“豆瓣电影评论情感分析”项目总结报告

成员： 丁华威 陈梓轩 陈意博 符景乐 郑宇轩

1. 项目概述

1.1 项目背景

何谓文本情感分析，在当今信息时代，海量的文本数据中蕴含着丰富的用户情感信息，利用算法来分析提取文本中表达的情感。例如分析一个句子表达的好、坏等判断，高兴、悲伤、愤怒等情绪。如果能将这种文字转为情感的操作让计算机自动完成，就节省了大量的时间。对于目前的海量文本数据来说，这是很有必要的。

1.2 总体目标

总体来说，该项目的目标是利用爬虫技术获取电影评论数据，通过深度学习（bert-base-chinese模型）进行情感标注和训练，最终建立一个能够自动分析豆瓣电影评论情感的系统。这个系统可以为用户提供对电影观影体验的更深层次理解，同时也为电影从业者提供了对影片口碑的直观认知。

2 模块设计

数据爬取： 获取豆瓣TOP250电影的评论。
数据标注： 将爬取的文本数据进行清洗、合并、统计，生成标准的CSV数据集。
模型训练： 基于bert-base-chinese模型进行文本情感分析，训练模型用于评论情感判断。

3. 开发环境

运行环境： (Colab) Python 3、T4 GPU
(本地) Python 3.9.18、Pytorch 2.1.1、CUDA 12.2、NVIDIA GeForce RTX 3060 Laptop GPU
开发语言： 整个项目的主要开发语言是 Python，同时涉及到深度学习模型的训练，使用了 PyTorch 框架。

4. 具体实现

4.1 数据爬取

数据爬取模块的主要任务是通过Python爬虫实现对豆瓣TOP250电影的评论数据的获取。该模块通过模拟浏览器行为，使用Selenium和Requests库，从豆瓣电影页面中获取电影的评论数据。以下是该模块的详细实现分析。

1. 爬取豆瓣TOP250电影的评论URL

```
base_url = 'https://movie.douban.com/top250'
params = {'filter': ''}
```

- 'base_url': 豆瓣TOP250电影页面的URL。
- 'params': URL的参数，这里是过滤条件。

```
movie_list=[]
for page in range(0, 250, 25):  # 从第一页到第五页，每页递增25
    params['start'] = page
    r = requests.get(base_url, params=params, headers=headers)
    soup = BeautifulSoup(r.text, 'lxml')

    movie_items = soup.find_all('li')
    for movie_item in movie_items:
        if movie_item.find('div', class_='pic') is not None:
            movie_url = movie_item.find('a').get('href')
            movie_list.append(movie_url+'comments/')
```

- 循环遍历每页，获取电影的URL。
- 'movie_url': 获取每部电影的URL，并将其加入'movie_list'中。

2. 将电影URL写入文件

```
with open(url_file, 'w', encoding='utf-8') as f:
    for url in movie_list:
        f.write(url + '\n')
```

- 将获取到的电影URL写入文件，方便后续爬取评论时调用。

3. 从URL文件中逐个提取评论

```
    movie_list = [line.strip() for line in f.readlines()]

i = 1
for base_url in movie_list:
    print("正在爬取电影", i)
    i += 1
    # ...
```

- 循环遍历每个电影的URL，逐个提取评论。

4. 具体的评论提取实现

```
comment_items = soup.find_all('div',class_ = 'comment-item')

for comment_item in comment_items:
    span = comment_item.find('span',class_ = 'comment-info')
    username = span.find('a').text

    stars_element = comment_item.find('span',class_ =
re.compile(r'allstar.*rating'))
    if stars_element is not None:
        stars_class = stars_element.get('class')
        stars = stars_class[0].replace('allstar','')
    else:
        stars = None
```

```
    comment =  comment_item.find('span',class_ ="short").text

    with open(output_file, 'a', encoding='utf-8') as f:
        star = stars if stars else ""
        f.write(username + ' ' + star + ' ' + comment + '\n')
        f.write('\n')
```

- 使用BeautifulSoup解析HTML，提取评论的作者、星级和评论内容。
- 将提取的信息写入'output.txt'文件。

5. 反反爬措施

```
headers = {
'User-Agent':random.choice(user_agent)
}

symbol = True
time.sleep(random.random()*5)
```

