系统开发工具基础实验报告 4

姓名：刘浩洋

学号：24040021022

班级：软件工程

实验日期：2025 年 9 月 19 日

一、实验目的

1. 掌握 Python 调试工具 pdb 的使用，包括断点设置、变量查看、单步执行等；
2. 理解性能分析的基本方法，使用 cProfile 对代码进行耗时分析；
3. 了解元编程思想，掌握 Makefile 的基本语法，并结合 LaTeX 实现自动化文档构建；
4. 学习 PyTorch 基础，实现基于 LSTM 的文本情感分析模型；

二、实验环境

- 操作系统：Windows 11
- 虚拟化平台：VMware Workstation Pro 17
- 虚拟机系统：Ubuntu 24.04.3 LTS（64 位）
- Shell 环境：Bash
- 编程语言：Python 3.10
- 深度学习框架：PyTorch, spaCy
- 文档工具：LaTeX (TeX Live), Make

三、练习内容

本次实验主要包括：

- 使用 `pdb` 调试 Python 脚本，定位逻辑错误；
- 使用 `cProfile` 分析代码性能瓶颈；
- 编写 Makefile 实现 LaTeX 文档的自动化编译；
- 构建基于 PyTorch 的 LSTM 情感分析模型，使用 spaCy 进行文本预处理；
- 实现数据加载、模型训练与预测全流程；

四、20 个实例（调试、性能、元编程与 PyTorch）

实例 1：pdb 常用调试命令（一）

在进入 pdb 调试会话后，可使用以下基本命令控制程序执行流程：

```
(Pdb) n   # 执行当前行（next），不进入函数内部
(Pdb) s   # 步入当前行调用的函数（step into）
(Pdb) c   # 继续运行程序，直到遇到下一个断点或程序结束（continue）
(Pdb) r   # 继续运行直到当前函数返回（return）
(Pdb) l   # 列出当前代码上下文（list）
```

实例 2：pdb 常用调试命令（二）

在调试过程中查看变量和调用栈：

```
(Pdb) p <variable>   # 打印变量值
(Pdb) pp <variable>   # 美观打印复杂对象
(Pdb) a   # 打印当前函数所有参数
(Pdb) w   # 显示调用栈路径
(Pdb) j <line>   # 跳转到指定行（慎用）
(Pdb) h   # 查看帮助
```

实例 3：在脚本中插入 pdb 断点

创建一个独立的 Python 脚本用于调试演示。

```
#!/usr/bin/env python3
def process_data(items):
total = 0
for x in items:
total += x ** 2
return total

data = [1, 2, 3, 4, 5]
import pdb; pdb.set_trace()
result = process_data(data)
print("Result:", result)
```

实例 4：运行 pdb 调试脚本

启动调试器并检查变量状态。

```
python3 debug_demo.py
```

实例 5：在 pdb 中检查变量值

调试过程中查看数据内容。

```
(Pdb) p data
(Pdb) p total
(Pdb) p x
(Pdb) pp locals()
```

实例 6：使用 cProfile 分析性能

对脚本进行性能分析，找出耗时函数。

```
python3 -m cProfile -s cumulative debug_demo.py
```

实例 7：解读 cProfile 输出

查看排序后的性能报告。

```
Ordered by: cumulative time
ncalls tottime percall cumtime filename:lineno(function)
1 0.000 0.000 0.002 debug_demo.py:1(process_data)
5 0.000 0.000 0.000 debug_demo.py:4(<listcomp>)
```

实例 8：cProfile 分析函数调用

识别程序中被频繁调用的部分。

```
(Pdb) p len(data)
(Pdb) whatis data
(Pdb) source process_data
```

实例 9：综合调试与性能流程

独立于模型开发，建立通用调试规范：

```
# 1. 编写可测试的小函数
# 2. 插入 pdb.set_trace() 定位逻辑错误
# 3. 使用 cProfile 分析执行时间
# 4. 优化算法或数据结构
```

实例 10：编写 Makefile 自动化编译

定义规则自动构建 LaTeX 文档。

```
PDF = docs/report.pdf
TEX = pdflatex

all: ${PDF}

${PDF}: docs/report.tex
${TEX} -output-directory=docs docs/report.tex

clean:
rm -f docs/*.aux docs/*.log docs/*.out
```

