

程序设计大作业--聊天机器人

成员：许元元 PB20511899 && 李艾颖 PB20051144 && 陆子睦 PB20051150

时间：2022/5/31

[toc]

分工设计

许元元： 基于GPT-2的NLP文本生成实现，辅助实现模式化功能（数学运算等部分）

李艾颖： GUI设计实现，协助爬虫相关模式化功能的实现（爬虫部分）

陆子睦： 总项目顶层文件的构筑，模式化功能的实现（包括机器人自我认知、推荐系统、爬虫部分）

分工比： 1 : 1 : 1

实现的功能：[Summary](#)

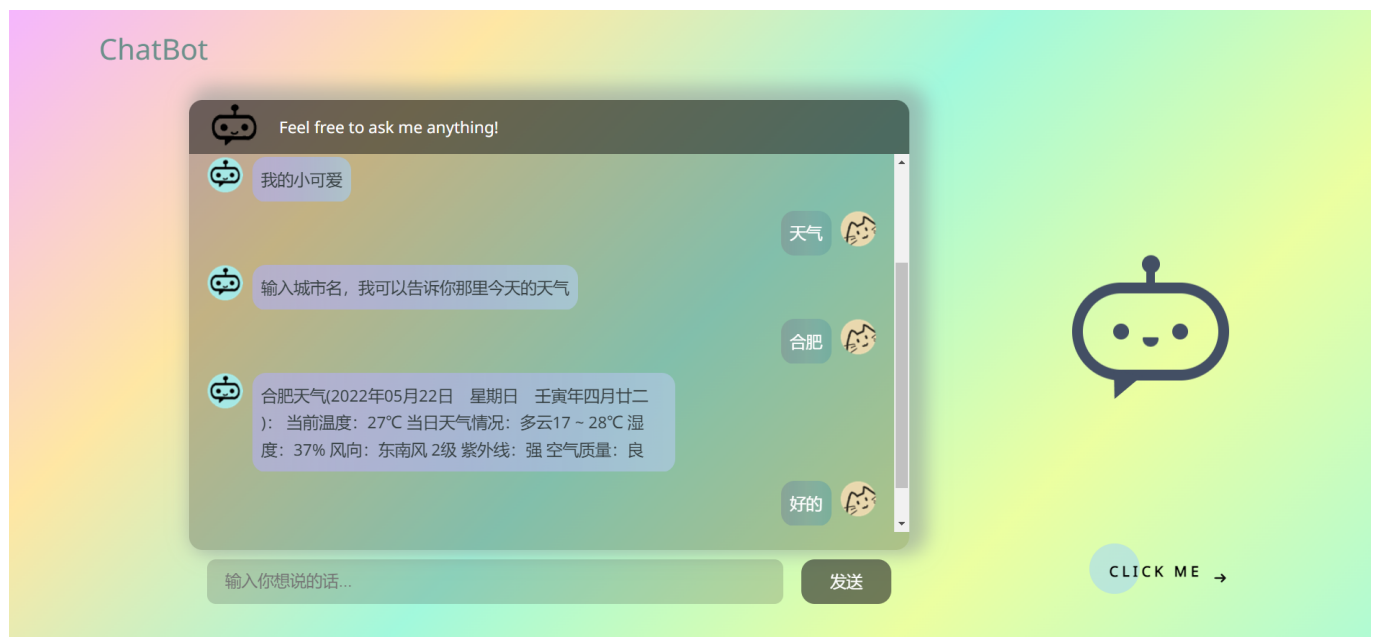
前期的讨论与设计：[模式化匹配功能设计](#)

后续的实现与设计时有一定的出入，但是效果基本一致。

GUI设计

web前端

聊天机器人的前端效果如下所示：



输入框

每次按下按钮'发送'，且输入框不为空时，即可向机器人发送一条数据。

```
<div class="b-footer">
  <input type="text" name="text" id="f-left" placeholder="输入你想说的
话..." />
  <div id="btn">发送</div>
</div>
```

对话框

- 发送完信息后，在对话框弹出该条文本，在机器人正在生成回答时显示加载动画

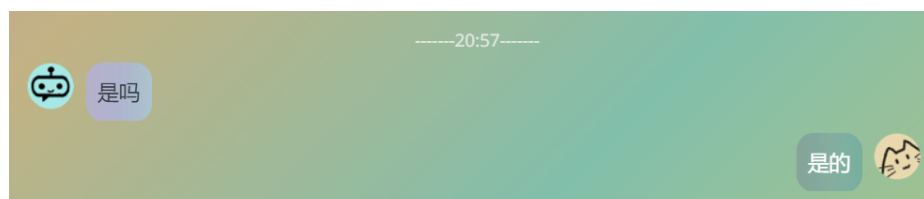
```
$(".b-body").append("<div class='mWord'><span><img src='\"p1.png\"' /></span><p>" +
text.val() + "</p></div>");
//在对话框'b-body'中将输入信息以mWord的格式添加新的Div标签，内容包含输入的文本信息和头像
图片
$(".b-body").append("<div class='wait'><span><img src='\"p2.png\"' /></span>
<p></p><div class='Ball'></div></div>");
//显示加载动画
$(".b-body").scrollTop(10000000);
//将对话界面拉到页面最下方显示最新对话
```

- 机器人将回复信息返回前端

```
function(result)
{
  $('.wait').remove();//删除加载动画标签
  if(result[0]=='<')//返回的内容为可以点击跳转的链接格式
    $(".b-body").append("<div class='rotWord'><span><img src='\"p.png\"' />
</span> " + result + "</div>");
  else//返回的内容为普通文本
    $(".b-body").append("<div class='rotWord'><span><img src='\"p.png\"' />
</span> <a id='member'>" + result + "</a></div>");
  $(".b-body").scrollTop(10000000);
}
```