- 在每次请求时，通过'random.choice'从预定义的'user_agent'列表中随机选择一个用户代理，模拟不同浏览器和设备的请求。
- 使用'time.sleep'函数在每次请求之前随机等待一段时间，以模拟人类的操作行为，并防止被网站识别为爬虫。

4.2 数据标注

数据标注模块的主要任务是将从豆瓣电影评论中爬取到的原始数据进行清洗、整理，并转换成标准的CSV数据集。此外，还对数据进行统计、分类等处理，以便为模型训练提供高质量的输入数据。

1. 数据处理 - main.py

```
# 使用正则表达式处理数据
pattern = re.compile(r'(\S+)\s+(\d{2,3})\s+([\s\S]*?)(?=\n\S|$)')
matches = pattern.findall(raw_data)

# 处理数据，删除换行符和空白行
processed_data = [(match[0], match[1], re.sub(r'\s+', ' ', match[2].replace('\n',
'''))) for match in matches if match[2].strip()]
```

- 通过正则表达式处理原始数据，提取评论的id、评分和内容。
- 删除换行符和空白行，保留有效的评论信息。

2. 数据过滤 - toclean.py

```
# 找到评分列的索引
rating_index = header.index('评分')

# 过滤评分不在指定范围内的行
filtered_rows = [row for row in reader if row[rating_index] in ['10', '20', '30',
'40', '50']]
```

- 读取CSV文件，找到评分列的索引。
- 过滤掉评分不在指定范围内的行。
- 将过滤后的数据写入新的CSV文件,得到数据集clean1。

3. 文本清洗 - toclean1.py

```
def clean_text(text):
    # 如果text不是字符串，直接返回
    if not isinstance(text, str):
        return text

    # 使用正则表达式匹配中文、英文和标点符号
    cleaned_text = re.sub(r'[^\u4e00-\u9fffA-Za-z\s，。！？：；、]+', '', text)

    return cleaned_text.strip()  # 去除首尾空白字符

def clean_csv(input_file_path, output_file_path):
    # 读取CSV文件
    df = pd.read_csv(input_file_path)

    # 清洗文本数据
    for column in df.columns:
        df[column] = df[column].apply(clean_text)

    # 删除包含空值的行
    df = df.dropna()
```

- 读取CSV文件。
- 使用正则表达式清洗文本数据，去除非中文、英文和标点符号的字符。
- 删除包含空值的行。
- 将清洗后的数据保存到新的CSV文件，得到数据集clean2。

4. 数据合并 - concat.py

```
# 读取两个CSV文件
df1 = pd.read_csv('clean1.csv')
df2 = pd.read_csv('clean2.csv')

# 合并两个数据集
merged_df = pd.concat([df1, df2], ignore_index=True)
```

- 合并clean1和clean2两个数据集，忽略原有索引。

5. 数据分类 - classify.py

```
# 针对每个评分随机抽取400条数据作为测试集
test_data_list = []
remaining_data_list = []

for rating in [1, 3, 5]:
    # 选取指定评分的数据
    subset_data = train_data[train_data['评分'] == rating]

    # 随机抽取400条数据
    test_subset, remaining_subset = train_test_split(subset_data, test_size=800,
random_state=42)

    # 将抽取的数据添加到测试集列表
    test_data_list.append(test_subset)

    # 将剩余的数据添加到剩余数据集列表
    remaining_data_list.append(remaining_subset)

# 将测试集和剩余数据集合并
test_data = pd.concat(test_data_list)
remaining_data = pd.concat(remaining_data_list)
```

- 读取原始数据集。
- 针对每个评分，从数据中随机抽取400条数据作为测试集，剩余数据作为剩余数据集。
- 将测试集和剩余数据集合并。

6. 空数据删除 - delete.py

```
# 删除'id'列
data.drop(columns=['id'], inplace=True)  # 如果'id'列已经被删除，需要注释掉这行代码

# 删除内容为空的行
data.dropna(subset=['内容'], inplace=True)
```

