数据分析基础(Python)

副标题：大模型和 AI 的应用：基于 BERT 的聊天机器人、生物信息安全与 GUI 设计

日期: 2024-12-10

小组成员:

1. 杨金源+302023510166
2. 郑华舰+302023510105
3. 王鎏清+302023510172
4. 林晓德+302023503377
5. 徐佳坤+302023503027

任务分配:

王鎏清：程序开发和模型训练

郑华舰：报告撰写

杨金源：设计 GUI

林晓德：数据标注和答辩 ppt 制作

徐佳坤：数据清洗和模型评估

I. 课题背景介绍

1.1 自然语言处理的发展与挑战

自然语言处理（NLP）作为人工智能领域的一个重要分支，其目标是使计算机能够理解、生成和响应人类语言。随着互联网和社交媒体的爆炸性增长，我们面临着前所未有的语言数据量，这为 NLP 的发展提供了丰富的资源，同时也带来了新的挑战。如何从海量数据中提取有价值的信息，如何提高机器对语言的理解和生成能力，成为了 NLP 领域研究的核心问题。

1.2 BERT 模型的革命性影响

在 NLP 的发展历程中，预训练语言模型的出现标志着一个重要的转折点。特别是 BERT（Bidirectional Encoder Representations from Transformers）模型[1]，它通过引入双向编码机制，使得模型能够更全面地理解语言的上下文信息。BERT 模型在多项 NLP 任务中取得了突破性成果，包括文本分类、情感分析、问答系

统等，其性能在多个基准测试中超越了以往最好的模型[3]。

1.3 聊天机器人与文档分析的需求增长

随着人工智能技术的普及，聊天机器人和文档分析工具的需求日益增长。聊天机器人能够提供 24/7 的客户服务，处理用户咨询，甚至进行情感交流。文档分析则能够帮助用户快速理解大量文本数据，提取关键信息，提高工作效率。在本项目中，我们采用了零一万物提供的 yi-large API 接口，这是一个基于 BERT 模型的高级接口，它提供了强大的聊天功能和文档分析能力。通过这个接口，我们能够实现一个基础的聊天机器人，它不仅能够理解用户的查询，还能够分析文档内容，提供深入的见解和摘要[2]。

1.4 项目的研究动机与目标 本项目旨在探索如何利用 BERT 模型的强大能力，开发一个高效、准确的聊天机器人和文档分析系统。我们的目标是实现一个能够进行情感分析、理解用户意图、并提供文档内容摘要的智能系统。为了提高系统的安全性和用户体验，我们还将集成生物信息解锁功能，并设计一个用户友好的图形用户界面（GUI）。

1.5 研究的创新点与预期贡献 本研究的创新点在于将 BERT 模型与实际应用场景相结合，通过零一万物的 yi-large API 接口实现聊天功能和文档分析能力。此外，生物信息解锁功能的集成为系统提供了额外的安全层，而 GUI 的设计则使得非技术用户也能轻松使用。我们预期，本项目将为 NLP 领域提供新的应用案例，推动聊天机器人和文档分析技术的发展。

II. 数据集描述

1. 测试集（Test Set）描述： 测试集中共有 4416 条评论，情绪分布如下：愤怒情绪的评论有 853 条，占比约为 19.3%；积极情绪的评论有 1616 条，占比约为 36.6%；消极情绪的评论有 1270 条，占比约为 28.8%；中性情绪的评论有 852 条，占比约为 19.3%。微博对特定愤怒情绪（如脏话）有一定程度的屏蔽，导致爬取的评论中愤怒情绪占比较低。可以预见的是，训练出来的模型对于脏话的情感分析能力并不出众，但是项目集成的脏话检测功能，将此影响减少到最低。

2. 训练集（Train Set）描述： 训练集中包含 361745 条评论，情绪分布如下：愤怒情绪的评论有 55267 条，占比约为 15.3%；积极情绪的评论有 199497 条，