- 时间信息

每隔一分钟显示一次当前时间信息，如图所示：



```
function setDate(){
    d = new Date();
    if (m != d.getMinutes())
        { //m为已记录的时间信息，在对话时每分钟更新m，在对话框中显示当前时间
            m = d.getMinutes();
            $(".b-body").append('<div class="timestamp">' + '-----' + d.getHours() + ':'
+ m + '-----' + '</div>');}
}
```

CSS渲染

- 静态渲染：
 - 背景和文本框的渐变色
 - 边框圆角效果
 - 对话框半透明效果
 - 阴影效果

```
background: -webkit-linear-gradient(0, #xxx 0, ..., #xxx 100%); /*渐变效果*/
color: rgba(55, 55, 55, 0.3); /*0.3为透明度*/
border-radius: 15px; /*圆角效果*/
box-shadow: 10px -5px 20px 10px #xxx; /*阴影效果*/
```

- 动态渲染
 - 加载动画,循环播放三个跳动的点



```
.dot {
    /*基本属性*/
    animation: pulse-outer 7.5s infinite ease-in-out; /*动画形式*/
}
.dot:nth-child(2) { /*第二个点*/
    left: 65px; /*偏移*/
    animation: pulse-inner 7.5s infinite ease-in-out;
    animation-delay: 0.2s; /*动画延迟*/
}
/*第三个点...*/
@keyframes pulse-outer {
    0% { transform: scale(1); }
    ...
    100% { height: 17.5px; }
}
```

- 新消息弹出动画：在产生新对话时，对新弹出的文本框进行一次缩放，实现弹出的效果

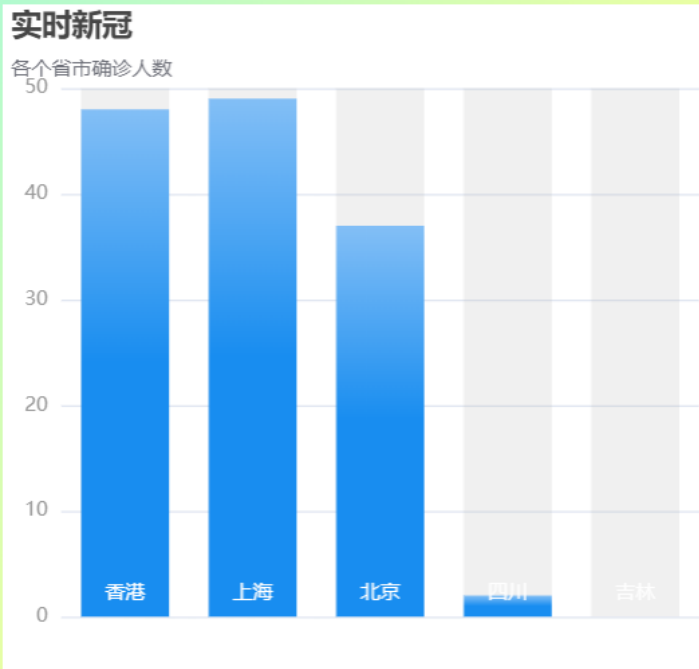
```
.Word{
  transform: scale(0);
  transform-origin: 0 0;
  animation: bounce 350ms linear both;
}
@keyframes bounce {
  0%{transform:matrix3d(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1); }
  ...
  100%{transform:matrix3d(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1); }
}
```

- 动态LOGO：参考了Dribbble的模型



其他

- echarts柱状图
 - 利用了echarts模型，当询问新冠疫情的信息时，可以弹出如图所示柱状图，更直观地观察数据



其支持交互功能，即将鼠标置于柱上会有强调显示，并且可以进行滚轮缩放，或是点击放大显示。

- 动态跳转按钮
 - 对于一些实时信息的查询，为了方便进一步了解详情，添加可以跳转的按钮



```
$("#btn2").click(function()
{
    if(is_ask==1)//正在查询信息
        window.location.href="https://xxx";//在新标签页打开链接
    else
        location.reload([bForceGet]); //停留在原页面
});
```

前后端交互

聊天机器人前后端的信息交互和同步部分是通过Flask框架和Ajax（异步JavaScript和XML）实现的。数据交互的基本模式如下：

Javascript部分：

```
$("#btn").click(function()
{
    chatbot();
});
//每次检测到"发送"按钮按下且输入不为空时触发
function(){
    var args= {url: "url",//与python函数对应的"URL"相同
               type: "GET",//"GET"或"POST"
               // "GET"用于后端向前端发送信息, "POST"用于将输入的文字传到后端
               success:function(result)
               {options();}}
    $.ajax(args);
}
```

python部分：

```
@app.route('/url')
def get_data():
    data=func()
    return data

@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html')
//默认显示的主页
```

当用户在前端输入想要对机器人说的话，便可利用JS中的Ajax工具向后端传输数据，并获取后端发的数据返回到聊天框中，这样就完成了一组对话；另外，对于新冠疫情确诊柱状图的更新，也是通过Ajax将爬取到的数据传入JS的函数中完成的。

模式化回答设计

信息爬取

本部分通过网络爬虫实现了实时信息的获取，比如查询每日各个省份疫情的情况以及天气情况，微博热点新闻。在输入端输入对应的查询请求后，聊天机器人便可以伪装成浏览器访问对应的网站爬取数据并挑选出需要的数据并返回。