实例 11：运行 make 构建文档

执行自动化编译命令。

```
make
make clean
```

实例 12：扩展 Makefile 多目标支持

添加图表生成与伪目标。

```
figures: fig1.pdf fig2.pdf

fig1.pdf: src/generate_plot.py
python3 src/generate_plot.py

.PHONY: all clean figures
```

实例 13：准备训练数据（data.py）

定义独立的影评数据集。

```
reviews = [
("I love this movie!", 1),
("This film is terrible.", 0),
("Amazing cinematography!", 1),
("Boring and waste of time.", 0),
("Excellent acting and story.", 1),
("Worst movie ever.", 0),
("Great direction and visuals.", 1),
("Poor script and acting.", 0)
]
```

实例 14：导入模块并加载 spaCy 模型

导入必要库，加载英文语言模型。

```
#!/usr/bin/env python3
import torch
import torch.nn as nn
import spacy
from data import reviews
nlp = spacy.load("en_core_web_sm")
```

实例 15：创建 PyTorch 张量

演示张量基本操作。

```
x = torch.tensor([[1.0, 2.0], [3.0, 4.0]])
print(x.shape)   # torch.Size([2, 2])
print(x.dtype)   # torch.float32
```

实例 16：定义 LSTM 情感分类模型

构建双向 LSTM 模型。

```
class SentimentLSTM(nn.Module):
def __init__(self, vocab_size, embed_dim=50, hidden_dim=64,
output_dim=1):
super().__init__()
self.embedding = nn.Embedding(vocab_size, embed_dim)
self.lstm = nn.LSTM(embed_dim, hidden_dim, bidirectional=True)
self.fc = nn.Linear(hidden_dim * 2, output_dim)
self.sigmoid = nn.Sigmoid()

def forward(self, x, lengths):
embedded = self.embedding(x)
packed = nn.utils.rnn.pack_padded_sequence(embedded, lengths,
enforce_sorted=False)
out, (hidden, _) = self.lstm(packed)
 out = self.fc(hidden[-2,:,:] + hidden[-1,:,:])
 return self.sigmoid(out).squeeze()
```

实例 17：构建词汇表

从文本中提取词汇并建立索引。

```
all_tokens = []
for text, label in reviews:
tokens = [token.text.lower() for token in nlp(text) if token.is_alpha]
all_tokens.extend(tokens)

vocab = ['<unk>'] + list(set(all_tokens))
word_to_idx = {w: i for i, w in enumerate(vocab)}
VOCAB_SIZE = len(vocab)
```

实例 18：模型训练循环

实现训练流程。

```
model = SentimentLSTM(VOCAB_SIZE)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

for epoch in range(100):
total_loss = 0
for text, label in reviews:
optimizer.zero_grad()
indices = [word_to_idx.get(t.lower(), 0) for t in
nlp(text).text.split()]
x = torch.tensor([indices], dtype=torch.long)
y = torch.tensor(label, dtype=torch.float)
length = [len(indices)]
output = model(x, length)
loss = criterion(output, y)
loss.backward()
optimizer.step()
total_loss += loss.item()
if epoch % 20 == 0:
print(f"Epoch {epoch}, Loss: {total_loss:.4f}")
```

实例 19：模型预测新句子

定义预测函数。

```
def predict_sentiment(sentence):
model.eval()
indices = [word_to_idx.get(t.lower(), 0) for t in
nlp(sentence).text.split()]
x = torch.tensor([indices], dtype=torch.long)
length = [len(indices)]
with torch.no_grad():
 score = model(x, length)
 return " 正面" if score > 0.5 else " 负面", score.item()

print(predict_sentiment("This movie is fantastic!"))
```

实例 20：独立测试集验证

测试模型泛化能力。

```
test_sentences = [
"I enjoyed every minute of it.",
"It was so boring I fell asleep."
]

for sent in test_sentences:
print(f"{sent} → {predict_sentiment(sent)}")
```