占比约为 55.1%；消极情绪的评论有 51714 条，占比约为 14.3%；中性情绪的评论有 55267 条，占比约为 15.3%，具体分布详见图 2。

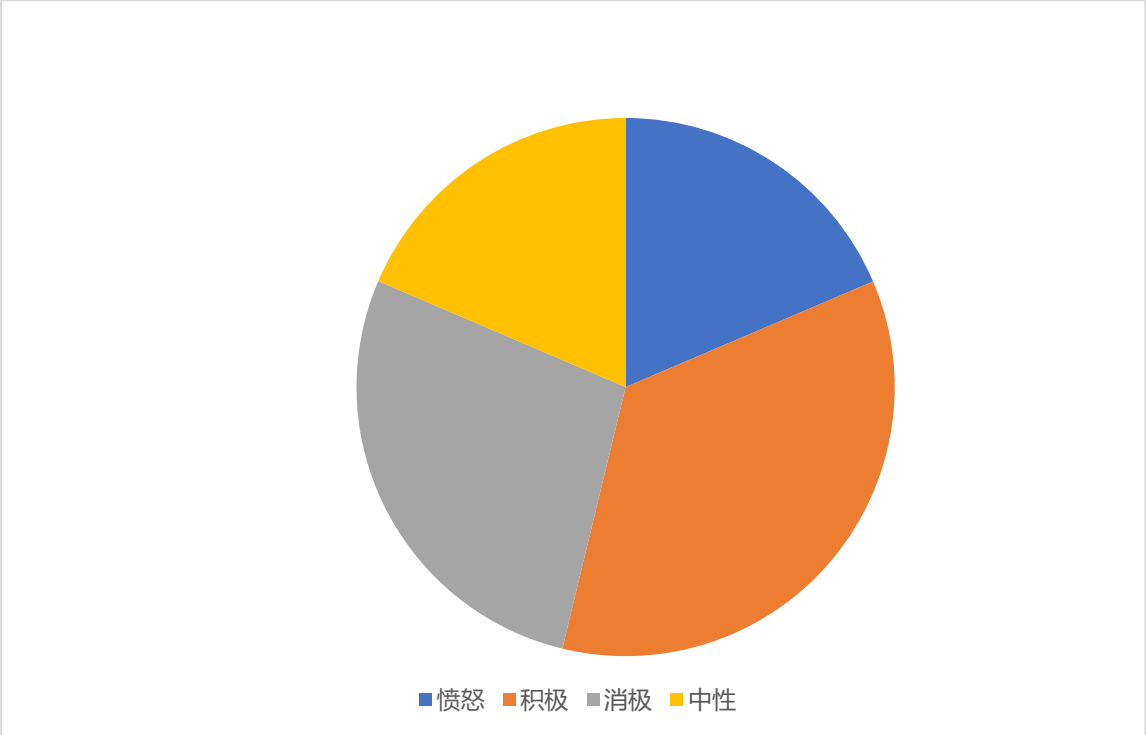

图 1 test 集数据分布图

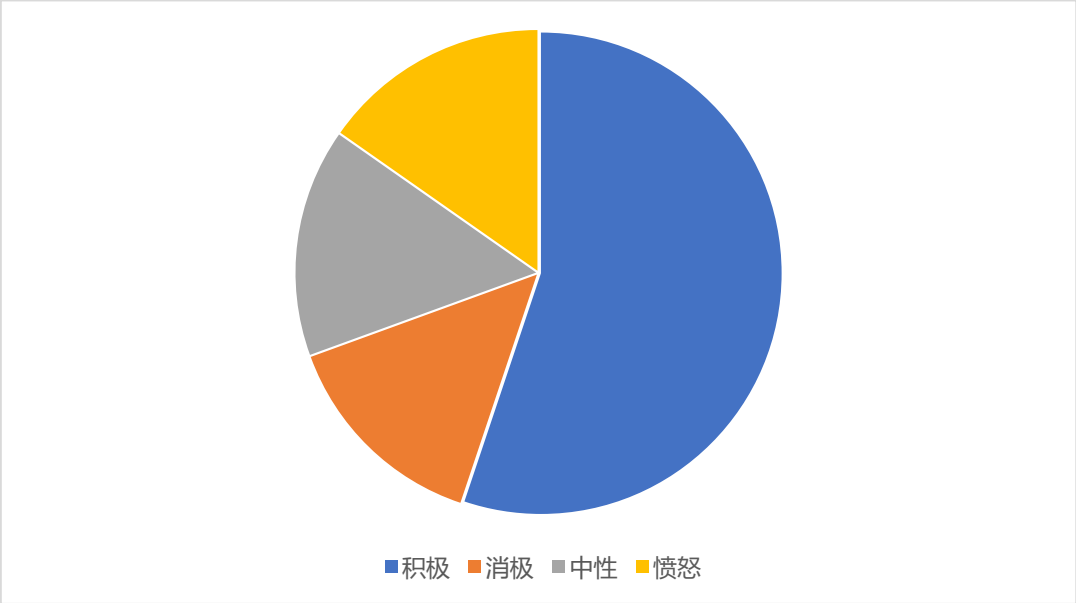

图 2 train 集数据分布图

III. 方法与结果

1.爬虫实现方法：

在开发初期，本项目采用了 SnowNLP 库进行语义分析。然而，由于其准确

率未达到预期，且现有模型无法充分满足项目需求，我们决定自行训练模型。为了提高开发效率，我们选择了公开的 simplifyweibo_4_moods 数据集作为训练集。此外，为了全面评估模型的性能，我们通过爬取微博热搜评论数据，并对其进行手工清洗和标注，构建了一个额外的测试集。