- 删除'id'列。
- 删除内容为空的行。

7. 数据统计 - tongji.py

```
# 统计每个评分的数据条数
count_by_rating = df['评分'].value_counts().sort_index()

# 打印结果
print(count_by_rating)

# 映射评分
```

```
df['评分'] = df['评分'].map({10: 1, 20: 1, 30: 3, 40: 3, 50: 5})

# 保存修改后的数据到新的CSV文件
df.to_csv('train_data_3type.csv', index=False)

# 统计每个评分的数据条数
count_by_rating = df['评分'].value_counts().sort_index()

# 打印结果
print(count_by_rating)
```

- 统计每个评分的数据条数。
- 打印结果。
- 映射评分。
- 保存修改后的数据到新的CSV文件。
- 重新统计每个评分的数据条数。

4.3 模型训练

利用预训练的bert-base-chinese模型，使用PyTorch搭建数据加载器，定义性能评估指标。通过优化算法和超参数，训练bert-base-chinese模型进行文本情感分析，从而为评论进行情感判断提供准确性和泛化性。

4.3.1 Copy_of_my_pro2.ipynb（模型训练）

1. 读取、分析数据

```
data_frame=pd.read_csv('../data/test_data1_3type.csv',
                       names=['category', 'text'],
                       index_col=False
                       )
possible_categories=data_frame.category.unique()
```

- 从CSV文件中读取数据，包含两列：'category'和'text'。
- 获取数据集中不同的分类标签。

2. 加载预训练模型

```
BERT_PATH = '../Models/bert-base-chinese'
tokenizer = BertTokenizer.from_pretrained(BERT_PATH)
model = BertForSequenceClassification.from_pretrained(
    BERT_PATH,
    num_labels = len(label_dict),
    output_attentions=False,
    output_hidden_states=False
    )
```

- 下载中文预训练模型bert-base-chinese，加载Tokenizer。
- 加载模型。

3. 创建DataLoader和Optimizer and Scheduler

```
dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
                              batch_size=batch_size)
dataloader_val = DataLoader(dataset_val,
                            sampler=RandomSampler(dataset_val),
                            batch_size=batch_size)
optimizer = AdamW(model.parameters(),
                  lr=1e-5,
                  eps=1e-8)
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,

num_training_steps=len(dataloader_train)*epochs)
```

- 创建训练集和验证集的DataLoader，用于模型训练。
- 使用AdamW优化器和学习率调度器。

4. 定义Performance Metrics

```
def f1_score_func(preds, labels):
    # ...

def accuracy_per_class(preds, labels):
    # ...
```

- 定义评价指标，包括F1分数和每个类别的准确率。

5. 定义训练模型

```
for epoch in range(epochs):
    # ...
    for batch in dataloader_train:
        # ...
        outputs = model(**inputs)
        loss = outputs[0]
        logits = outputs[1]
        # ...
    # ...
    save_model(epoch)
```

- 训练模型，迭代多个epoch，计算损失、更新参数，并保存训练过程中表现最好的模型权重。

6. 评估模型

```
_, predictions, true_vals = evaluate(dataloader_val)
accuracy_per_class(predictions, true_vals )
```

- 在验证集上评估模型性能，输出准确率和每个类别的准确率。

4.3.2 model_test.ipynb（模型测试）

前三步与4.3.1中模型训练类似

1. 读取、分析测试数据

2. 加载预训练模型

3. 创建DataLoader和Optimizer and Scheduler

4. 加载训练好的权重

```
save_path='./bert_checkpoint'
model.load_state_dict(torch.load(f'{save_path}/bert-base-
chinese_2type_39200_epoch_6.model'))
```

- 加载之前训练好的模型权重。

5. 输入单个文本进行预测分类

```
text_to_predict = "这个剧算是我2023年最大的惊喜，..."
predict_single_example_2(model, tokenizer, text_to_predict)
```

- 输入单个文本，使用训练好的模型进行分类预测。

6. 测试集上评估

```
_, predictions, true_vals = evaluate(dataloader_test)
accuracy_per_class(predictions, true_vals )
accuracy_score_func(predictions, true_vals)
recall_score_func(predictions, true_vals)
precision_score_func(predictions, true_vals)
```

