项目详细方案

运用文本相似度实现（证券）智能客服【恒生电子】

团队编号： 1801733

参赛选手： 邵茂仁、杨晓飞、林丽婷

使 用 人： 公开

密 级 ： 公开

日 期 ： 2019 年 3 月 20 日

目　录

1 需求分析

1.1 设计背景

作为企业客户关系管理（CRM）的重要组成部分，客服是连接企业与客户的重要桥梁，极大地影响着企业的销售成果、品牌影响及市场地位。而随着移动互联网时代的到来，智能客服系统应运而生，提供了及时、快速、准确的全渠道服务，大大地提高了客服工作效率，优化了客服工作流程。

针对互联网金融行业，我们希望开发一套智能客服系统，该系统能够根据客户提问的问题，自动匹配知识库中相似度最高的的标准问，从而帮助客户更加准确地定位问题。基于可扩展的文本相似度算法实现的证券类智能客服系统，利用自然语言处理（NLP）和机器学习的关键技术，将在金融这一特定领域发挥优势，克服金融产业的专业性和复杂性问题，为客户提供强针对性、高准确率、反馈快速的服务。

1.2 概要介绍

1.2.1 功能简介

根据需求分析，智能客服系统的设计可分为两个模块：前端交互和后台管理。两个模块之间互相联系，经过整理得到各模块功能要求如下：

模块	要求
前端交互	➢ 实现 QA 对话界面 ➢ 实现问题引导功能 ➢ 实现反馈功能 ➢ 与后台管理进行交互
后台管理	➢ 根据客户提问匹配输出标准问 ➢ 管理客户反馈信息 ➢ 实现扩展知识库标准问集的功能 ➢ 与前端进行交互

1.2.2 模块设计

- 用户界面基本功能：

1）登录、注册

2）热门金融资讯查看

3）QA 会话界面进行提问

4）用户反馈

5）人工客服转接

6）常见问答浏览

- 后台管理基本功能：

1）根据客户问匹配标准问

2）管理客户反馈信息

3）标准问数据集的扩展

1.2.3 软件设计总体流程

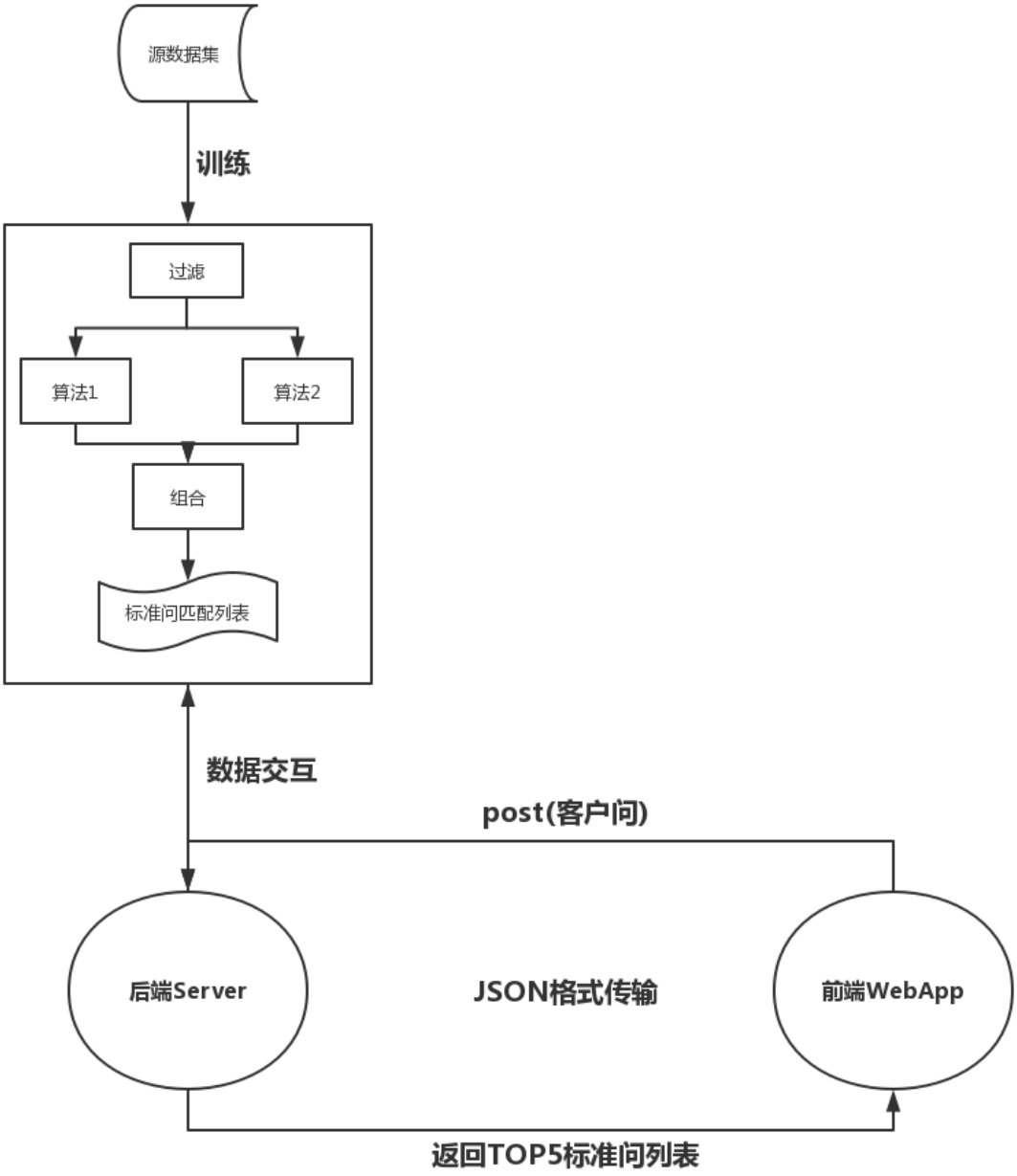

图 1-1 软件设计总体流程图

在软件系统的总体设计上，我们采用了前端后分离架构。主要分为前端、后端、算法三个模块。总体步骤如下：

Step1：客户通过前端 webAPP 输入客户问。

Step2：前端向后端服务器发起 POST 请求，将客户问发送给后端服务器。

Step3：后端服务器接收到请求后与算法模块进行数据交互。

Step4：后端服务器接收到算法模块的最终 TOP5 匹配标准问列表后以 POST 的形式回传给前端。

Step5：前端接收到结果后重新渲染给用户看。

1.3 预期目标

🕮 针对金融领域，服务更专业

本系统实现智能识别金融证券类的专有名词，凭借预置的领域知识，应用多种人工智能技术，深入理解客户提问，挖掘客户真正关心的问题，更快地为客户提供准确的问题定位。

🕮 多种算法结合，问题匹配更准确

本系统在基于文本相似度的技术下结合了多种核心算法，使得系统在问题分析、问题检索、问题匹配的部分都更加智能准确。在经过多次实验后，得出最佳的算法结合方法，大大地提高系统的精准性。