在爬虫开发过程中，我们面临了多重挑战：

1. 缺乏自动化请求机制，导致每次爬取都需要手动修改请求参数，如微博帖子 ID，这严重限制了爬取过程的自动化和效率。
2. 未将评论数据封装为对象，这增加了数据处理和维护的复杂性。
3. 原始代码将数据直接写入 CSV 文件，这种存储方式导致了较低的读写速度和较高的延迟。
4. 代码缺乏模块化，这可能对代码的维护和扩展带来困难。
5. 在写入 CSV 文件时，程序中断可能导致数据不完整或文件损坏。
6. 爬取的数据质量参差不齐，部分数据对模型训练无益，需要进行清洗。

为了解决这些问题，我们对爬虫进行了迭代改进改进后的伪代码如下：

Algorithm 1 CrawlTask

Input: None

Output: Crawled data stored in Redis

Begin:

1: Initialize Redis connection

```
redis_conn = RedisUtil.get_redis_connection()
```

2: Define constants for list sizes and key expiration

```
HOT_LIST_SIZE = 30
WB_LIST_SIZE = 20
CM_LIST_SIZE = 50
KEY_EXPIRE_TIME = 60 * 60
```

3: Start the crawling process

```
start_time = current_time()
print("Starting crawl at", start_time)
try:
    crawl()
```

```
        catch Exception as e:
            print("Crawl task failed:", e)
        end_time = current_time()
        print("Crawl completed at", end_time, "Duration:", (end_time - start_time))
```

4: Crawl function to fetch and store data

```
    FUNCTION crawl():
        timestamp = current_timestamp()
        url = "https://m.weibo.cn/api/container/getIndex?..."
        hot_list = WeiboParser.get_hot_list(url, HOT_LIST_SIZE)
        FOR each hot in hot_list DO
            hot_key = create_key(timestamp, hot)
            insert_hot(hot_key, hot)
            weibo_list = WeiboParser.get_weibo_list(hot, WB_LIST_SIZE)
            FOR each weibo in weibo_list DO
                wb_key = create_key(hot_key, weibo)
                insert_weibo(wb_key, weibo)
                comment_list = WeiboParser.get_comment_list(weibo, CM_LIST_SIZE)
                cm_key = create_key(wb_key, "cm")
                FOR each comment in comment_list DO
                    score = classify_comment(comment.text)
                    comment.score = score
                    insert_comment(cm_key, comment)
                END FOR
                set_key_expiration(cm_key, KEY_EXPIRE_TIME)
                update_weibo_stats(wb_key, comment_list)
                sleep(3)
            END FOR
```

```
                calculate_hot_status(hot_key, weibo_list)
            END FOR
            store_timestamp("timestamp", timestamp)
    5: Insert hot data into Redis
        FUNCTION insert_hot(key, hot):
            hot_data = {"desc": hot.desc, "scheme": hot.scheme}
            redis_conn.hmset(key, hot_data)
            redis_conn.expire(key, KEY_EXPIRE_TIME)
            print("Hot Insert:", hot.desc)
    6: Insert weibo data into Redis
        FUNCTION insert_weibo(key, weibo):
            weibo_data = {"id": weibo.id, "url": weibo.url, ...}
            redis_conn.hmset(key, weibo_data)
            redis_conn.expire(key, KEY_EXPIRE_TIME)
    7: Insert comment data into Redis
        FUNCTION insert_comment(key, comment):
            comment_json = json.dumps(comment.__dict__)
            redis_conn.rpush(key, comment_json)
    8: Classify comment sentiment
        FUNCTION classify_comment(text):
            RETURN len(text) % 3 - 1
    9: Update weibo statistics
        FUNCTION update_weibo_stats(key, comment_list):
   10: Calculate hot status
       FUNCTION calculate_hot_status(key, weibo_list):
End:
```

我们直接爬取微博热搜下的评论，通过调用`get_hot_list`方法，使用 HTTP 请求获取热搜条目的 JSON 数据，并将每个热搜条目封装为`Hot`对象。针对每个热搜话题，系统调用`get_weibo_list`方法获取相关微博列表，并通过构建 API 请