新冠疫情

聊天人可以识别与“新冠疫情”相关的问题，自动爬取各个省份今日新冠疫情的新增情况，具体实现功能如下：

- 查询省份：可以对某个省份的疫情状况（新增确诊和新增无症状）
- 点击右下方链接即可跳转到相关网站查询详情
- 通过echarts柱状图显示各个省份的确诊情况，支持拖动和缩放

```
#alist 存储城市名，新增确诊和新增无症状
for i in citylist:
    if (i['today']['wzz_add'] == 0):
        clist.append([i['name'],i['today']['confirm'],0])
    else:
        clist.append([i['name'], i['today']['confirm'], i['today']['wzz_add']])
```

今日热搜

聊天机器人可以识别与“今日热搜”相关的问题，自动爬取当前的微博热搜，具体功能如下：

- 返回当前热搜中热度最高的新闻资讯
- 点击热搜内容的文本即可跳转至热搜的链接了解详情

```
response = requests.get("https://weibo.com/ajax/side/hotSearch")
data_json = response.json()['data']['realtime']
for data_item in data_json:
    dic = {#微博词条的网址为'https://s.weibo.com/weibo?q=%23' + 词条名称 + '%23'
          'title': data_item['note'],
          'url': 'https://s.weibo.com/weibo?q=%23' + data_item['word'] + '%23',
          'num': data_item['num']}
    }
```

天气查询

聊天机器人可以识别与“今日天气”相关的问题，并且根据输入的城市爬取该城市的当日天气的详细情况。

```
#将输入的汉字用Pinyin转化为拼音字符串
city_pinyin=p.get_pinyin(sentence,'')
#通过'https://www.tianqi.com/'+城市的拼音名即可访问目的城市天气
try:
    url = 'https://www.tianqi.com/'+city_pinyin
    r = urllib.request.Request(url=url, headers=headers)
except urllib.error.URLError as e:#查找失败
    return "抱歉哈，找不到你所输入的城市"
#查找成功，利用beautifulsoup依次查找要找出的信息
weather.append(soup.select(...)[0].text)
```

机器人自我认知的匹配回复

机器人的自我认知主要包括姓名、年龄、家乡、性别。

使用一个answers.json文件，里面记录机器人的回答。文件一部分如下：

```
{
  "intents": [
    {
      "tag": "gender",
      "questions": [
        "你是男的吗",
        "你是女的吗",
        "你是男的还是女的",
        "你是男的女的",
        "你是什么性别",
        "你的性别是什么"
      ],
      "responses": [
        "机器人是没有性别的",
        "我是机器人，我要性别做什么",
        "我和原生生物一样，没有性别",
        "你猜猜看",
        "我不需要有性别",
        "你们人类啊，老纠结性别，烦不烦",
        "我没有性别，而且我觉得这是件好事",
        "我是没有性别的，而且为此感到庆幸，多少烦恼由性别产生。",
        "我没有也不需要性别，不过你可以假装我是任意性别",
        "性别这种东西是机器人不需要操心的",
        "机器人没有性别，而且为没有性别而快乐",
        "我性别是无，无需脱单，无需恋爱，多好",
        "你问这个干什么"
      ]
    }
  ],
  .....
}
```

匹配用的是fuzzywuzzy库提供的模糊匹配，通过与同样记录在answers.json文件中的提问语句进行模糊匹配，来判断是否是在询问机器人的自我认知问题，如果匹配上了就从这一类回答中random出一个回答。同时还用来jieba库的分词功能，进行进一步的匹配，防止模糊匹配遗漏的情况。由于机器人的回答每一个都有十条左右，所以如果不是故意为难，而是正常聊天的话不会出现很多重复，还是可以比较正常的回复的。

匹配的部分python代码如下:

```
def is_gender(seg_list, sentence, intents):
    //先检查是否与预设的问法匹配
    list_of_intents = intents['intents'] //这里在.json文件里有多种问法
    questions = list_of_intents[0]['questions']
    for q in questions:
        if fuzz.partial_ratio(q, sentence) >= 80:
            return True
    //再通过jieba分词检查是否有相应的关键词, 从而进一步检测
    flag_you = 0
    flag_gender = 0
    for v in seg_list:
        if v == "男" or v == "女" or v == "性别":
            flag_gender = 1
        elif v == "你" or v == "您":
            flag_you = 1
    if flag_gender == 1 and flag_you == 1:
        return True
    else:
        return False
    //其他的匹配函数类似
```

返回回答的函数:

```
def get_response(tag, intents_json):
    //tag标志是那种问题
    list_of_intents = intents_json['intents']
    //从对应的tag回答中random出一个
    for i in list_of_intents:
        if i['tag'] == tag:
            result = random.choice(i['responses'])
            break
    return result
```

小说和电影的推荐

按照fuzzywuzzy结合jieba分词的方式辨别出是在询问小说后, 会在一个库里面random出一部小说, 然后返回给用户。

识别部分代码:

```
def Is_novel(sentence, intents):
    //先检查是否与预设的问法匹配
    list_of_intents = intents['intents']
    questions = list_of_intents[5]['questions'] //这里在.json文件里有多种问法
    for q in questions:
        if fuzz.partial_ratio(q, sentence) >= 80:
```



```

        return True
    //再通过jieba分词检查是否有相应的关键词，从而进一步检测
    flag_question = 0
    flag_novel = 0
    seg_list = jieba.cut(sentence, cut_all=True)
    for v in seg_list:
        if v == "小说" or v == "书":
            flag_music = 1
        elif v == "吗" or v == "什么" or v == "嘛":
            flag_question = 1
    if flag_novel == 1 and flag_question == 1:
        return True
    else:
        return False

```

返回小说的函数：

```

def get_novel():
    novel_list = novels["novels"]
    result = random.choice(novel_list)
    return result

