(y_test, y_pred)}')
print(f'决定系数 (R^2): {r2_score(y_test, y_pred)}')
```

```
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.xlabel('target')
plt.ylabel('prediction')
plt.title('target vs prediction')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linewidth=2)  # 理想预测线

# 在理想情况下，模型预测值应该与真实值完全一致，所有点都会落在这条红线上。具体来说：
# - 如果预测点位于红线上方，表示模型预测值高于实际值（过高估计）
# - 如果预测点位于红线下方，表示模型预测值低于实际值（过低估计）
# - 预测点离红线越远，说明预测误差越大

plt.show()
```

三、乳腺癌数据集的逻辑回归

首先我们要明白，逻辑回归是一个二分类模型，它的输出是一个概率值，表示样本属于某个类别的概率。我们可以通过设置一个阈值（通常是 0.5）来将概率值转换为类别标签。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, roc_curve,
auc,precision_recall_curve,average_precision_score
# 1. 加载乳腺癌数据集
data = load_breast_cancer()
X = data.data
y = data.target

# 2. 数据标准化
# 需要标准化的原因
# - 逻辑回归模型对特征的尺度敏感，标准化可以提高模型的收敛速度和性能
# - 标准化可以使得不同特征在同一尺度上进行比较，避免某些特征对模型的影响过大
# 这里我们用StandardScaler 进行标准化，使得每个特征的均值为0，标准差为1
# 你也可以使用MinMaxScaler进行归一化，使得每个特征的值在0到1之间
scaler = StandardScaler()
# scaler = MinMaxScaler()
X = scaler.fit_transform(X)

# 3. 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=67656)
# test_size=0.20表示将20%的数据用于测试，80%用于训练
# random_state=67656是随机种子，确保每次划分数据集时都能得到相同的结果
# 这个别抄，注意审题

# 4. 训练逻辑回归模型
model = LogisticRegression()
model.fit(X_train, y_train)

# 5. 进行预测
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]  # 预测为正类的概率
# predict_proba返回的是每个类别的概率，[:, 1]表示取正类的概率，[:, 0]表示取负类的
概率

# 6. 评估模型性能
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
print(f'准确率 (Accuracy): {accuracy:.2f}')
print(f'精确率 (Precision): {precision:.2f}')
print(f'召回率 (Recall): {recall:.2f}')
print(f'F1 分数 (F1 Score): {f1:.2f}')

# 7. 绘制混淆矩阵 Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# 8. 绘制 ROC 曲线 Receiver Operating Characteristic Curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.grid(True)
plt.show()

# 9. 绘制PR曲线 Precision-Recall Curve
# PR曲线是另一个二分类模型性能评估的标准工具，它展示了精确率（Precision）和召回率
（Recall）之间的关系
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
average_precision = average_precision_score(y_test, y_pred_proba)
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'PR curve (area =
{average_precision:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()
```

四、其他补充

1. 混淆矩阵 Confusion Matrix

	预测正类	预测负类
实际正类	TP	FN
实际负类	FP	TN

，

2. ROC 曲线 Receiver Operating Characteristic Curve

ROC 曲线是一个二分类模型性能评估的标准工具，它展示了真正率（TPR）和假正率（FPR）之间的关系

- TPR = TP / (TP + FN) 假正率（FPR） = FP / (FP + TN)
- ROC 曲线下面积（AUC）越接近 1，模型的分类能力越强
- ROC 曲线下面积（AUC）越接近 0.5，模型的分类能力越弱

3. PR 曲线 Precision-Recall Curve

PR 曲线是另一个二分类模型性能评估的标准工具，它展示了精确率（Precision）和召回率（Recall）之间的关系

- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)

4. 其他

```
y_pred_proba = model.predict_proba(X_test)[:, 1]  # 正类的概率
threshold = 0.3  # 自定义阈值
y_pred_ = (probabilities ≥ threshold).astype(int)
```