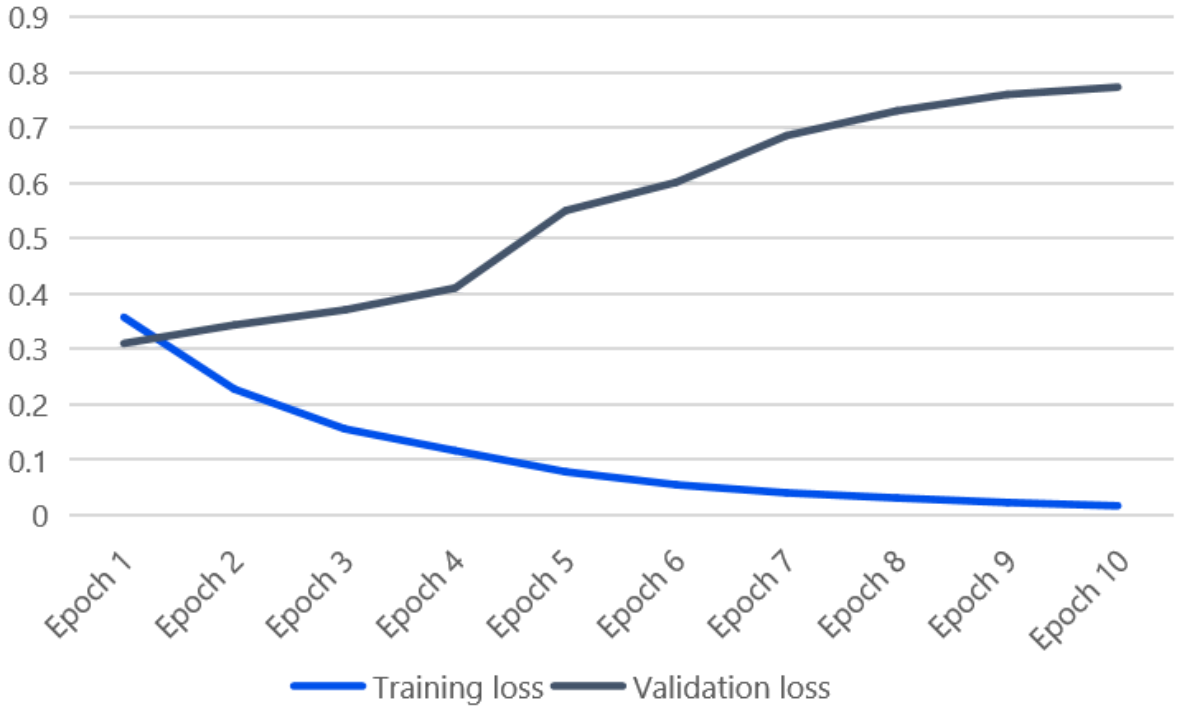
- 在测试集上评估模型性能，输出准确率和每个类别的准确率。

5. 团队分工

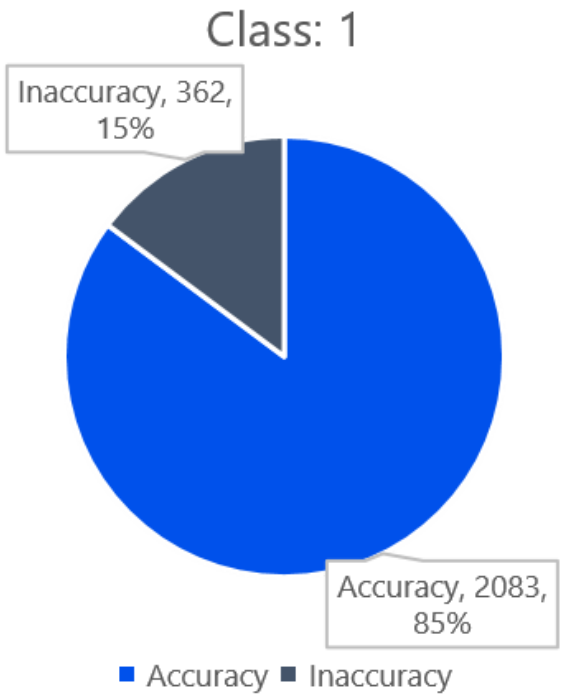
陈梓轩——数据爬取
符景乐——数据标注
丁华威——建立模型及训练、pre
陈意博——PPT制作及pre
郑宇轩——项目论文报告撰写