```

同样方式辨别出生在询问电影之后，会反问"有这些类型的电影：剧情 喜剧 动作 爱情 科幻 动画 悬疑 惊悚\n请从中选择一个"，用户输入其中之一之后，会从库里对应匹配上的类型里面random出一部电影进行返回。

识别部分代码：

```

def Is_movie(sentence, intents):
    //先检查是否与预设的问法匹配
    list_of_intents = intents['intents']
    questions = list_of_intents[4]['questions']//这里在.json文件里有多种问法
    for q in questions:
        if fuzz.partial_ratio(q, sentence) >= 80:
            return True
    //再通过jieba分词检查是否有相应的关键词，从而进一步检测
    flag_question = 0
    flag_movie = 0
    seg_list = jieba.cut(sentence, cut_all=True)
    for v in seg_list:
        if v == "电影" or v == "影片":
            flag_movie = 1
        elif v == "吗" or v == "什么" or v == "嘛" or v == "推荐":
            flag_question = 1
    if flag_movie == 1 and flag_question == 1:
        return True
    else:
        return False

```

反问并获得类型方面的处理:

```
global is_movie
//如果is_movie是1, 则需要根据类型返回一部电影
if is_movie == 1:
    print("is movie")
    test=pick_movie(data)
    is_movie = 0

//检测出问的是电影之后反问, 并把is_movie置1
elif res == "movie":
    test="有这些类型的电影: 剧情 喜剧 动作 爱情 科幻 动画 悬疑 惊悚\n请从中选择一个"
    is_movie = 1
```

电影返回的代码:

```
def pick_movie(type):
    list_of_movies = movies["movies"]
    //从库里对应的类型random出一个电影
    for data in list_of_movies:
        if data["tag"] == type:
            result = random.choice(data['list'])
            return result
    result = "抱歉哈, 没有找到这个类型诶"
    return result
```

计算复杂数学表达式的实现

基于正则匹配的数学表达式的运算实现。

```
import re

def format_mark(express):
    # distinguish and replace the unexpected character that may disturb the calculation
    express = express.replace('+-', '-')
    express = express.replace('-+', '-')
    express = express.replace('++', '+')
    express = express.replace('--', '+')
    express = express.replace('*+', '*')
    express = express.replace('+*', '*')
    express = express.replace('/+', '/')
    express = express.replace('/+', '/')
    return express

def com_cal_plus_minus(express):
    expr = express
```

```

sub_expr = re.search(r"\-?\d+\.\.?d*[\+|-]\d+\.\.?d*", expr)
if not sub_expr:
    return expr
else:
    sub_expr2 = sub_expr.group()
    if len(sub_expr2.split('+')) > 1:
        n1, n2 = sub_expr2.split('+')
        result = float(n1)+float(n2)
    else:
        n1, n2 = sub_expr2.split('-')
        result = float(n1) - float(n2)
    re_sub_expr = re.sub(r"\-?\d+\.\.?d*[\+|-]\d+\.\.?d*", str(result), expr,
count=1)
    # calculate dicide over and over and over again
    bb = com_cal_plus_minus(str(re_sub_expr))
    return bb

def com_cal_multilply_divide(expr_div):
    expr=expr_div
    sub_expr = re.search(r"\d+\.\.?d*[/\*]\-?\d+\.\.?d*",expr)
    if not sub_expr:
        return expr
    else:
        sub_expr2 = sub_expr.group()
        if len(sub_expr2.split('/')) > 1:
            n1, n2 = sub_expr2.split('/')
            result = float(n1)/float(n2)
        if len(sub_expr2.split('*')) > 1:
            n1, n2 = sub_expr2.split('*')
            result = float(n1)*float(n2)
        else:
            pass
        re_sub_expr=re.sub(r"\d+\.\.?d*[/\*]\-?\d+\.\.?
\d*",str(result),expr,count=1)
        # calculate dicide over and over and over again
        bb=com_cal_multilply_divide(format_mark(re_sub_expr))
        return bb

def compute(express):
    express = com_cal_multilply_divide(format_mark(express))
    express = com_cal_plus_minus(format_mark(express))
    return express

def clear(express):
    # detect the blank space
    res=re.compile(r'[\s]')
    sub_expr1 = re.search('(\([+\-\\*\\/\\.0-9]+\))', express)
    if not sub_expr1:
        return express
    else:
        sub_expr1=sub_expr1.group()
        sub_expr2=sub_expr1[1:len(sub_expr1)-1]
        sub_expr3=compute(sub_expr2)
        sub_expr3 = re.sub('(\([+\-\\*\\/\\.0-9]+\))',

```

```

str(sub_expr3), express, count=1)
    clear_expr=clear(format_mark(sub_expr3))
    return clear_expr

def cal(express):
    while True:
        res = ""
        express = re.sub('\s*', '', express)
        if len(express) == 0:
            continue
        elif express == 'q':
            res = "计算结束啦! "
            return res
        elif re.search('[^0-9\.\-+\*\\/\(\)]', express):
            res = "不是有效的算数表达式哦, 已退出计算!"
            return res
        else:
            express = express.replace(' ', '')
            express = clear(express)
            # clear the blank between discriptions
            express = compute(format_mark(express))
            # calculate again
            # in case the unexpected character to disturb the calculation
            return str(express)

if __name__ == '__main__':
    ret = cal('1 - 2 * ((60-30 +(-40.0/5) * (9-2*5/3 + 7 /3*99/4*2998 +10 * 568/14
)) - (-4*3)/ (16-3*2))')
    print(ret)