求 URL 来获取微博的详细信息，包括微博 ID、用户信息、发布时间和内容等。对于每条微博，系统还会获取完整的微博正文及发布时间。

系统通过`get_comment_list`和`parse_comment`方法获取微博的评论列表，并采用递归方式分页获取评论数据。爬虫解析 JSON 数据，提取评论文本、时间戳和点赞数等信息，并将每条评论封装为`Comment`对象。

在数据处理方面，我们对评论进行了手工标注，以判定语句的情感（积极、消极、中性、愤怒），并据此对数据进行清洗。我们使用 Eraser 工具从 dictionary 中读取清洗规则，以便在爬取过程中实时清洗数据，并允许动态添加新的清洗规则。

数据存储方面，我们采用 Redis 数据库，通过`RedisUtil.get_redis_connection`方法获取连接，并为热搜、微博和评论定义了键值存储方案。利用 Redis 的`hmset`和`rpush`命令存储数据，并设置过期时间以自动清理过期数据[6]。

最终，整个爬虫任务由`run_crawler.py`文件中的`CrawlTask`实例启动，通过调用`run`方法执行爬取过程。这一改进确保了数据的准确性、爬取的高效性以及存储系统的可靠性和数据的实时性。

2.训练方法：

我们选择了 BERT 模型[3]进行监督学习，是因为在有明确任务目标且具备标签数据的情况下，监督学习能够更高效地实现目标。与无监督学习相比，监督学习可以充分利用已有的标签信息，直接优化分类性能。

在传统的无监督学习中，由于没有标签，模型无法直接知道正确答案，因此通常需要依赖诸如聚类、降维等方法从数据中提取潜在模式。这种方法适用于没有标签的探索性分析,然而，对于情感分析等有明确任务目标的场景，监督学习显然更加合适,它能够通过标签信息来引导模型训练，从而高效地提高分类任务的准确性。

我们通过训练数据中的标签信息来指导模型学习。具体而言，模型的预测结果与真实标签之间的差异通过交叉熵损失函数量化，并利用反向传播来更新模型参数，从而优化分类性能。反向传播通过梯度下降法计算损失函数对模型参数的梯度，并调整模型参数，使得损失逐渐减小，模型的预测逐步接近真实标签。

在数据集中，我们注意到不同情感标签的样本数量可能存在不平衡问题，这可能导致训练时模型偏向样本较多的类别。为了解决这个问题，我们采用了加权损失函数。具体来说，我们根据每个类别的样本数量，计算出类别的权重，并在交叉熵损失函数中进行加权调整。这样，训练过程中，模型会对样本较少的类别给予更高的关注，从而提高模型在这些类别上的表现，具体加权的实现方式如下：

Algorithm 2 Weighted Loss and Training Step

Input: Model outputs logits, true labels, and class weights tensor

Output: Loss value for training step

Begin:

FUNCTION weighted_loss_function(logits, labels, class_weights_tensor):

1: class_weights_tensor = class_weights_tensor.to(logits.device)

2: loss = CROSS_ENTROPY_LOSS(logits, labels)

3: weighted_loss = loss * class_weights_tensor[labels]

4: RETURN weighted_loss

FUNCTION training_step(model, inputs, class_weights_tensor):

5: model.train()

6: labels = inputs.pop("labels")

7:features = inputs.to(model.device)

8: outputs = model(features)

logits = outputs.logits

9: loss = weighted_loss_function(logits, labels, class_weights_tensor)

10: loss.backward()

11: RETURN loss

End:

3.聊天机器人实现方法：

（1）聊天交互：GptAssistant 类初始化一个客户端，用于与聊天模型进行交互。对话历史被维护在一个列表中，以便于生成上下文相关的回复。API 发送对话请求，并返回 AI 的回复。AI 的回复也被添加到对话历史中，以便于未来的交

互能够保持上下文的连贯性。

（2）脏话识别：考虑到项目运行效率问题，我们选择了使用本地词典而不是已有的 api。程序接收用户输入的文本和本地词典作为参数，将文本转换为小写后，检查是否包含词典中的任何脏话。当检测到用户输入脏话，对话框将输出警告并且不会发送该消息。