🕮 用户界面美观，功能丰富

实现一个界面美观，交互友好的智能客服系统。简洁的界面设计使得客户更加直观地了解整个前端结构，能够快速地找到各种功能的入口。功能丰富，除了基本的QA界面，还提供了热点资讯、常见问答等功能，为客户打造贴心的智能系统。

🕮 多种渠道支持，服务更便捷

本系统选用了 Web app 的开发模式，具有跨平台的优势，支持多种渠道，嵌入更加方便。与原生APP相比，Web app在开发、功能、应用安装使用等方面都更有优势，也更加适用于智能客服的需求。

2 算法构建

2.1 实现技术与过程

表 2-1 算法实现环境说明表

环境	说明
编程语言	Python3.6.2
操作系统	Windows10
主要框架	Web.py

表 2-2 算法核心模块说明表

模块名称	模块说明
filter_by_IR	基于信息检索角度过滤部分客户问
get_standard_question_by_cosSim	基于余弦相似度获取标准问
get_standard_question_by_NB	基于朴素贝叶斯获取标准问
get_standard_question_bt_compose	基于组合模型获取最终的标准问

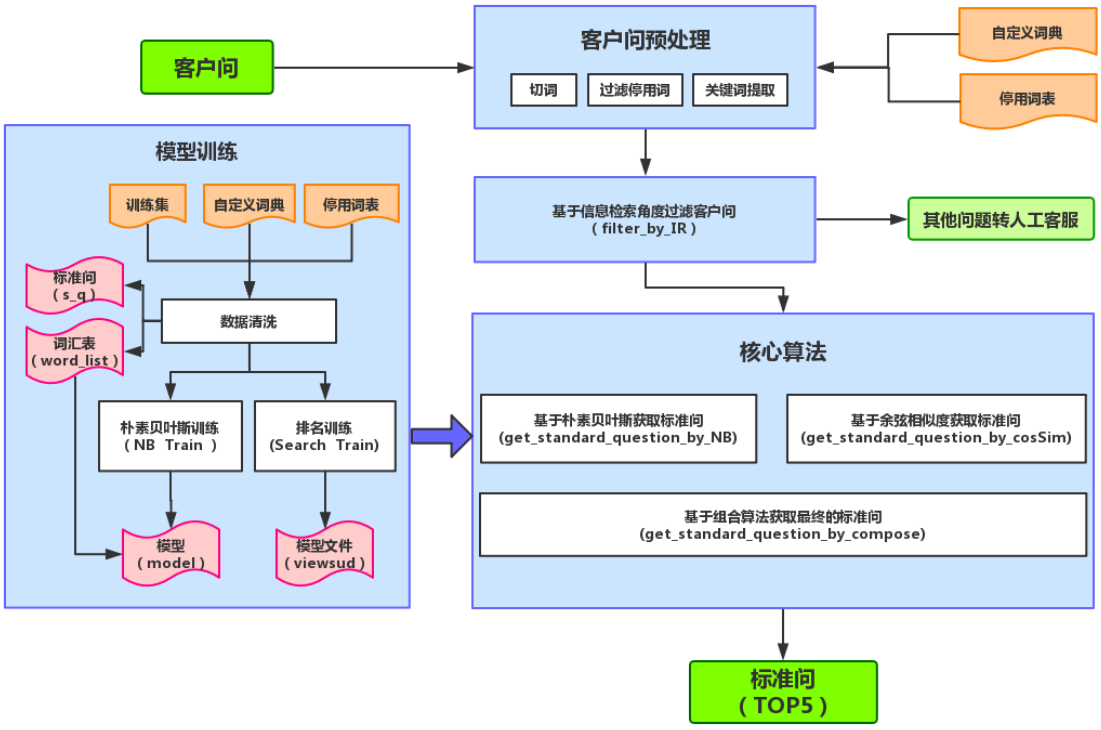

图 2-1 算法框架图

如图 2-1 所示，本团队针对该赛题所设计实现的核心算法主要分为以下四个部分：

1）客户问预处理。主要是对接收到的客户问进行分词、过滤停用词、提取客户问中的语义词。

2）客户问过滤。主要是过滤掉部分 “闲聊”式的提问，进一步提升匹配标准问的准确性。

3）模型训练。主要是展现了核心算法的大致的训练过程。

4）核心算法。主要是通过核心算法对过滤后的客户问匹配相应的标准问。

2.2 算法分析与流程

2.2.1 数据集分析

在获取到命题企业下发的竞赛数据集后，我们团队首先从“数据集结构特征”、“数据内容”、“数据分布”等角度对其进行了数据可视化分析。

	A	B
1	**客户提问**	**知识库标准问**
2	自助开户的第三方存管要怎么补办？	自助开户后如何补办三方存管？
3	自助开户三方存管	自助开户后如何补办三方存管？
4	手机App里开户后，如何绑定第三方存管	自助开户后如何补办三方存管？
5	自助开户后如何补办三方存管？	自助开户后如何补办三方存管？
6	我的帐户怎么不能进行和银行间的转账了？	转账失败的原因？
7	转账失败的原因	转账失败的原因？
8	转账失败的原因？	转账失败的原因？
9	什么原因银行转账到帐户，转入失败	转账失败的原因？
10	我刚才为何银行资金转账到证券失败？	转账失败的原因？
11	登录不成功，是什么原因	转账失败的原因？
12	不允许取款的原因？	转账失败的原因？
13	转账失败为什么	转账失败的原因？
14	我能在金融终端查到我的银行余额信息的，但转账为何不成功	转账失败的原因？
15	貌似我现在无法银证转账呢？	转账失败的原因？
16	转帐失败的原因	转账失败的原因？
17	你好 我银证转账从银行转证券怎么转不成功	转账失败的原因？
18	我要从银行转账到证券帐户上为什么转不了？	转账失败的原因？
19	你好，请问为何我的邮政储蓄银行转账不到帐户？	转账失败的原因？
20	证券转银行卡为什么转不了	证券转银行为什么转不了？
21	为什么银行转证券不行	证券转银行为什么转不了？

图 2-2 数据集结构图

如图 2-2 所示，通过分析数据集的结构特征，我们发现该数据集主要以“客户问-标准问”的形式存在，并且夹杂着中文、大小写英文、标点、特殊符号、重复等。在此分析基础上，我们首先进行了数据清洗：过滤掉部分标点符号、常用停用词，保留自定义词典中的词语，去重等操作。从而去除掉数据集中的部分噪声，为后续算法构建及模型训练提供帮助。