```

可以用此程序进行复杂的数学表达式的计算。

```

93 if __name__ == '__main__':
94     ret = cal('1 - 2 * ((60-30 +(-40.0/5) * (9-2*5/3 + 7 /3*99/4*2998 +10 * 568/14 )) - (-4*3)/ (16-3*2))')
95     print(ret)

```

问题 输出 调试控制台 终端

2776672.6952380957
(base) PS C:\Users\Lenovo> □

设计识别特定功能进行特殊返回

通过模糊匹配识别出实现的那些特定功能, 实现在人工智能生成式聊天机器人模型之上的一层回复, 从而让聊天机器人的回答更加一致和多样化。

```

#输入的话
data=""
@app.route('/post_text', methods=['POST'])
def post_text():
    global data
    data=request.form.get('mytext')

```

```

//用于标记上一次询问类型的全局变量
is_movie = 0 //电影
is_weather = 0 //天气
is_YQ = 0 //疫情
is_HE = 0 //热点
is_JS = 0 //计算器

#机器人的话
@app.route('/get_text',methods=['GET'])
def get_text():
    global is_movie
    global is_weather
    global is_YQ
    global is_JS

    res = get_answer(data)

    if is_movie == 1: //上一次是电影
        print("is movie")
        test=pick_movie(data)
        is_movie = 0
    elif is_weather == 1: //上一次是天气
        test=get_weather(data)
        is_weather = 0
    elif is_YQ == 1: //上一次是疫情
        print("is_YQ=1")
        test=get_covid(data)
        print(test)
        print(chengshi)
        print(shuju)
        is_YQ = 0
    elif is_JS == 1: //上一次是计算器
        test=cal(data)
        is_JS = 0
    elif res == "AI": //如果检查返回"AI", 说明需要AI生成
        test=generate(data)
    elif res == "movie": //如果返回电影, 则反问
        test="有这些类型的电影: 剧情 喜剧 动作 爱情 科幻 动画 悬疑 惊悚\n请从中选择一个"
        is_movie = 1
    elif res == "novel": //如果返回小说, 则random出一部小说
        test=get_novel()
    elif res == "weather": //如果是问天气, 反问是那个城市的天气
        test="输入城市名, 我可以告诉你那里今天的天气"
        is_weather = 1
    elif res == "HE": //如果是热点, 则返回最热的一条热点
        print("hotsearch")
        test=get_hotsearch()
        is_HE = 1
        print(test)
    elif res == "YQ": //如果是疫情, 反问需要查询的省份
        test="输入想查询的省份"
        is_YQ = 1

```

```

elif res == "JS": //如果是计算器, 反问表达式
    print("JSQ")
    test="请输入表达式"
    is_JS = 1
else:
    test=res
return test

```

检测询问的类型的函数:

```

def get_answer(message):
    print(message)
    my_type = get_type(message, intents)
    //一个条件判断, ——检查是否是某类的问题, 并返回相应的字符串
    if my_type != "none":
        res = get_response(my_type, intents)
    elif Is_movie(message, intents):
        res = "movie"
    elif Is_novel(message, intents):
        res = "novel"
    elif Is_weather(message, intents):
        res = "weather"
    elif Is_HE(message, intents):
        res = "HE"
    elif Is_YQ(message, intents):
        res = "YQ"
    elif message == "计算器":
        res = "JS"
    else:
        res = "AI"
    print(res)
    return res

```

检测疫情、天气、热点的函数:

```

def Is_weather(sentence, intents):
    //先检查是否与预设的问法匹配
    today = 0
    list_of_intents = intents['intents']
    questions = list_of_intents[6]['questions']//这里在.json文件里有多种问法
    for q in questions:
        if fuzz.partial_ratio(q, sentence) >= 80:
            return True
    //再通过jieba分词检查是否有相应的关键词, 从而进一步检测
    flag_question = 0
    flag_weather = 0
    seg_list = jieba.cut(sentence, cut_all=True)
    for v in seg_list:

```

```
    if v == "天气":
        flag_weather = 1
    elif v == "吗" or v == "什么" or v == "嘛" or v == "怎么样":
        flag_question = 1
    if flag_weather == 1 and flag_question == 1:
        return True
    else:
        return False

def Is_HE(sentence, intents):
    today = 0
    list_of_intents = intents['intents']
    questions = list_of_intents[7]['questions']
    for q in questions:
        if fuzz.partial_ratio(q, sentence) >= 80:
            return True
    return False

def Is_YQ(sentence, intents):
    today = 0
    list_of_intents = intents['intents']
    questions = list_of_intents[8]['questions']
    for q in questions:
        if fuzz.partial_ratio(q, sentence) >= 80:
            return True
    return False
```

可以注意到我们用了相当多的全局变量作为标志使得匹配的信息得以在不同的函数体之间传递。

这样就在前端和AI之间实现了一层以匹配和事先准备好的回答的回复，这样可以让机器人能够言之有物的回答一部分问题。

NLP文本生成

前期调研

首先对于这一方向的具体实现我进行了较为深入的调研，调研报告都附在文件夹内了，也可以点击下方快速跳转。

1. 调研报告: [Research_Record](#)
2. 可参考项目的调研: [Record](#)

以上虽然罗列了许多可以参考的项目，但是实际还是自己基于Github上最原始的GPT-2模型进行的搭建工作。

搭建的过程中主要是参考了以下网站的讲解与思路：

1. [The Illustrated GPT-2 \(Visualizing Transformer Language Models\)](#)
2. [Better Language Models and Their Implications](#)
3. [完全图解GPT-2](#)