（3）文档分析：我们使用 QFileDialog.getOpenFileName 方法打开一个文件选择对话框，让用户选择 Word 文档。这个方法返回一个元组，包含文件名和选中的过滤器，但这里只关心文件名，所以第二个返回值被忽略（用下划线_表示）。然后我们使用了 Doc_reading.read_docx_file(file_path)方法，从指定路径读取 Word 文档，并提取其中的文本内容。通过使用 python-docx 库，它遍历文档的所有段落，并将这些段落的文本合并成一个单一的字符串，从而实现文档内容的完整提取。然后我们使用了 ChatApp.analyze_document()方法对 Word 文档进行分析。首先，它弹出一个文件选择对话框，用户通过此对话框选择一个 Word 文档后，Doc_reading.read_docx_file 方法被调用来读取并提取文档内容。如果文档内容成功读取，该方法将内容发送至 AI 进行总结。最后，将 AI 生成的总结结果显示在聊天窗口中。

通过这些功能的实现，聊天机器人不仅能够与用户进行自然的对话交互，还能够理解和处理用户上传的文档，提供有价值的信息摘要。同时，通过脏话识别和情绪分析，聊天机器人能够维护一个健康和积极的对话环境。

4. 人脸识别：

我们选择使用 OpenCv 进行面部识别功能，在图像处理方面我们使用了以下方法[4]：

（1）二值化：二值化是将之转换为是黑白图像。这有助于简化图像，突出人脸的轮廓和特征，减少背景噪声，使得人脸检测算法更容易识别出人脸区域。二值化的公式如下：

$$THRESH\_BINARY(x,y) = \begin{cases} max\,val, & 当 src(x,y) > thresh \\ 0, & others \end{cases} \quad (4\text{-}1)$$

（2）降噪：采用高斯滤波降噪，这在可以去除暗光环境下产生的噪点。高斯滤波是一种常用的图像处理技术，用于对图像进行平滑处理，减少图像中的噪

声[5]。高斯滤波的公式如下：

$$\sigma = 0.3 \times (\frac{ksize-1}{2} - 1) + 0.8 \quad (4\text{-}2)$$

（σ：高斯分布的标准差。ksize：高斯核的大小。）

$$Gi = \alpha \cdot e^{-\frac{(i-\frac{ksize-1}{2})^2}{2\sigma^2}} \quad (4\text{-}3)$$

(Gi：高斯核中的第 i 个元素。α：归一化因子，确保高斯核中所有元素的和为。i：当前元素的索引，范围从 0 到 ksize−1）

在比较人脸方面我们使用了直方图比较，直方图比较是一种评估两幅图像相似度的方法，通过比较它们的直方图来实现。在人脸比较中，直方图比较可以用来评估捕获的人脸与注册人脸的相似度，从而判断是否为同一人。

结果：在本研究中，我们开发的文本分析模型在情感分析任务上取得了85.24%的准确率，这一结果证实了模型在大多数情况下具备精确识别和分类情感的能力。通过分析模型的 ROC 曲线，我们进一步评估了其对不同情感类别的区分性能。对于积极、消极和中性情感，模型的 AUC 值分别达到了 0.95、0.89 和 0.91，这些数值均表明模型在识别这些情感类别时具有高准确性。然而，对于愤怒情绪的识别，模型的 AUC 值为 0.78，这是所有情感类别中最低的，暗示了模型在区分愤怒情绪方面的性能相对较弱。此外，通过混淆矩阵的分析，我们观察到模型在愤怒情绪的分类上存在一定的误差，这与 ROC 曲线分析的结果相一致。这些发现为我们提供了宝贵的洞见，指出了模型性能的不足之处，并为未来的模型改进和优化工作指明了方向。