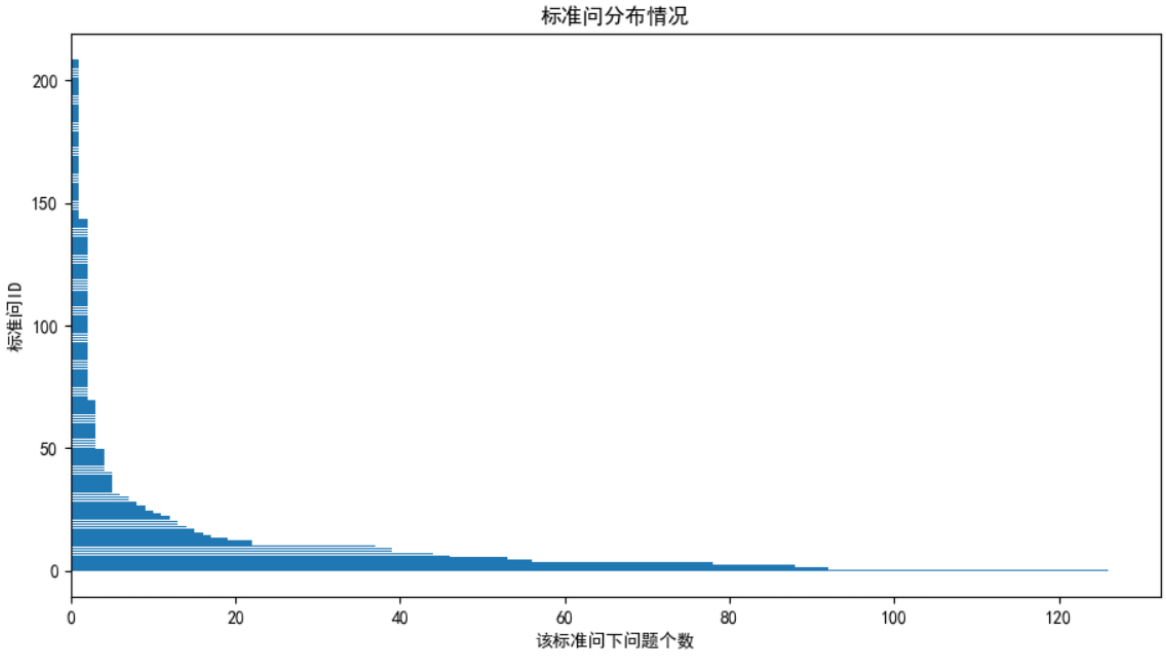

图 2-3 标准问分布图

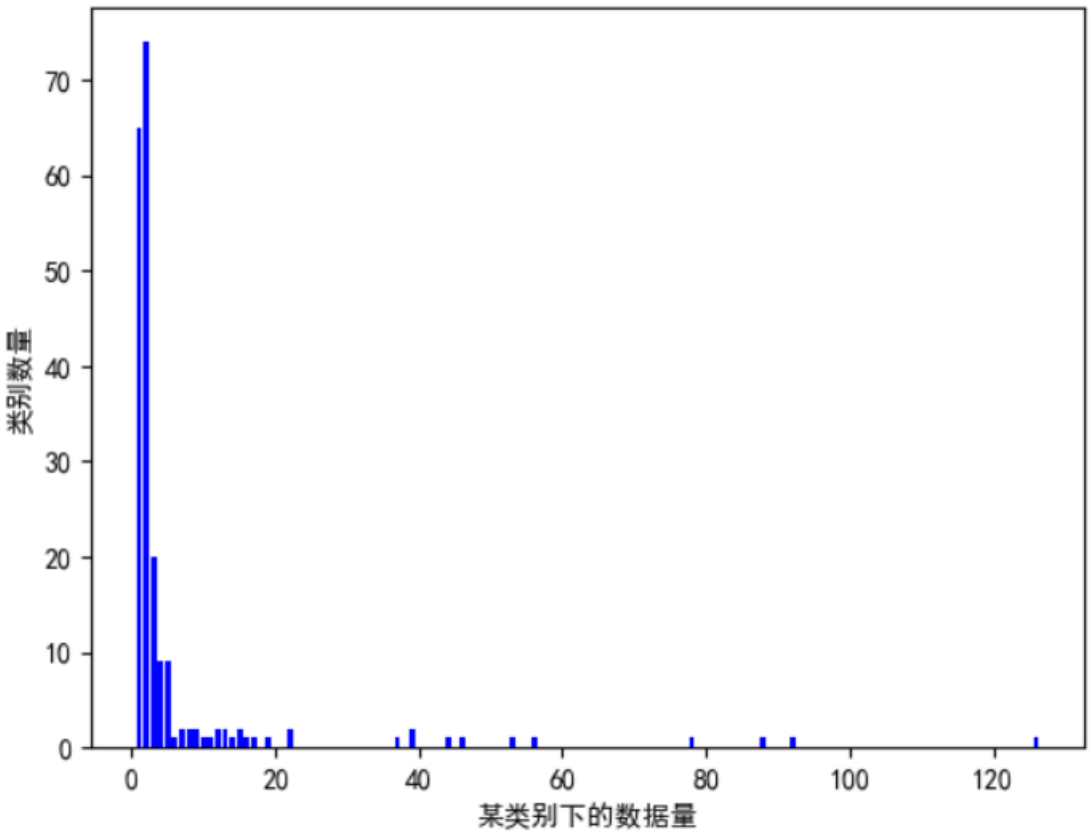

图 2-4 数据量分布图

如图 2-3 和图 2-4 所示，在分析数据集中标准问的分布情况和改标准问对应的客户问数量后，我们发现该数据集存在严重的类不平衡问题。其中，大部分标准问对应的客户问数量小于等 5，极少个标准问对应的客户问数量则达到了上百条。对此，我们从“数据增强”和“采样方式”两个角度进行了处理。