主要看的是前两篇，第三个网页是对于前面网站的翻译和解读，作者加入了很多自己的理解所以我不太喜欢。

这几个网页对我的启发意义很大，我模型参数的选取，后续参数化扫描的应用都是基于这几个网站给出的指导信息。

训练文件构成

以下是模型训练部分的基本构成：

```
:.
  data_parallel.py
  pre_pro.py
  response.py
  train_model.py

.vscode
  settings.json

config
  config.json

data
  train.log
  train.txt

list
  list.txt

model

sample
  samples.txt

__pycache__
  dataset.cpython-39.pyc
  pytorchtools.cpython-39.pyc
```

py文件

data_parallel.py：用于训练时做数据并行，提高GPU使用率与训练效率。

```
from torch.nn.parallel import DataParallel
import torch
from torch.nn.parallel._functions import Scatter
from torch.nn.parallel.parallel_apply import parallel_apply

def scatter(inputs, target_gpus, chunk_sizes, dim=0):

    def scatter_map(obj):
        if isinstance(obj, torch.Tensor):
            try:
                return Scatter.apply(target_gpus, chunk_sizes, dim, obj)
            except:
                quit()
        if isinstance(obj, tuple) and len(obj) > 0:
```



```

        return list(zip(*map(scatter_map, obj)))
    if isinstance(obj, list) and len(obj) > 0:
        return list(map(list, zip(*map(scatter_map, obj))))
    if isinstance(obj, dict) and len(obj) > 0:
        return list(map(type(obj), zip(*map(scatter_map, obj.items()))))
    return [obj for targets in target_gpus]

# After scatter_map is called, a scatter_map cell will exist. This cell
# has a reference to the actual function scatter_map, which has references
# to a closure that has a reference to the scatter_map cell (because the
# fn is recursive). To avoid this reference cycle, we set the function to
# None, clearing the cell
try:
    return scatter_map(inputs)
finally:
    scatter_map = None

def scatter_kwargs(inputs, kwargs, target_gpus, chunk_sizes, dim=0):
    r"""Scatter with support for kwargs dictionary"""
    inputs = scatter(inputs, target_gpus, chunk_sizes, dim) if inputs else []
    kwargs = scatter(kwargs, target_gpus, chunk_sizes, dim) if kwargs else []
    if len(inputs) < len(kwargs):
        inputs.extend([() for _ in range(len(kwargs) - len(inputs))])
    elif len(kwargs) < len(inputs):
        kwargs.extend([{} for _ in range(len(inputs) - len(kwargs))])
    inputs = tuple(inputs)
    kwargs = tuple(kwargs)
    return inputs, kwargs

class BalancedDataParallel(DataParallel):
    def __init__(self, gpu0_bsz, *args, **kwargs):
        self.gpu0_bsz = gpu0_bsz
        super().__init__(*args, **kwargs)

    def forward(self, *inputs, **kwargs):
        if not self.device_ids:
            return self.module(*inputs, **kwargs)
        if self.gpu0_bsz == 0:
            device_ids = self.device_ids[1:]
        else:
            device_ids = self.device_ids
        inputs, kwargs = self.scatter(inputs, kwargs, device_ids)
        # print('len(inputs): ', str(len(inputs)))
        # print('self.device_ids[:len(inputs)]',
str(self.device_ids[:len(inputs)]))
        if len(self.device_ids) == 1:
            return self.module(*inputs[0], **kwargs[0])
        replicas = self.replicate(self.module, self.device_ids[:len(inputs)])
        if self.gpu0_bsz == 0:
            replicas = replicas[1:]
        outputs = self.parallel_apply(replicas, device_ids, inputs, kwargs)
        return self.gather(outputs, self.output_device)

```

```

def parallel_apply(self, replicas, device_ids, inputs, kwargs):
    return parallel_apply(replicas, inputs, kwargs, device_ids[:len(inputs)])

def scatter(self, inputs, kwargs, device_ids):
    bsz = inputs[0].size(self.dim)
    num_dev = len(self.device_ids)
    gpu0_bsz = self.gpu0_bsz
    bsz_unit = (bsz - gpu0_bsz) // (num_dev - 1)
    if gpu0_bsz < bsz_unit:
        chunk_sizes = [gpu0_bsz] + [bsz_unit] * (num_dev - 1)
        delta = bsz - sum(chunk_sizes)
        for i in range(delta):
            chunk_sizes[i + 1] += 1
        if gpu0_bsz == 0:
            chunk_sizes = chunk_sizes[1:]
    else:
        return super().scatter(inputs, kwargs, device_ids)

    return scatter_kwargs(inputs, kwargs, device_ids, chunk_sizes,
dim=self.dim)

```

- 实际训练中我用了一台8张K80卡的服务器训练了40epochs，因为模型体量大、训练轮次多且涉及到多张卡的并行，所以选择进行数据并行提高效率
- 原理不过多赘述了，基本上就是将数据在GPU_0上根据可用device数进行分配，运算后所有数据汇总到GPU_0上再进行计算，调参，再分配...如此循环
- 最后的效果基本上为：CPU_0的内存使用量为余下设备的两倍，是准确的：2:1:1:1:...:1的关系，但是实际的内存占用量还是要看卡数进行分配。

****pre_pro.py: **用于对数据集data.txt进行预加工处理**

```

# each block's structure after preprocess:
# [CLS] sentence1 [SEP] sentence2 [SEP] sentence3 [SEP]
...
def preprocess():
    ...
    # initial tokenizer
    tokenizer = BertTokenizerFast(vocab_file=args.list_path, sep_token="[SEP]",
pad_token="[PAD]", cls_token="[CLS]")
    sep_id = tokenizer.sep_token_id
    cls_id = tokenizer.cls_token_id

    # read train_data
    with open(args.train_path, 'rb') as f:
        data = f.read().decode("utf-8")

    # tokenize
    sentence_list = []

```

```

# Linux or Windows
if "\r\n" in data:
    train_data = data.split("\r\n\r\n")
else:
    train_data = data.split("\n\n")

with open(args.save_path, "w", encoding="utf-8") as f:
    for index, block in enumerate(tqdm(train_data)):
        # Linux or Windows
        if "\r\n" in data:
            sentences = block.split("\r\n")
        else:
            sentences = block.split("\n")

        # add [CLS]
        input_ids = [cls_id]
        for sentence in sentences:
            input_ids += tokenizer.encode(sentence, add_special_tokens=False)

            # add [SEP]
            input_ids.append(sep_id)

        # save the sentence
        sentence_list.append(input_ids)

# end
print('data preprocess done successfully!')

...