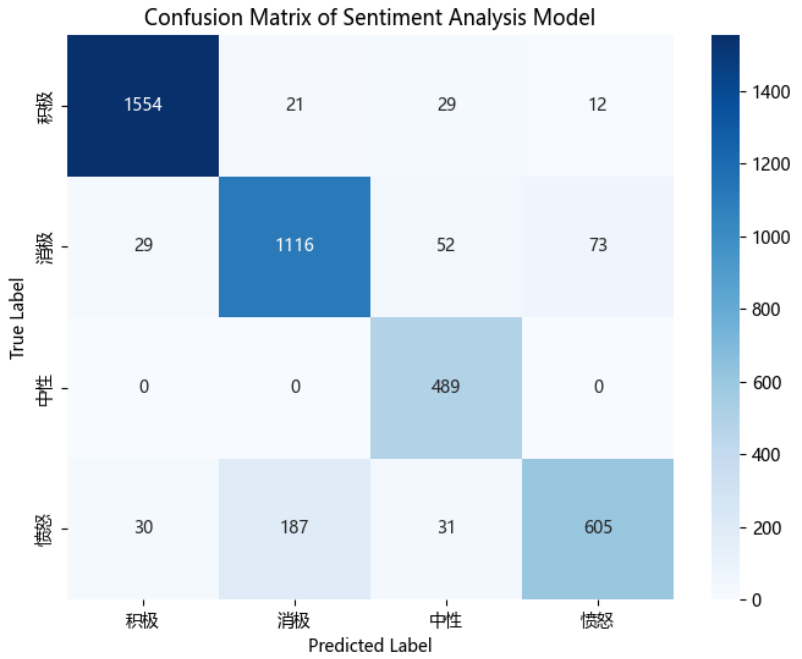

图 3 混淆矩阵

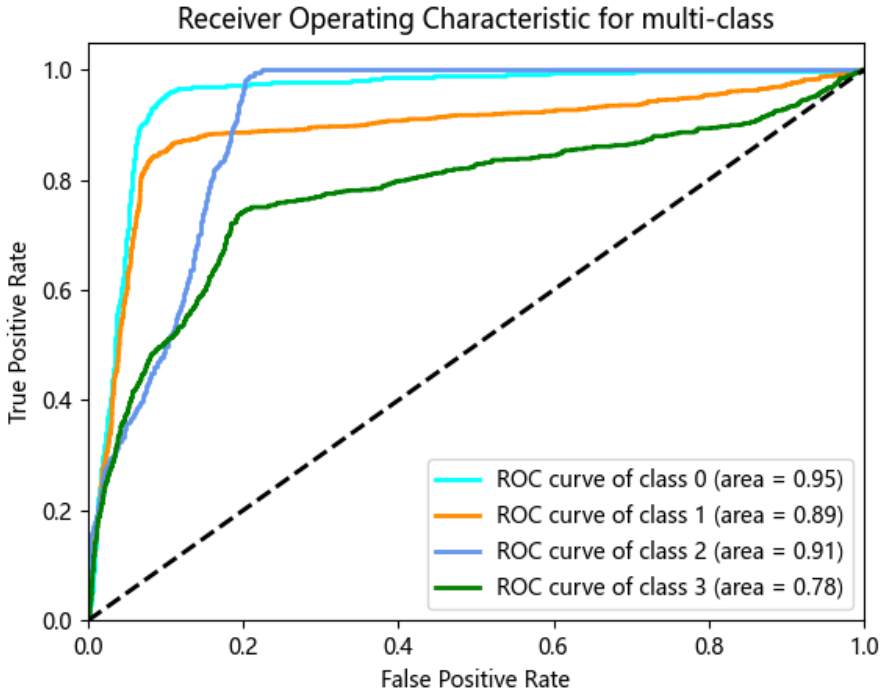

图 4 ROC 曲线

IV. GUI 设计

1. 窗口介绍：

窗口标题：窗口的标题为“智能聊天助手”，表明了应用程序的功能。

消息显示区域：GUI 的主要部分是一个文本区域，用于显示用户和聊天机器人之间的对话。在这个区域，用户可以看到自己的输入（如“哈哈哈”）以及聊天机器人的回复（如“AI：哈哈，看来你很开心！有什么好消息或者有趣的事情想要分享吗？或者有什么我可以帮助你的？”），如图 5。

输入框：在消息显示区域下方，有一个输入框，用户可以在这里输入自己的问题或消息。

按钮：“发送”按钮：用户输入消息后，点击此按钮将消息发送给聊天机器人。

“退出”按钮：点击此按钮可以关闭应用程序。

情绪分析结果：在 GUI 的底部，有一个区域显示了聊天机器人对用户输入的情绪分析结果及其置信度。例如，“情绪分析结果：积极 置信度：0.86”表示聊天机器人分析用户的情绪为积极，并且有 86%的置信度，如图 5。