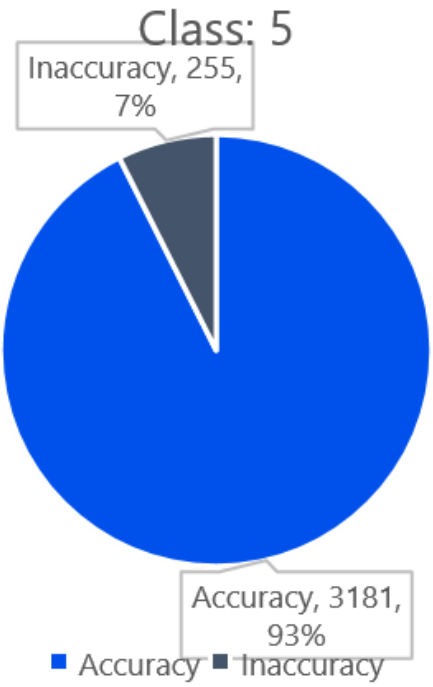
6. 结果分析

- 在lr=1e-5，epoch=10时，某次训练的结果
  我们发现验证损失逐渐增大，但模型在验证集上的预测准确率和 F1 Score上表现优异。增大的原因在于模型在早期就已经学到了一些训练数据的特征，但随着时间的推移，它可能过度拟合了训练数据，导致在验证集上的性能下降。