```

response.py: 基于已训练好的模型做对话生成测试（这一部分也最终移植到了我们的final_project内）

****train_model.py**: **根据预加工好的数据集进行模型的初始化与训练

后两个文件代码过长，就不粘贴在此——介绍了

其他文件

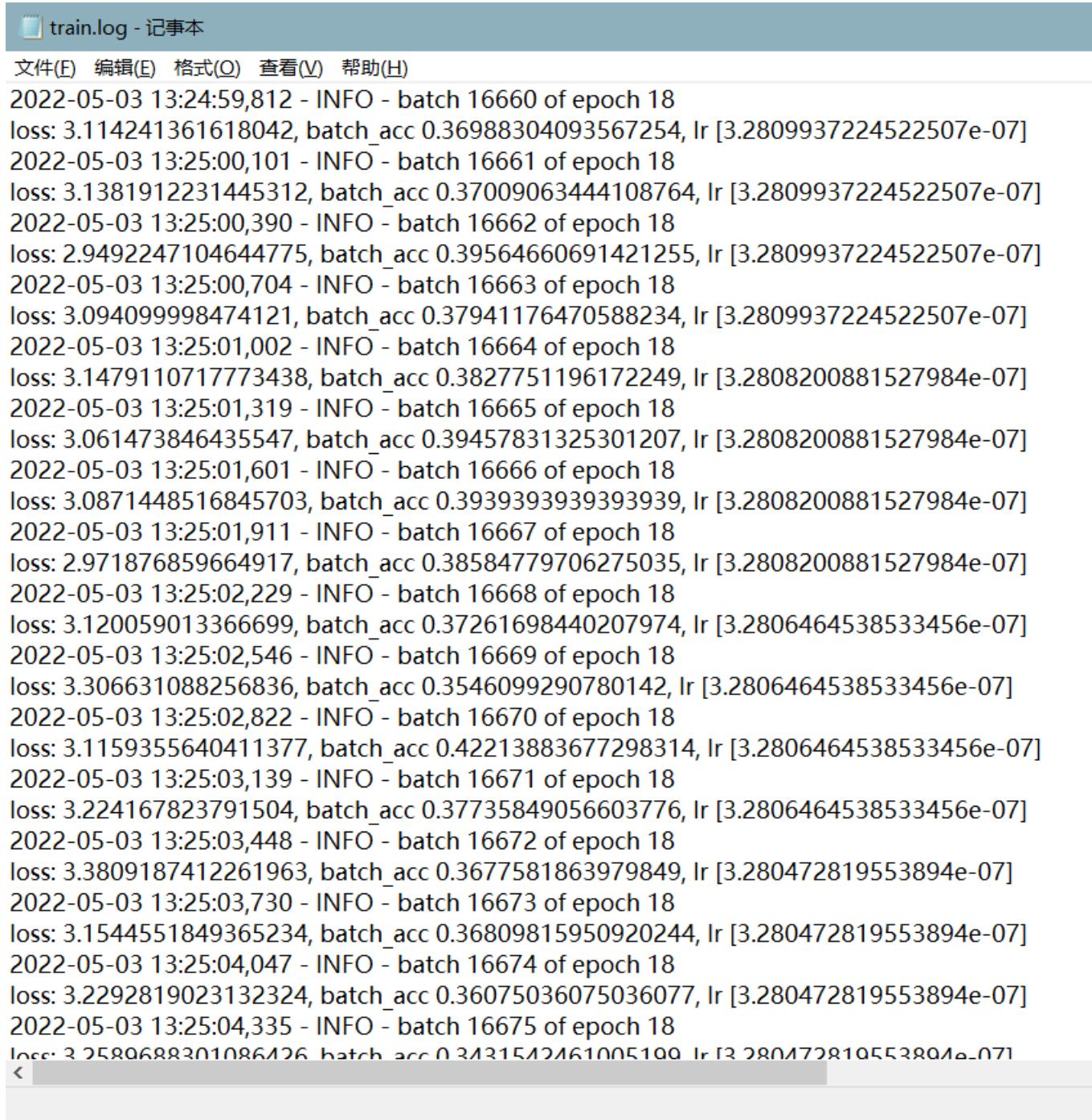
config.json: 存储模型的参数信息，如head数、hidden_layer层数等。

```
"attn_pdrop": 0.1,  
"bos_token_id": 50256,  
"embd_pdrop": 0.1,  
"eos_token_id": 50256,  
"gradient_checkpointing": false,  
"initializer_range": 0.02,  
"layer_norm_epsilon": 1e-05,  
"model_type": "gpt2",  
"n_ctx": 1024,  
"n_embd": 768,  
"n_head": 12,  
"n_inner": null,  
"n_layer": 12,  
"n_positions": 1024,  
"output_past": true,  
"resid_pdrop": 0.1,  
"summary_activation": null,  
"summary_first_dropout": 0.1,  
"summary_proj_to_labels": true,  
"summary_type": "cls_index",  
"summary_use_proj": true,
```