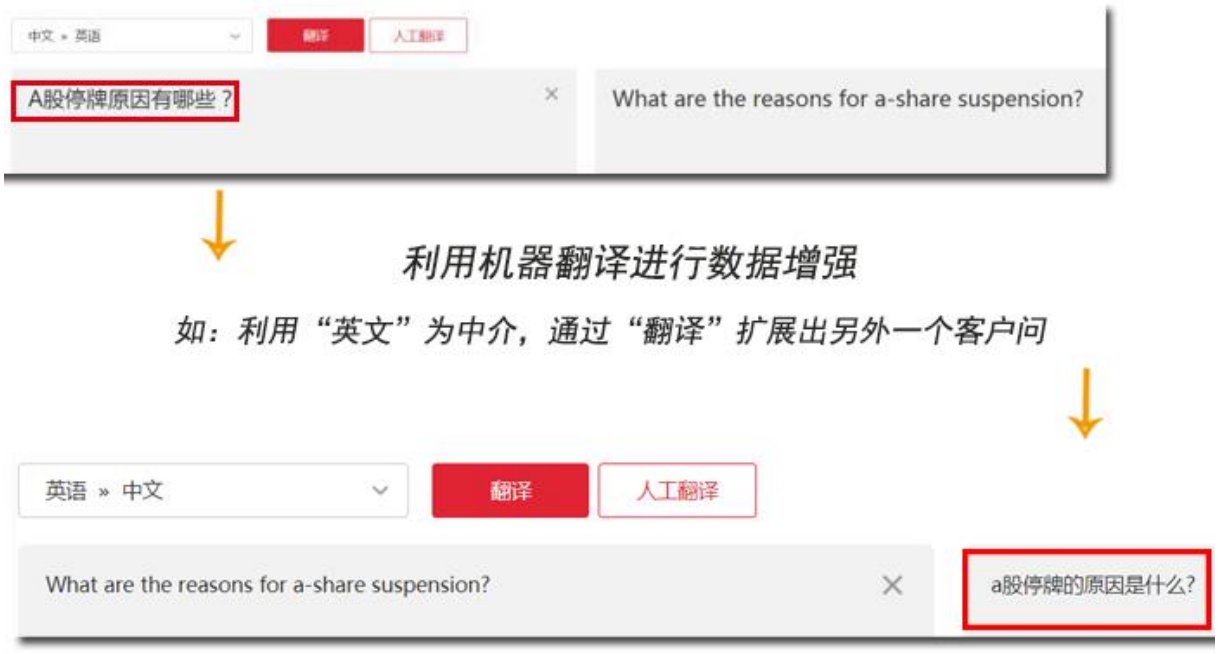

图 2-5 使用“机器翻译”进行数据增强原理图

首先，我们使用“机器翻译”的方式对数据进行了增强。从图 2-5 中可看出，标准问“A 股停牌原因有哪些？”经过中文机器翻译成英文，然后再从英文机器翻译回中的方式，原标准问则变成了“a 股停牌的原因是什么？”这个描述从语义上可近似相等，因而可扩展成为该标准问的一个客户问。此外，除了英文，我们还加入日文、韩文等语言充当“中介”语言，进而通过机器翻译这种方式所带来的语言之间的差异性增强了原有数据。

其次，通过分析数据量分布情况，在进行了数据增强的基础上，我们对于客户问数量较多的标准问采用了“欠采样”的方式进行采样处理；对于客户问数量较少的标准问采用了“过采样”的方式进行采样处理。

最后，通过“数据增强”与“过/欠采样”两种方式对原数据集中类不平衡问题进行了缓解。

2.2.2 问题分析与算法构建

经过讨论分析，该赛题主要目的是要客户问尽可能准确的匹配到知识库中的标准问。该问题属于 NLP 领域的问题，传统的做法是利用句子间的相似度计算从而进行匹配。主要的构建整体思路如下图所示：

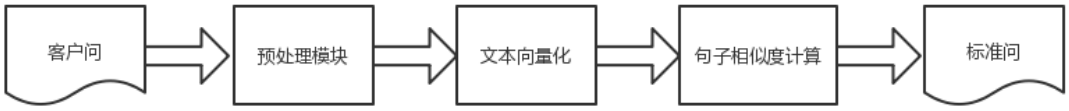

图 2-6 传统算法构建整体思路图

根据传统的解决方案思路,我们设计了基于余弦相似度标准问匹配算法。在文本向量化这一模块中我们采用了 TF-IDF 算法进行客户问与标准问的文本向量化。其中，余弦相似度以及 TF-IDF 的计算公式如下：

$$similarity = \cos(\theta) = \frac{A \bullet B}{\| A \| \| B \|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (1)$$

$$\begin{aligned} TF - IDF(\mathrm{w}_i) &= tf(w_i) \times idf(w_i) \\ &= tf_j(w_i) \times \log(N / df(w_i)) \end{aligned} \quad (2)$$

给定两个属性向量 A 和 B，其余弦相似性 θ 则可用式（1）表示，其中 A_iB_i 分别代表向量 A 和向量 B 的各分量。

在式（2）中 $tf_j(w_i)$ 的含义为当前关键词 w_i 在文本 j 中出现的频率，N 则表示总文本数，$df(w_i)$ 则表示的是当前关键词 w_i 出现在了多少个文本中。

其中基于余弦相似度获取标准问算法（get_standard_question_by_cosSim）的核心步骤如下：

Step1：注入当前客户问以及知识库中的标准问

Step2：数据预处理

Step3：使用 TF-IDF 分别对客户问及标准问进行文本向量化

Step4：使用余弦相似度计算当前客户问与标准问的相似度

Step5：排序输出与当前客户问最相似的 TOP5 个标准问列表

除了传统的解决方案外，经过分析和讨论，我们认为该问题可以转化为机器学习中的多分类问题。于是变从该角度入手，运用机器学习中的常用分类算法针对本赛题进行建模分析，最终采用了朴素贝叶斯算法的模型并加以优化，最后构建了整体算法框架中另外一个核心标准问匹配算法——get_standard_question_by_NB。该算法的训练过程伪代码如下：

表 2-3 get_standard_question_by_NB 伪代码符号说明表

符号	说明
get_standard_question_by_NB	基于朴素贝叶斯获取标准问
train_set	训练集
model	模型文件
load_data(self)	加载数据
get_question_txt()	得到模型文件（question）
prepare(self)	到模型文件（wordlist）
cal_probability(self)	计算先验/后验概率
save_model(self)	保存模型文件（model）

算法名称： get_standard_question_by_NB

输入： train_set

输出： model

（1）def load_data(self):

（2）def get_question_txt():

（3）def prepare(self)：

（4）def cal_probability(self)：

（5）def save_model(self):

2.3 算法优化

2.3.1 get_standard_question_by_NB 算法优化

该算法核心部分由朴素贝叶斯分类器组成，在给定目标值时假定属性之间相互条件独立的情况下，则该分类器可由以下式子表示：

$$P(\text{Category}|\text{Document}) = \frac{P(Document|Category)*P(Category)}{P(Document)} \quad (3)$$

在算法实现过程中，出现了没有出现过的词概率为 0 的情况以及在概率求积中遇到浮点数溢出的情况。针对该两种情况，我们讨论分析并在查找相关资料后作出了以下处理：

1） 加入拉普拉斯平滑参数。

2） 对贝叶斯公式中的分子取对数。

2.3.2 get_standard_question_bt_compose 算法优化

我们采用控制变量法对基于 NB 的标准问获取算法和基于余弦相似度的标准问获取算法进行并行实验，最后在测试集中我们发现两种单模型在 F1 指标值上分别只有 0.8 和 0.7 左右的表现。为分析模型误判的原因，我将误判的数据集提取出来进行可视化分析：