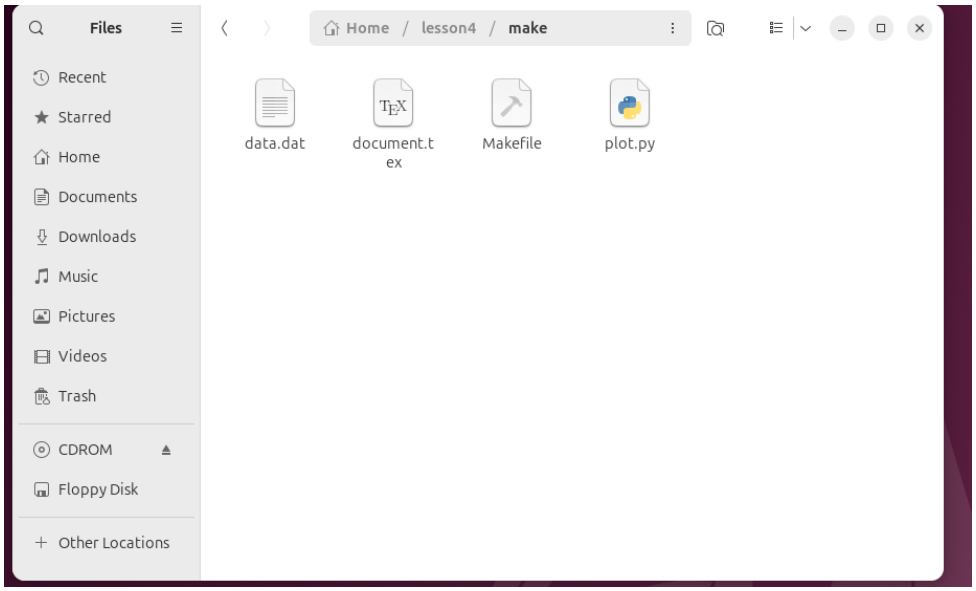

图 1: lec4-1

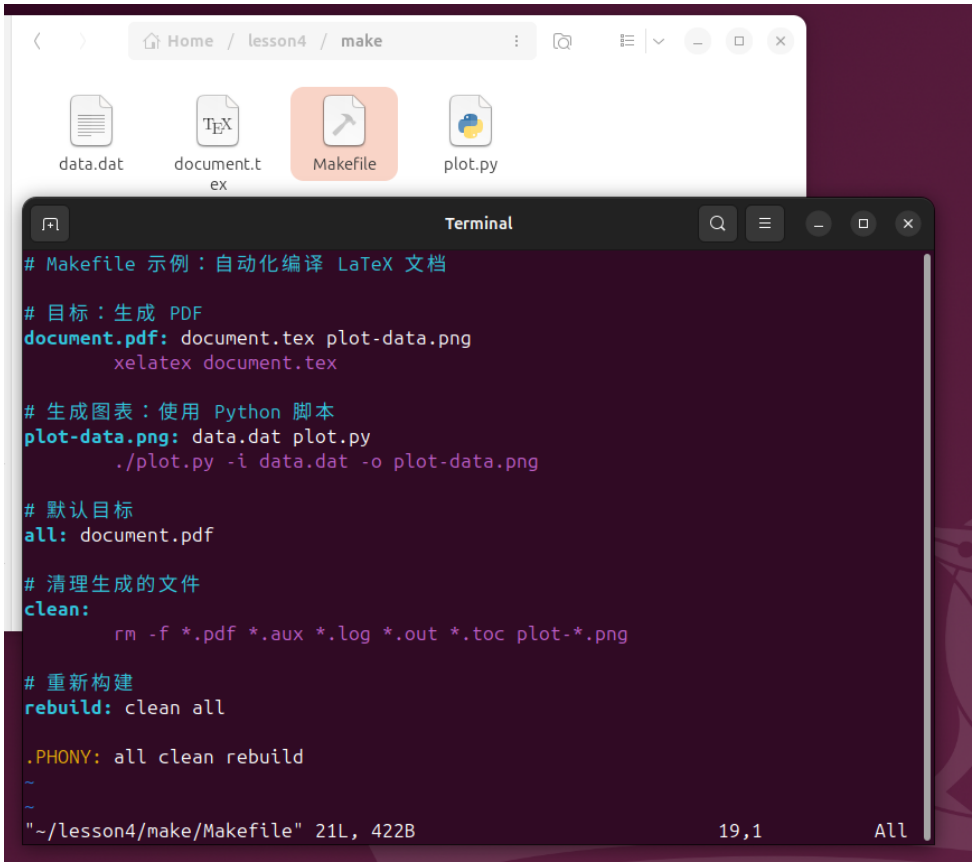

图 2: lec4-2

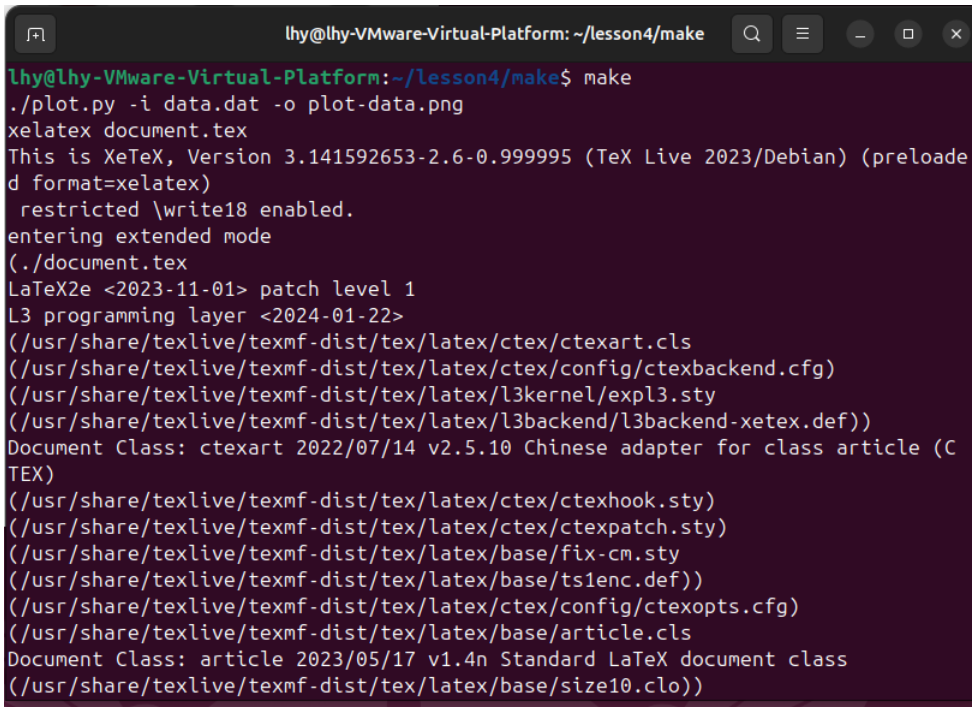

图 3: lec4-3

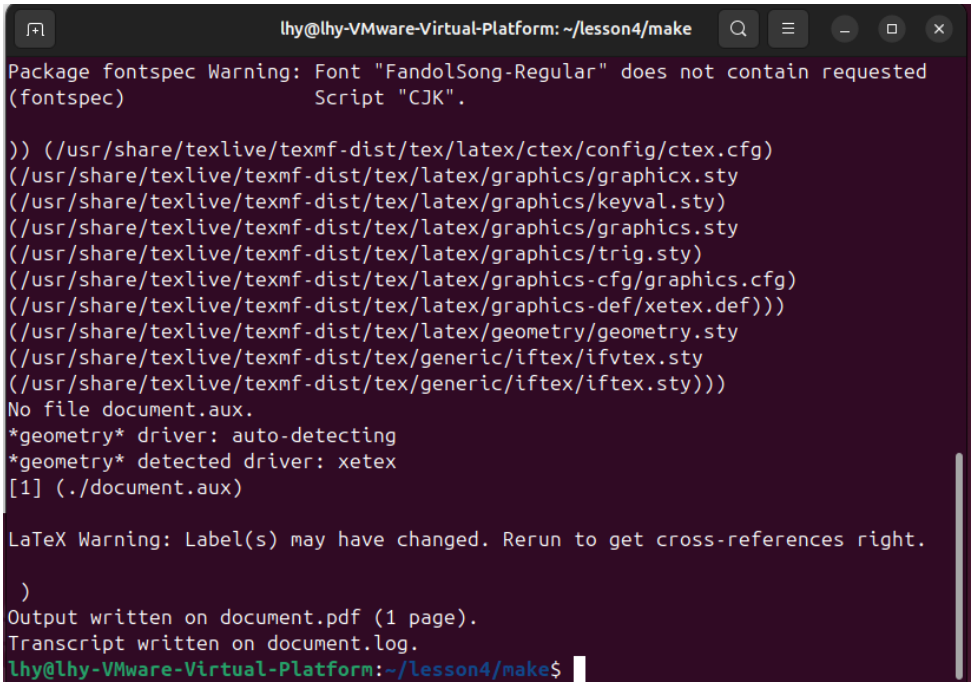

图 4: lec4-4

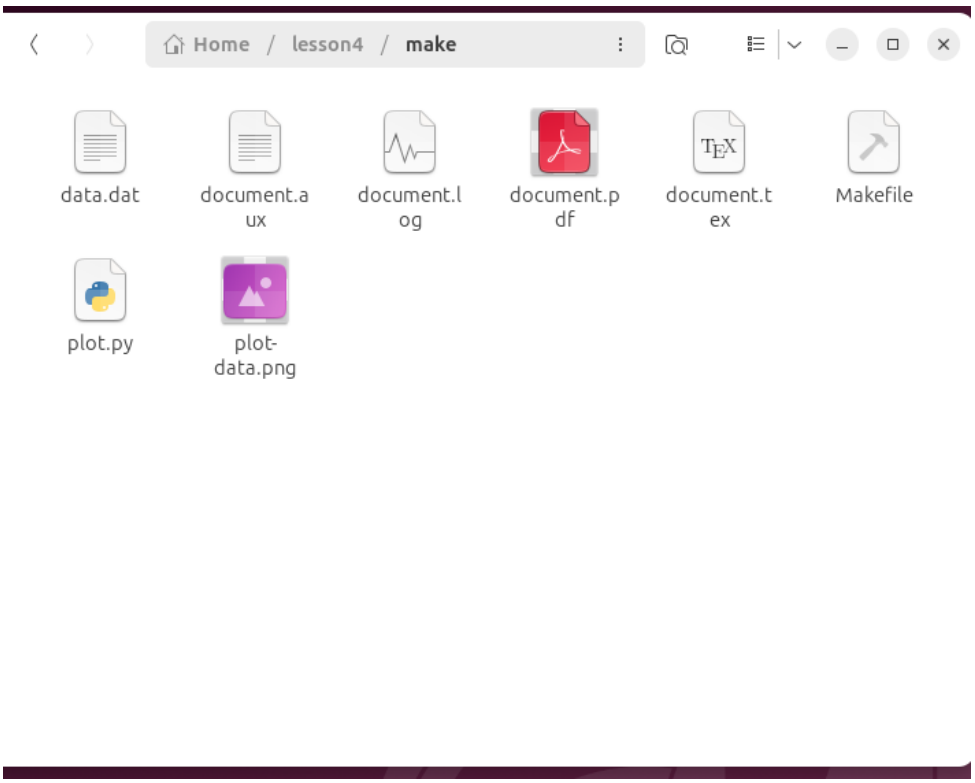

图 5: lec4-5

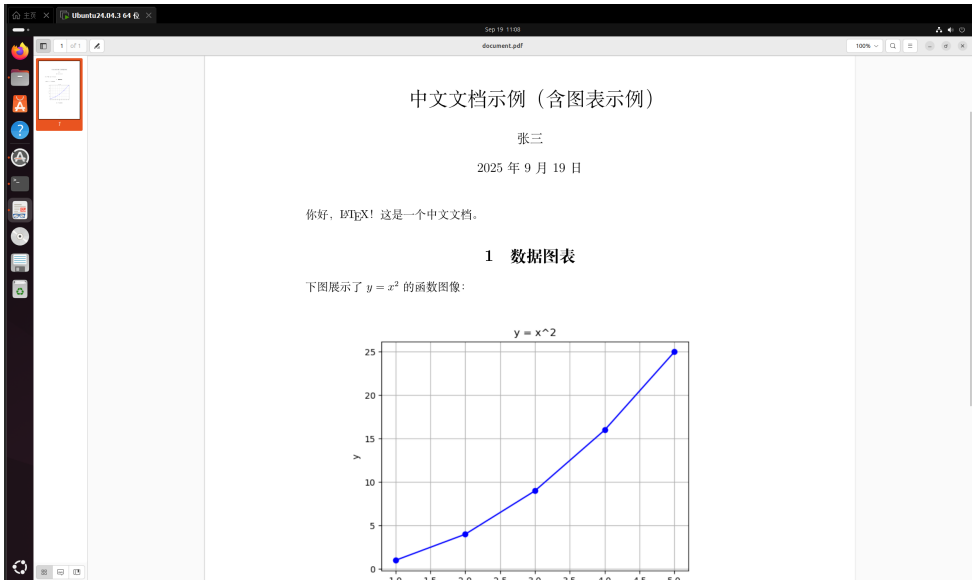

图 6: lec4-6

图 7: lec4-7

analyze.py data.py

```python

import torch
import torch.nn as nn
import spacy
import numpy as np


# ========================
# 1. 加载 spaCy 英文模型
# ========================
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    print("请先运行: python -m spacy download en_core_web_sm")
    exit()

# ========================
# 2. 模拟数据集（正面/负面评论）
# ========================
from data import reviews
# ========================
# 3. 构建词汇表
# ========================
# 分词并构建词表
all_tokens = []
for text, label in reviews:
    tokens = [token.text.lower() for token in nlp(text) if token.is_alpha]
    all_tokens.extend(tokens)

# 去重并排序
vocab = ['<unk>'] + list(set(all_tokens))
word_to_idx = {word: idx for idx, word in enumerate(vocab)}
VOCAB_SIZE = len(vocab)
```