脏话识别：聊天机器人具备脏话识别功能，当用户输入不恰当的语言时，聊天机器人会提示用户避免使用不恰当的语言，并不会发送该消息，如图 6。

“分析文档”按钮：用户可以通过点击此按钮上传并分析文档，聊天机器人将提供文档的总结或分析结果。

文档分析功能：当用户点击“分析文档”按钮并选择一个文档后，聊天机器人将读取文档内容并进行分析。分析结果将显示在消息显示区域中，如实验报告的总结，如图 7。

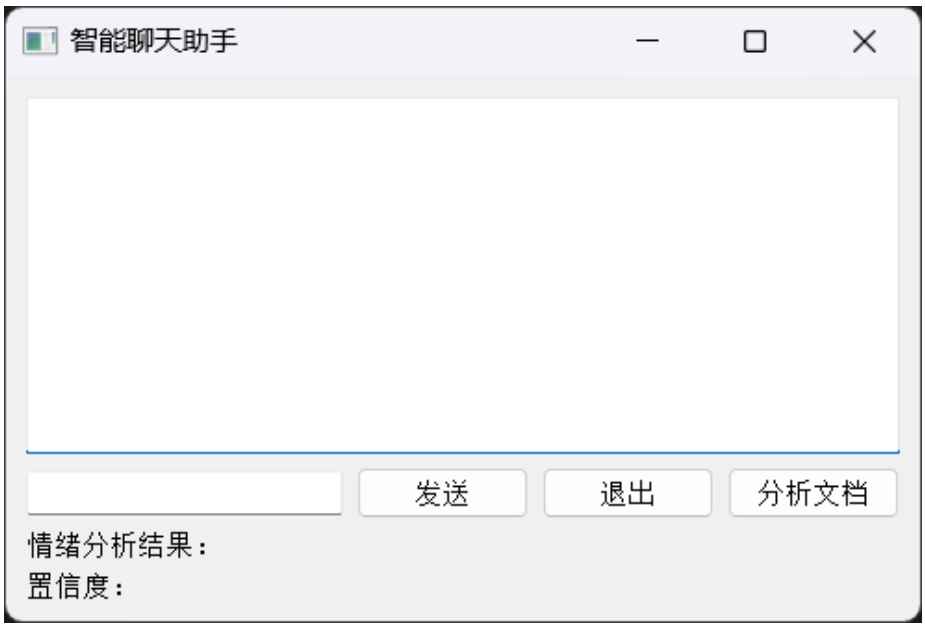

图 4 初始界面图

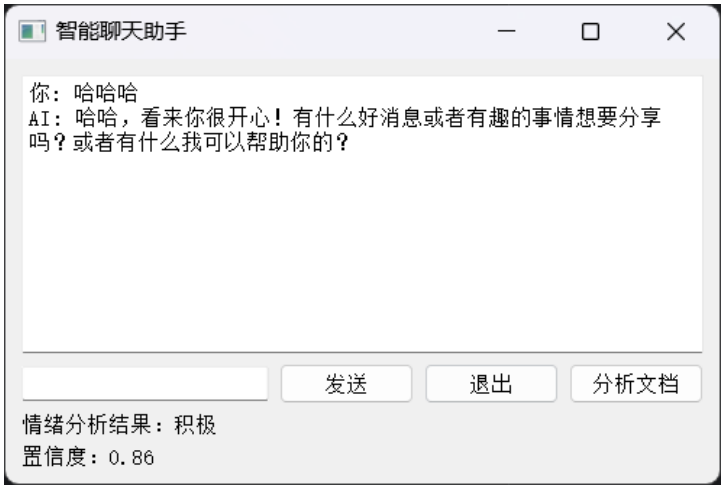

图 5 聊天示例图

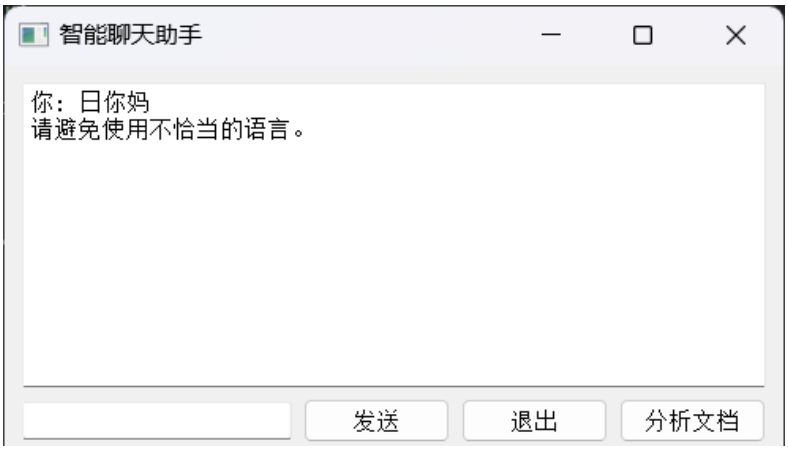

图 6 脏话输入示例图

图 7 文档分析示例图

VI. 总结与讨论

总结：

在本项目实施过程中，我们的团队掌握了多项关键技术，具体包括：

1. 数据爬取与处理：我们开发了一套高效的爬虫系统，用于从微博平台抓取评论数据。这些数据经过清洗和处理后，形成了适合训练模型的数据集。

2. 模型训练：利用 BERT 模型，我们训练了一个能够满足特定需求的文本分析模型。该模型在情感分析任务中表现出色，达到了 85.24%的准确率。

3.情感分析实现：通过训练好的模型，我们实现了情感分析功能，能够准确识别和分类文本中的情感倾向。

4.人脸识别技术：采用 OpenCV 库，我们实现了人脸识别功能，为系统提供了生物信息安全保障。

5.聊天机器人开发：结合上述功能，我们成功开发了一个聊天机器人，它能够在多种语境下与用户进行互动，提供回复，检测并提示脏话，以及对用户上传的文档进行总结。

通过这些技术的整合，我们的聊天机器人展现出了全面的功能，包括但不限于对话回复、脏话检测、文档分析等，为用户提供了一个智能、安全且高效的交互体验。

讨论：

项目还存在以下不足：

1. 当前模型在识别愤怒与消极情绪的任务中表现出一定的不足。这一局限性可能源于模型设计上的缺陷以及训练过程中数据的不充分。为了提升模型的性能，未来的工作将聚焦于扩充和丰富训练数据集，特别是增加反映消极和愤怒情绪的样本。通过增强数据的多样性和覆盖范围，模型将能够更有效地学习并区分这些情感状态。此外，进一步的模型优化和训练策略的改进也是提高识别准确性的重要途径。

2. 目前人脸识别功能采用灰度转换方法以提升处理速度，但该方法在变化的光照条件下可能导致识别准确度下降。为解决此问题，我们提出以下改进措施：

(1) 实施图像预处理技术，包括直方图均衡化以增强图像对比度，以及伽马校正以调整亮度，从而在不同光照条件下提高特征可识别性。

(2) 采用多尺度分析策略，识别在多种分辨率下保持稳定的人脸特征，以增强识别系统的鲁棒性。

(3) 利用颜色信息作为辅助线索，以区分不同个体的人脸，提高识别的准确性。

(4) 通过计算并归一化图像的光照分布，减少光照变化对识别过程的影响。

此外，考虑引入深度学习模型，例如卷积神经网络（CNN），这些模型能够通过学习大量数据集，在不同光照条件下有效提取人脸识别特征，进一步提升系统的识别能力。