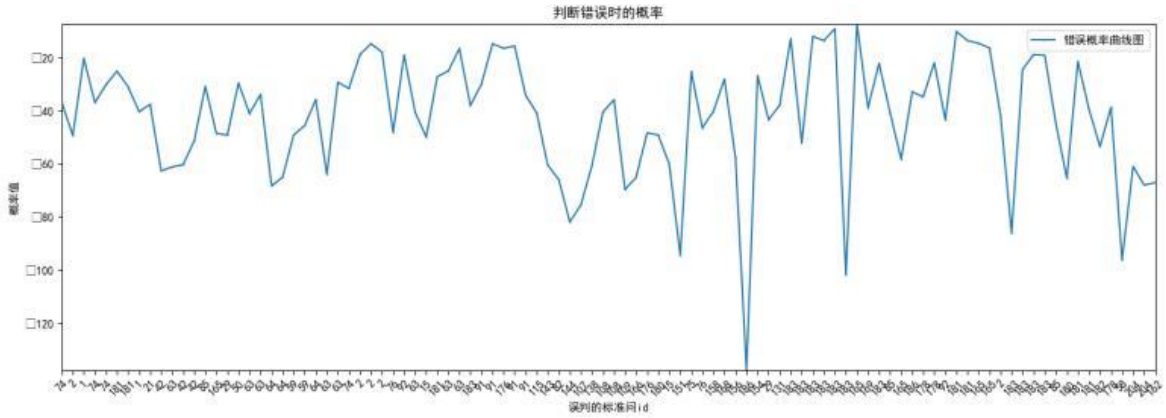

图 2-7 误判数据概率分布图

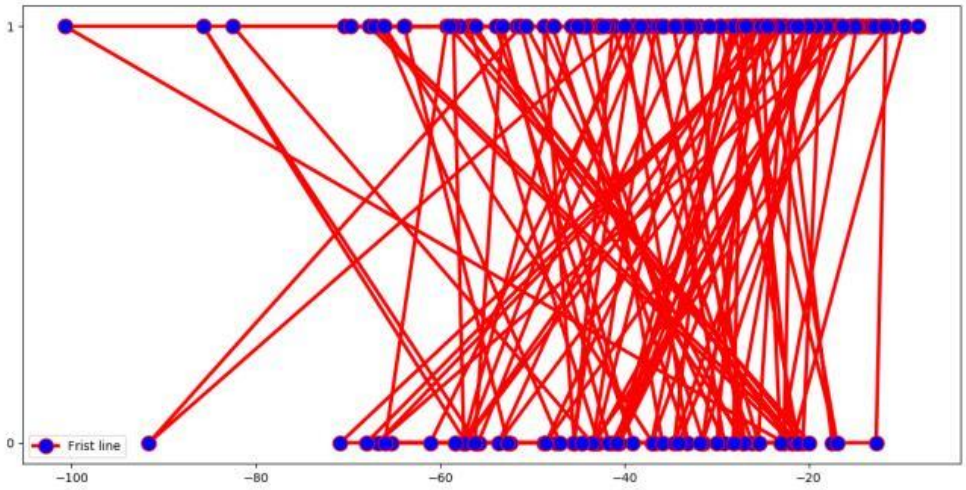

图 2-8 测试数据 0 1 分布图

通过分析图 2-7，我们发现在基于朴素贝叶斯的标准问获取算法下被误判的标准问所对应的概率值大部分集中在一定的范围值中。从图 2-8 的分析中，我们发现判断正确（标记为 1）的标准问和判断错误（标记为 0）的标准问所处的概率值范围也相近。综合图 2-7 和图 2-8，未能得出较好的结论。

但是，在通过控制客户问不变的情况下，两种获取标准问算法的表现在某些客户问下的表现却截然不同，如下图所示：

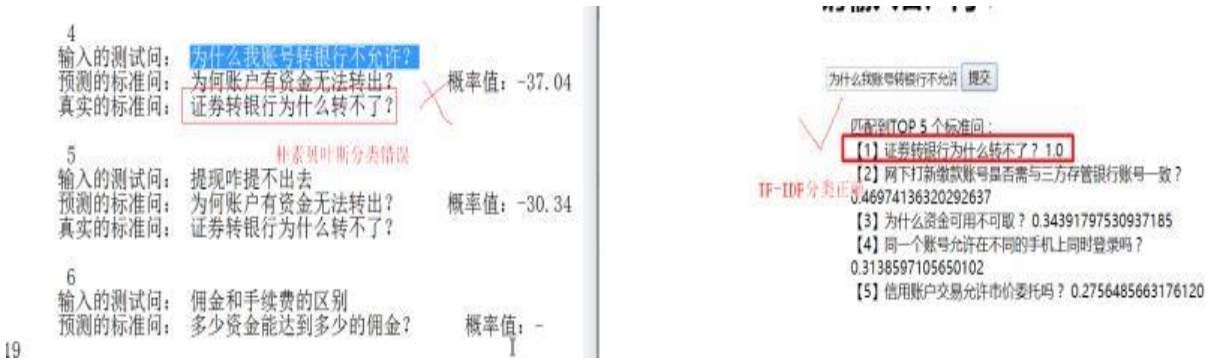

图 2-9 单模型算法测试客户问对比图

由图 2-9 可看出，虽然在整体 F1 值的表现中基于余弦相似获取标准问算法略逊一筹，但是在例如“为什么我账号转银行不允许？”这个客户问下，基于余弦相似度的获取标准问算法比基于 NB 的获取标准问算法判断的更加正确。经过多次试验，在两个算法输出的标准问 TOP5 列表中，我们发现其匹配相同的标准问但是在列表的排列顺序上却有所差异。如果客户问中存在与标准问过多的关键词，则基于余弦相似度的算法更能匹配出正确的标准问。

对此，我们设计了基于组合模型获取最终标准问列表算法，伪代码如下：

表 2-4 get_standard_question_bt_compose 伪代码符号说明表

符号	说明
get_standard_question_bt_compose	基于组合模型获取最终的标准问
sq_list_by_NB	基于朴素贝叶斯匹配到的 TOP5 标准问列表
sq_list_by_cosSim	基于余弦相似度匹配到的 TOP5 标准问列表
top5_list	最终匹配到的 TOP5 标准问列表
get_list(self)	获取模型产物
get_same_item(self)	获取相同的标准问
ranking(self)	重新排名
get_top5_list	获得最终匹配到的 TOP5 标准问列表

算法名称： get_standard_question_bt_compose

输入： sq_list_by_NB, sq_list_by_cosSim

输出： top5_list

（6）def get_list(self):

（7）def get_same_item(self):

（8）def ranking(self):

（9）def get_top5_list(self):

2.3.3 filter_by_IR 算法优化

在进行算法的组合后，我们模拟真实用户对整个问答系统进行人工自检测试。在此过程中，我们发现如果用户提出了一些“闲聊”式的提问，如“天气不错鸭”在之前版本的系统中则仍会匹配到一个标准问列表出来，而实际上此处应该转化为“闲聊”型机器人对话模式。为进一步提高用户体验，我们受到信息检索角度领域技术的启发，设计了基于信息检索角度过滤部分客户问。例如，当客户输入“天气不错鸭”时，我们提示他是否转为人工客服？使得主要业务场景得以保留并提高了用户体验。