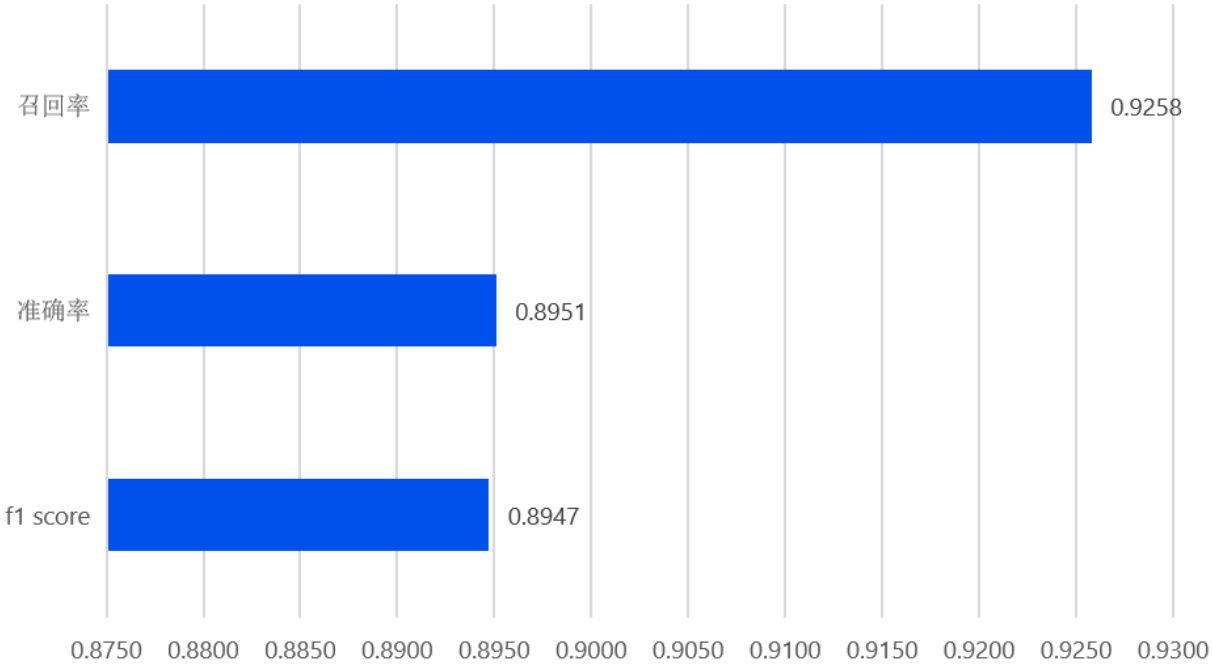


- 差评分类的准确率和好评分类的准确率：

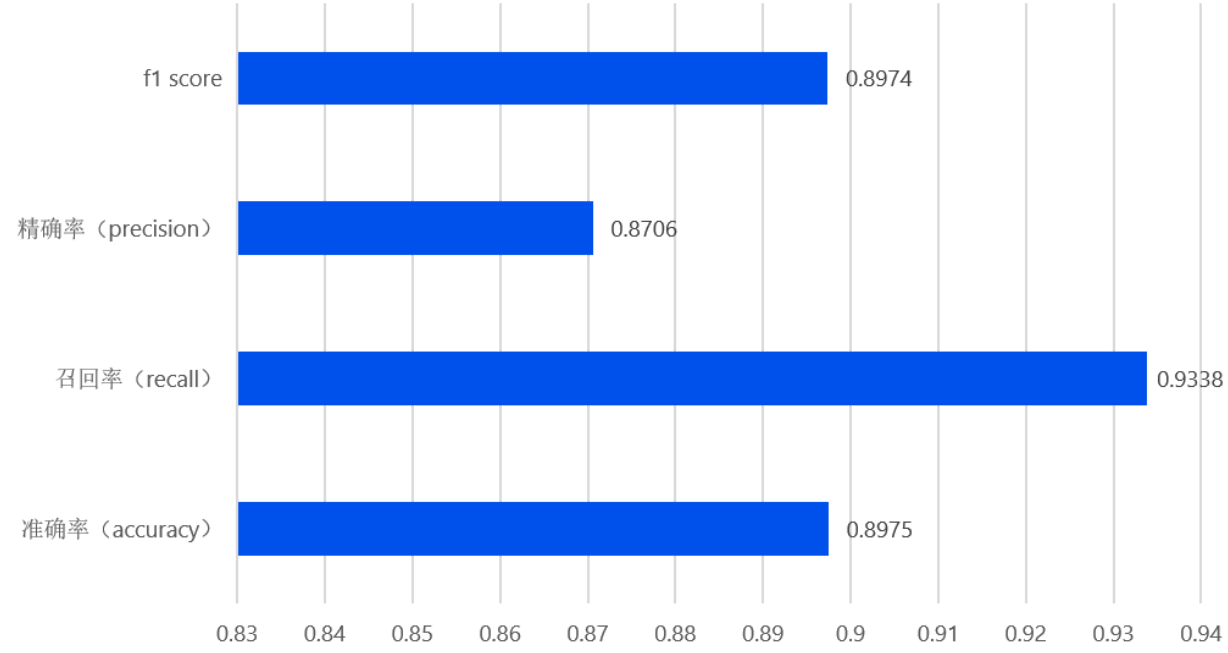




- 训练模型的准确率、召回率与f1 score：



- 测试模型的准确率、召回率、精确率与f1 score：



7. 项目总结

在“豆瓣电影评论情感分析”项目中，我们利用Python爬虫技术，通过模拟浏览器行为，成功获取豆瓣TOP250电影的评论数据，并对其进行自动化标注。基于bert-base-chinese模型，成功建立了一个情感分析模型，实现了一个完整的情感分析系统，能够较准确地对电影评论进行情感分类，为用户提供更深层次的电影观影评估。

未来，我们的项目还可以进行一些改进和扩展：

- 模型性能提升： 可以通过更大规模的数据集进行训练，或者尝试其他预训练模型，以提高情感分析模型的性能。
- 多语言支持： 目前项目主要以中文为主，但未来可以考虑支持多语言，使得系统具备更广泛的应用范围。
- 用户界面优化： 对系统的用户交互界面进行进一步优化，使用户能够更直观、方便地使用情感分析系统。

在这个项目中，我们对自然语言处理（NLP）和深度学习有了更深层次的认识，掌握了Python爬虫库（如Selenium和Requests）进行数据爬取的基本原理和实现方式，学习了情感分析的基本原理，以及如何使用深度学习模型（bert-base-chinese）进行情感分析任务、如何使用预训练的深度学习模型。我们不仅仅是学习者，还是实践者。我们必须将我们所学的知识转化为实际的功能解决方案。这需要创造力和解决问题的能力，这也是我们在这个项目中得到的宝贵经验之一。

此外，团队合作也是这个项目中不可或缺的一部分。通过这个项目，我们学会了如何更好地进行团队成员之间的合作，实现共同的目标。我们必须协调工作，合理分工，确保各个模块能够协同工作，最终构建出一个完整的系统，这要求我们具有很强的沟通和协调能力，也需要相互信任和支持。