3. 本项目代码在编写规范性和版本控制方面存在若干不足，具体问题包括：

(1) 代码编写缺乏统一标准，影响了代码的整洁性和一致性。

(2) 版本维护措施不够严格，迭代过程中采取的是在旧代码基础上注释并添加新代码的方式，而非采用更为规范的版本控制流程。

(3) 缺少详尽的日志记录，这在追踪代码变更和问题调试过程中造成了不便。

(4) 代码封装不充分，导致代码的可读性和可维护性不足，增加了后期维护和升级的难度。

针对上述问题，我们认识到在未来的项目开发及代码编写过程中，必须采取以下改进措施：

(1) 遵循编码规范，确保代码风格的一致性。

(2) 实施严格的版本控制流程，利用版本控制系统（如 Git）来管理代码的变更历史。

(3) 引入日志记录机制，详细记录代码的变更和运行时的关键信息。

(4) 加强代码封装，提高代码模块化程度，以提升代码的可读性和可维护性。

这些改进将有助于提升项目的整体质量，确保代码的长期可持续性。

VII. 参考文献

[1] Jacob Devlin, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[2] 零一万物. 如何调用零一万物 API_大模型服务平台百炼(Model Studio)-阿里云帮助中心

[EB/OL].(2024-06-13)[2024-12-13].https://help.aliyun.com/zh/model-studio/developer-reference/yi-large-api.

[3] 张泽源,张光妲,李佳雨,等.基于 BERT 的日常文本情感分析[J].齐齐哈尔大学学报(自然科学版),2024,40(06):37-41.DOI:10.20171/j.cnki.23-1419/n.2024.06.005.

[4] 基于 OpenCV 的人脸识别算法研究与实现 https://doc.taixueshu.com/journal/20201473dnzsyjsxsb.html. 2020-06-11.

[5] 高斯滤波官方文档 https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#Mat%20getGaussianKernel%28int%20ksize,%20double%20sigma,%20int%20ktype%29

[6] Antirez. Redis 官方文档[EB/OL]. (2024-06-13)[2024-12-13]. https://redis.io/documentation.

VIII. 代码

Python 版本：3.1.2，

所 用 包 ： Openai-1.56.2 python-docx-1.1.2 transformers-4.46.3 torch-2.5.1 tensorflow-2.18.0 opencv-python-4.10.0.84 redis-5.2.0 pandas-2.2.3

matplotlib-3.9.3 scikit-learn-1.5.2 datasets-3.1.0