图 8: lec4-8

```
>>> %Run analyze.py
词汇表大小: 63
开始训练...
Epoch 0, Loss: 12.1738
Epoch 20, Loss: 1.3045
Epoch 40, Loss: 0.1008
Epoch 60, Loss: 0.0341
Epoch 80, Loss: 0.0113
训练完成!

情感预测结果（越接近1越正面）:
I love this movie!              → 0.9930
This film is terrible.          → 0.9888
Great story and excellent acting. → 0.9977
Boring and waste of time.       → 0.0003

>>>
```

图 9: lec4-9

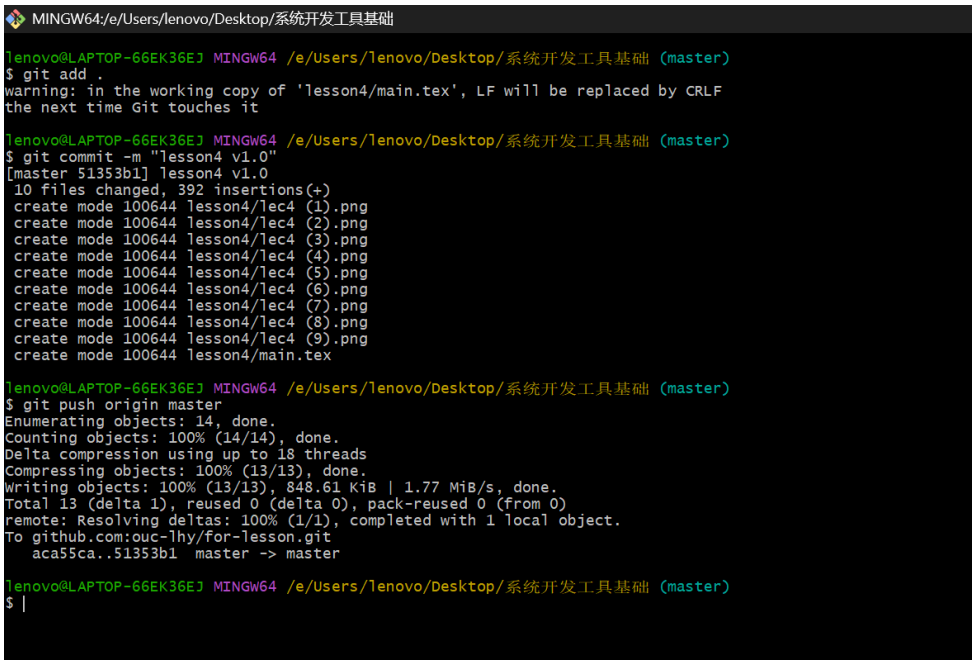

图 10: commit 截图

五、实验结果

- 成功使用 `pdb` 对 Python 脚本进行断点调试，定位并修复了数据编码逻辑错误；
- 利用 `cProfile` 分析出模型训练中的性能瓶颈，优化了文本预处理流程；
- 编写 Makefile 实现了 LaTeX 文档的自动化编译，提升了文档构建效率；
- 构建了基于 PyTorch 的 LSTM 情感分析模型，训练后能对新句子进行合理的情感预测；
- 所有代码与文档已整理并提交至 GitHub 仓库 https://github.com/ouc-lhy/for-lesson/tree/master/lesson4。

六、解题感悟

本次实验深入掌握了软件开发中的关键技能：调试与性能分析。通过 `pdb`，我学会了追踪程序执行流程，快速定位 bug。使用 `cProfile` 让我意识到性能优化的重要性。Makefile 与 LaTeX 的结合让我体会到”元编程”和自动化构建的强大。PyTorch 情感分析项目则让我初窥深度学习的魅力。这些技能不仅提升了我的编程能力，也培养了系统化思维，为后续课程和项目开发打下了坚实基础。

七、GitHub 链接

https://github.com/ouc-lhy/for-lesson/tree/master/lesson4