其中，filter_by_IR 算法核心步骤如下：

Step1：将当前标准问下的所有客户问“合并”成一篇“文章”。

Step2：对客户问进行数据清洗，提取出关键词。

Step3：计算客户问中所有关键词出现在所有“文章”中的 TF-IDF 的值。

Step4：若对应的 TF-IDF 值为 0 则判定为“闲聊”式提问直接转人工客服。

3 后端部署

3.1 模块功能要求

1）与前端进行及时有效的数据传输

2）与算法模块进行数据间的交互

3）算法模块部署在后端服务器上且能流畅的运行,在接收到客户问后能根据该客户问返回 TOP5 匹配的标准问列表或者返回是否转人工客服的提示信息。

3.2 后台管理系统模块

为了方便数据管理，使得整个系统更具扩展性和便于管理。我们设计并实现了一个简易的智能问答后台管理系统。其中后台管理系统部分功能模块如下：

图 3-1 后台管理系统登录页面

图 3-2 后台管理系统首页

图 3-3 数据管理页面

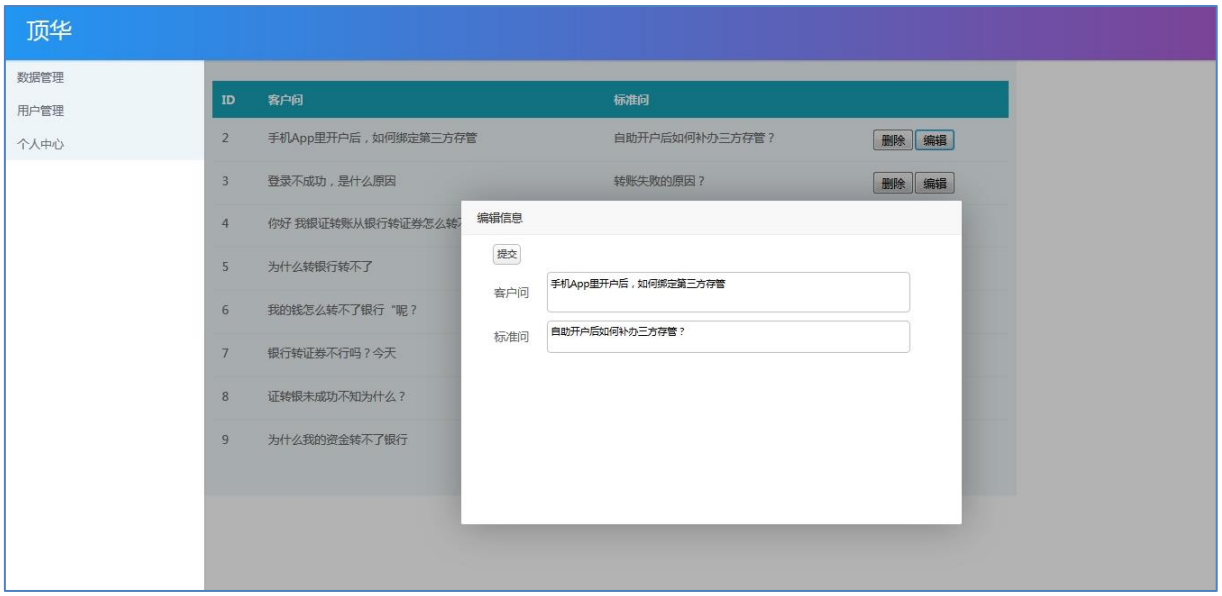

图 3-4 数据信息编辑页面

3.3 实现技术与部署流程

3.3.1 数据交互

软件整体设计前端后分离，所有的数据交互皆通过统一的 JSON 数据格式进行传输。

例如，前端 webAPP 通过 JQuery 发起 POST 请求，后端处理后得到 JSON 格式的数据结果。

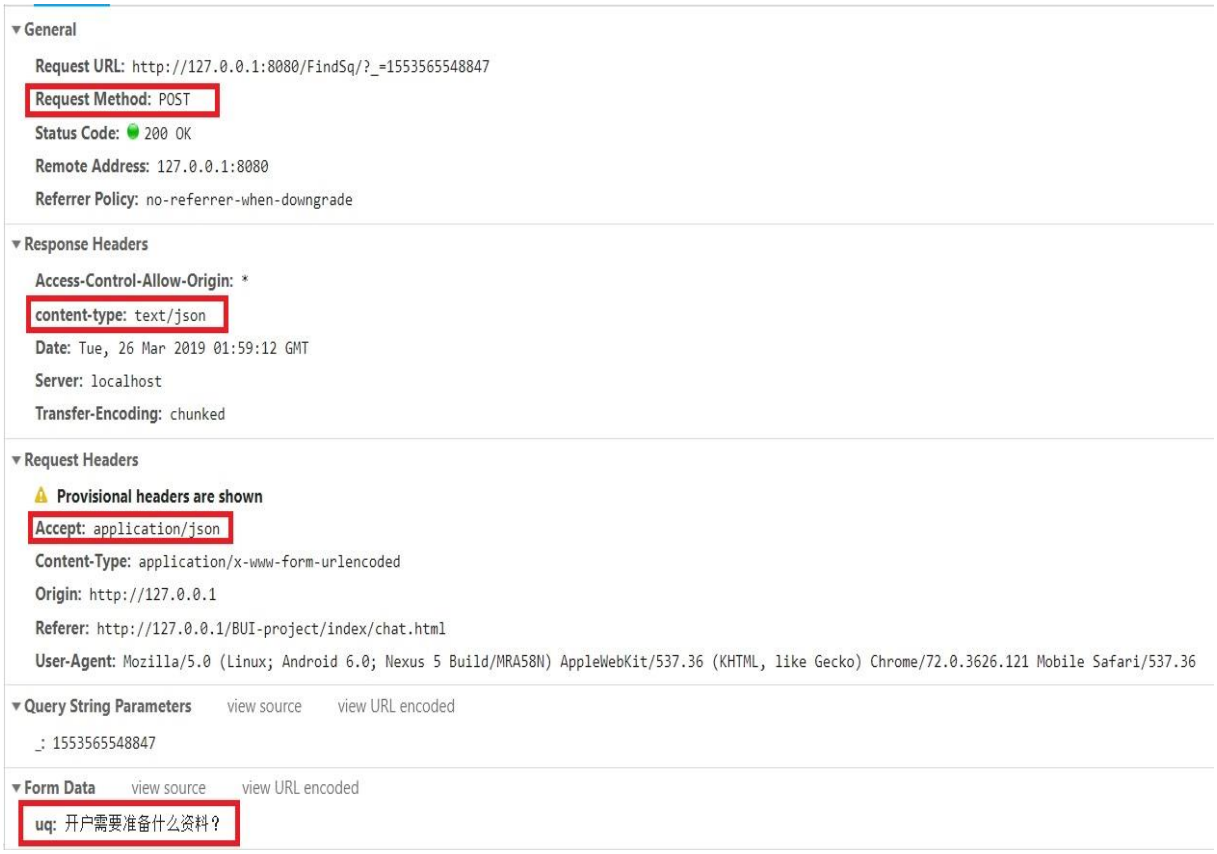

图 3-5 前端发起请求（本地调试）图

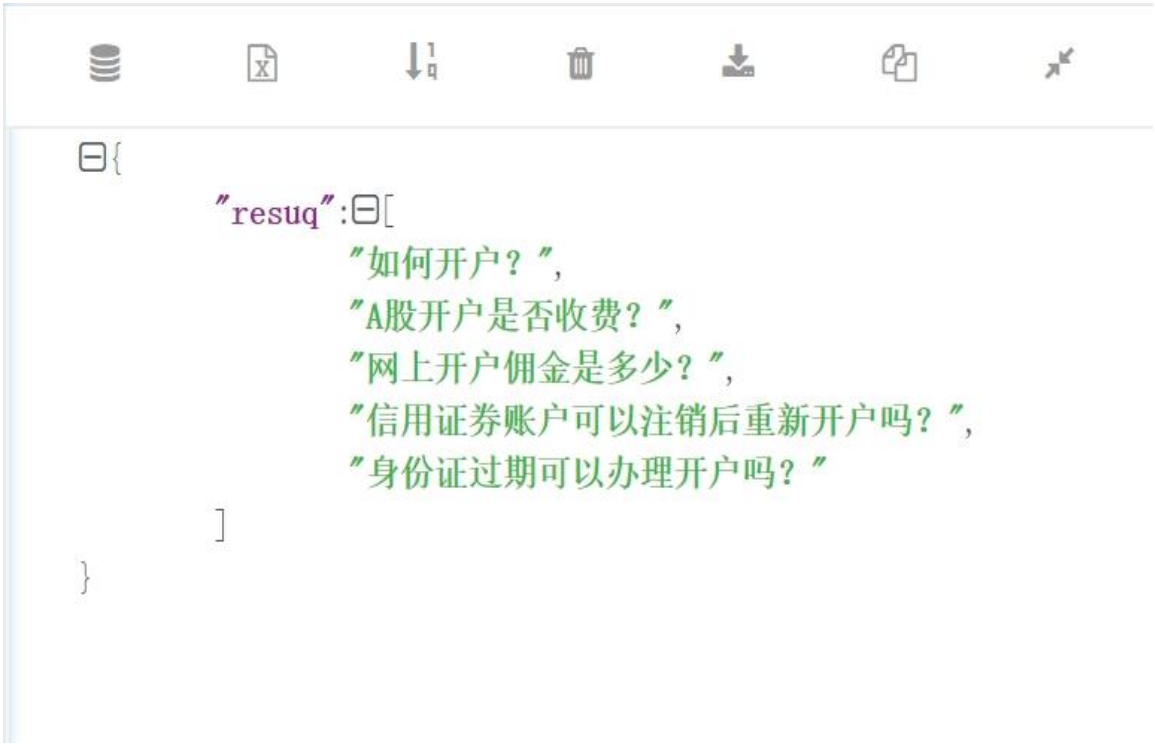

图 3-6 后端返回数据格式图

3.3.2 web 框架

在搭建后端服务时，我们采用了 web.py 这个框架。web.py 是一个轻量级 Python web 框架，它简单而且功能强大。web.py 是一个开源项目。该框架由已故美国作家、Reddit 联合创始人、RSS 规格合作创造者、著名计算机黑客 Aaron Swartz 开发。web.py 目前已被很多家大型网站所使用。当运行主文件时，该框架会自动调起服务，默认端口为 8080，监听所有发送过来的请求。

```