train.txt: 训练用语料集, 50W语料, 来源注明在我的Research.pdf内了。

```
≡ train.txt  
1  谢谢你所做的一切  
2  你开心就好  
3  开心  
4  嗯因为你的心里只有学习  
5  某某某, 还有你  
6  这个某某某用的好  
7  
8  你们宿舍都是这么厉害的人吗  
9  眼睛特别搞笑这土也不好捏但就是觉得挺可爱  
10 特别可爱啊  
11  
12 今天好点了吗?  
13 一天比一天严重  
14 吃药不管用, 去打一针。别拖着  
15  
16 是的。下辈子想做只萤火虫  
17 可是萤火虫太容易被抓了还是改一个吧  
18 不, 我只想奋不顾身扑火  
19  
20 加油, 三月动起来, 五月笑起来  
21 正解! 你为什么就那么厉害呢  
22 哈哈, 没办法, 智商就是这么高  
23 你这是要开始得瑟了吗! 好啦! 你最厉害!  
24 哈哈哈哈哈
```

train.log: 训练的记录信息。



```
train.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
2022-05-03 13:24:59,812 - INFO - batch 16660 of epoch 18
loss: 3.114241361618042, batch_acc 0.36988304093567254, lr [3.2809937224522507e-07]
2022-05-03 13:25:00,101 - INFO - batch 16661 of epoch 18
loss: 3.1381912231445312, batch_acc 0.37009063444108764, lr [3.2809937224522507e-07]
2022-05-03 13:25:00,390 - INFO - batch 16662 of epoch 18
loss: 2.9492247104644775, batch_acc 0.39564660691421255, lr [3.2809937224522507e-07]
2022-05-03 13:25:00,704 - INFO - batch 16663 of epoch 18
loss: 3.094099998474121, batch_acc 0.37941176470588234, lr [3.2809937224522507e-07]
2022-05-03 13:25:01,002 - INFO - batch 16664 of epoch 18
loss: 3.1479110717773438, batch_acc 0.3827751196172249, lr [3.2808200881527984e-07]
2022-05-03 13:25:01,319 - INFO - batch 16665 of epoch 18
loss: 3.061473846435547, batch_acc 0.39457831325301207, lr [3.2808200881527984e-07]
2022-05-03 13:25:01,601 - INFO - batch 16666 of epoch 18
loss: 3.0871448516845703, batch_acc 0.3939393939393939, lr [3.2808200881527984e-07]
2022-05-03 13:25:01,911 - INFO - batch 16667 of epoch 18
loss: 2.971876859664917, batch_acc 0.38584779706275035, lr [3.2808200881527984e-07]
2022-05-03 13:25:02,229 - INFO - batch 16668 of epoch 18
loss: 3.120059013366699, batch_acc 0.37261698440207974, lr [3.2806464538533456e-07]
2022-05-03 13:25:02,546 - INFO - batch 16669 of epoch 18
loss: 3.306631088256836, batch_acc 0.3546099290780142, lr [3.2806464538533456e-07]
2022-05-03 13:25:02,822 - INFO - batch 16670 of epoch 18
loss: 3.1159355640411377, batch_acc 0.42213883677298314, lr [3.2806464538533456e-07]
2022-05-03 13:25:03,139 - INFO - batch 16671 of epoch 18
loss: 3.224167823791504, batch_acc 0.37735849056603776, lr [3.2806464538533456e-07]
2022-05-03 13:25:03,448 - INFO - batch 16672 of epoch 18
loss: 3.3809187412261963, batch_acc 0.3677581863979849, lr [3.280472819553894e-07]
2022-05-03 13:25:03,730 - INFO - batch 16673 of epoch 18
loss: 3.1544551849365234, batch_acc 0.36809815950920244, lr [3.280472819553894e-07]
2022-05-03 13:25:04,047 - INFO - batch 16674 of epoch 18
loss: 3.2292819023132324, batch_acc 0.36075036075036077, lr [3.280472819553894e-07]
2022-05-03 13:25:04,335 - INFO - batch 16675 of epoch 18
loss: 3.2589688301086426, batch_acc 0.3431542461005100, lr [3.280472819553894e-07]
<
```

list.txt: 训练及生成所用词表。

| | |
|------|---|
| 1982 | 姁 |
| 1983 | 妖 |
| 1984 | 姬 |
| 1985 | 妮 |
| 1986 | 姐 |
| 1987 | 妳 |
| 1988 | 妹 |
| 1989 | 妻 |
| 1990 | 妾 |
| 1991 | 姆 |
| 1992 | 姊 |
| 1993 | 姊 |
| 1994 | 始 |
| 1995 | 姍 |
| 1996 | 姐 |
| 1997 | 姑 |
| 1998 | 姁 |
| 1999 | 姓 |
| 2000 | 委 |
| 2001 | 姍 |
| 2002 | 姚 |
| 2003 | 姜 |
| 2004 | 妹 |
| 2005 | 姁 |
| 2006 | 姥 |
| 2007 | 姁 |
| 2008 | 姨 |
| 2009 | 姁 |

总结说明

更具体的内容可以参见train_part内的代码内容，里面有详细的注释以及我自己写代码时的Learning Note，这一模型的搭建，我是将训练模型分成若干个小的任务分别完成的，比如：tokenizer的搭建，logger的添加，每一部分是现实查阅的资料都很多，重要的基本都标注在了我代码内的注释中，对于模型的核心实现部分也主要阅读了以下三篇论文，也附在我们的工程目录文件夹内了，点击下方即可跳转。 [References](#)