#URL映射
urls = (
    '/(.*)', 'FindSq'
)
app = web.application(urls, globals())

class FindSq:

    def POST(self, uq):
        # 跨域
        web.header("Access-Control-Allow-Origin", "*")
        web.header('content-type', 'text/json')
        data = web.input()
        user_question = data['uq']
        sq_list_by_cosSim = tfidf_cosin_main.main_get_top_n(user_question)
        nbp = N.NavieBayesPredict('./model.txt')
        sq_list_by_NB = nbp.predict(user_question)
        get_top5_list = get_standard_question_bt_compose(sq_list_by_cosSim, sq_list_by_NB)
        res_dic = {'resuq': get_top5_list}

        return json.dumps(res_dic)

if __name__ == "__main__":
    app.run()
```

图 3-7 web.py 处理请求部分代码截图

3.3.3 部署流程

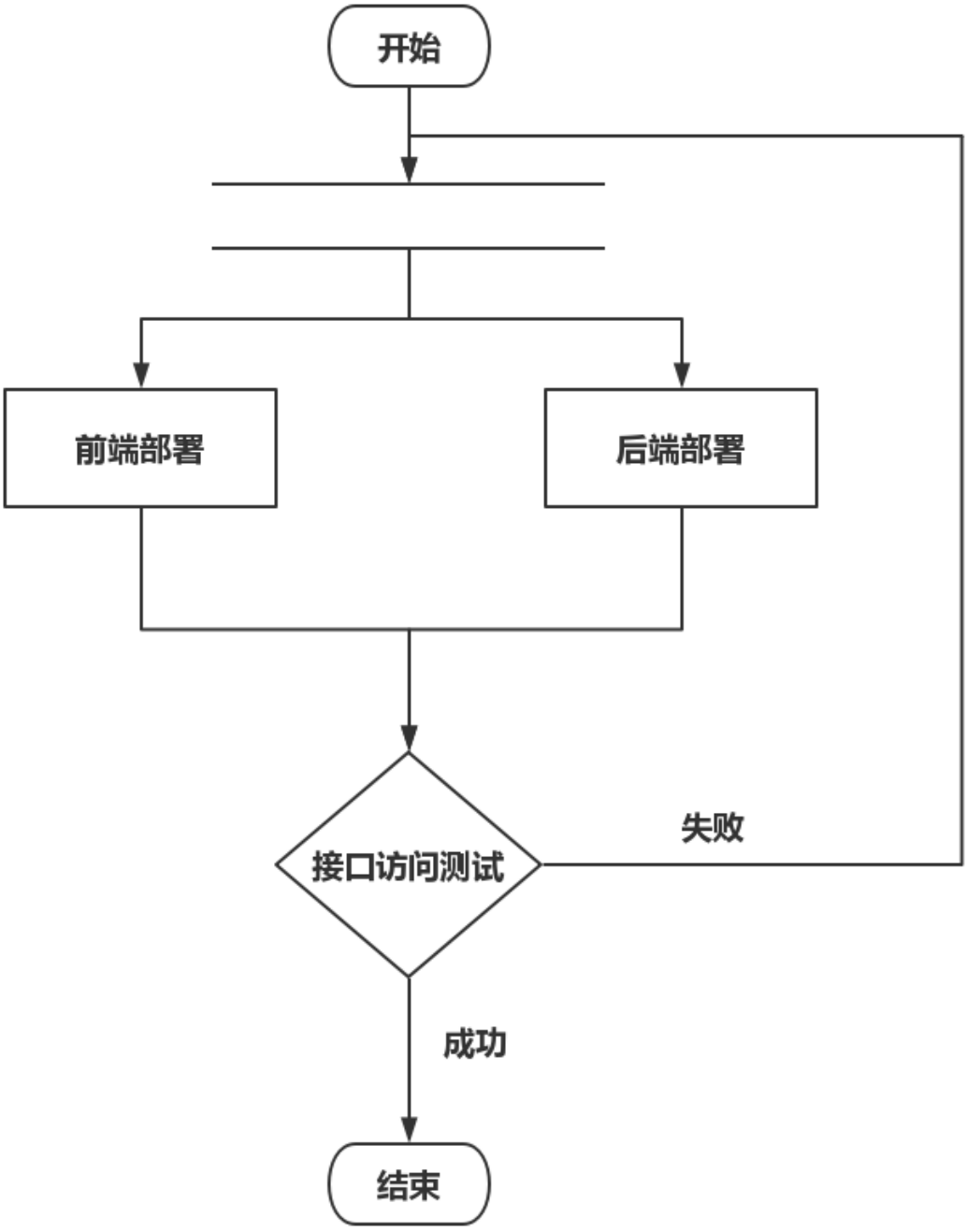

图 3-8 系统部署流程图

如图 3-8 所示，系统部署主要流程分为三大部分。

第一部分为前端部署，将 webAPP project 部署到 Apache 服务器上，并开启服务器，默认端口为 80 。

第二部分为后端部署，将核心算法模块与 web.py 搭建起来，并运行 web.py 主文件，默认端口为 8080，并使得该主文件一直运行着，不断的监听发送过来的请求。

第三部分为测试接口，打开前端 webAPP 在客服页面输入客户问，然后点击“发送”测试后端服务是否正常运行。若存在异常，则根据异常信息重新部署；若能接收到后端返回的匹配到的标准问列表则表示部署成功。

4 前端设计

4.1 模块功能要求

- 实现 QA 对话界面
- 实现问题引导功能
- 实现反馈功能
- 与后台管理进行交互

4.2 开发环境

表 4-1 开发环境表

操作系统	Windows10
开发工具	Sublime Text 3
开发模式	Web app
使用的框架	BUI（Build In User Interface）
使用的服务器	Apache HTTP Server

4.3 实现技术与流程

我们使用 Web app 为本次系统的开发模式,与原生 APP 相比，其开发成本低，开发速度快，具有跨平台、统一性高的优势。开发使用了 BUI 框架，提供了丰富的 DPL 含有强大的控件库，灵活地开发了用户端使用的界面，且界面简洁美观，交互良好。可以将用户提问后匹配到的问题以对话的形式呈现给用户。以下是 Web app 部分设计图：

✧ 主要用例图

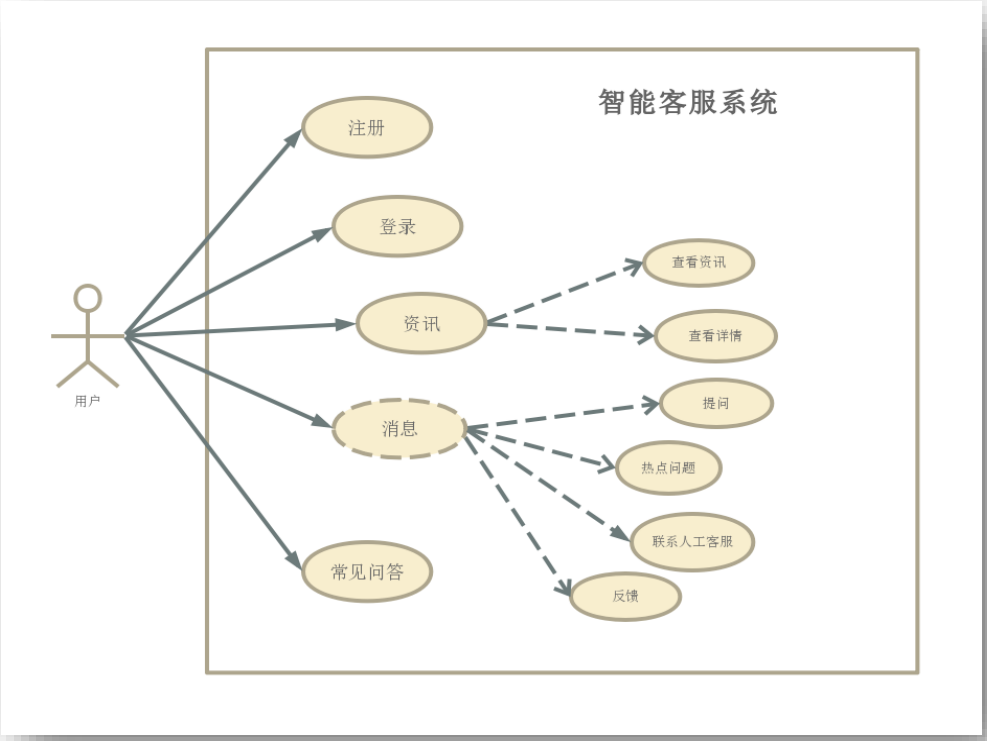

图 4-1 WEB 端设计主要用例图

✧ 主要顺序图

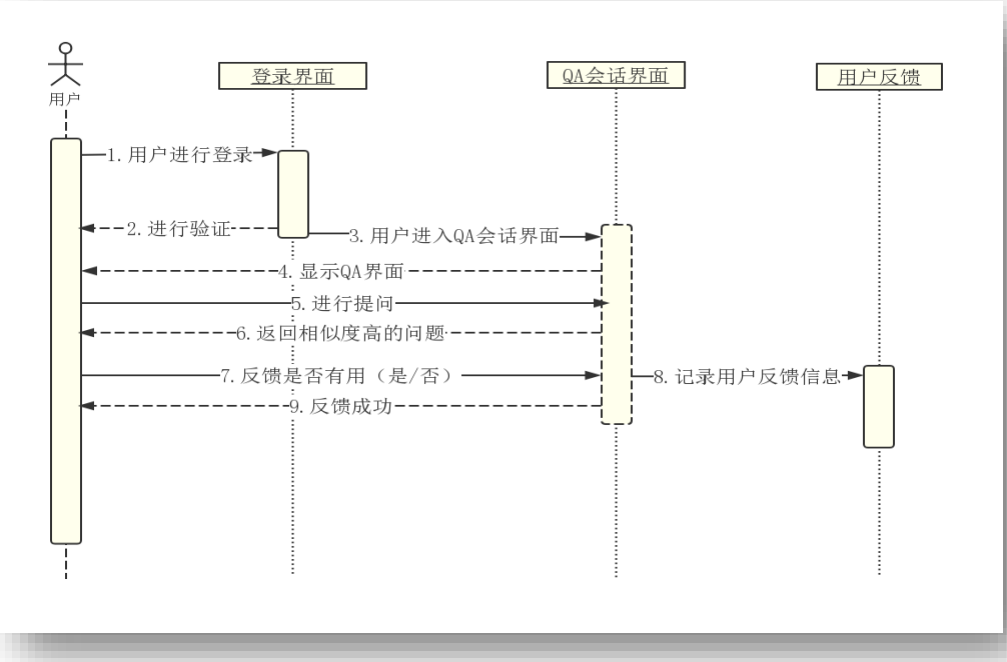

图 4-2 WEB 端设计主要顺序图

4.4 界面设计

本次系统的界面设计，整体以蓝调为主，以简洁美观为主要特点，融合了扁平化设计，使得整体界面的观赏性较好。以下是前端界面的截图及介绍：

✧ **登录页面：**

图 4-3 登录页面

✧ **注册页面：**

图 4-4 注册页面

✧ **资讯页面：**

图 4-5 资讯页面

✧ 新闻详情页面：

图 4-6 详情页面

✧ QA 会话页面：

图 4-7 QA 会话页面

✧ 人工客服页面：

图 4-8 人工客服页面

✧ 热门问答：

图 4-9 热门问答页面