互联网 + 法律

摘要

对于给定数量的律师的历史回答，本文将给出一个合理的解决办法来设计对应的律师推荐系统，最终做出一个法律机器人从而识别律师擅长的法律领域，评估律师的回复水平并进行自动回复。

本文通过分词和模仿 NLTK 库实现对 3000 个律师的历史问答的词频分析，然后通过人工选词，选出在不同法律领域具有独特代表性的特征词语。并且要求这些特征词语仅在其对应的领域出现，我们认为当特征词语在问题中只要出现，即视为特征词语代表的法律领域被问及，对 3000 个律师的历史回答用这些进行筛选，我们统计出每个律师在不同领域被问及的数目，经过排序，我们认为排名前 1500 的律师擅长该领域。最终我们得出民事民法最擅长的是 2899 号律师，但他仅擅长该领域。对经济金融最擅长的律师是 2583 号，而他还擅长民事民法，公司企业，刑事行政。对刑法行政最擅长的律师是 2315 号，他还擅长民事民法和其他，对涉外纠纷最擅长的律师是 2839 号，他还擅长经济金融，公司企业和其他。对公司企业最擅长的律师是 2583 号，他还擅长民事民法，经济金融，刑事行政。对其他类别最擅长的律师是 2315 号，他还擅长民事民法和其他。我们认为律师的回复水平与回复的长度有极强的正相关关系，因此我们对律师回答问题的长度进行统计选取回答最多的认为回答的质量越高，回答质量前三的是 652 号律师，776 号律师, 58 号律师。

通过上述工作，我们通过对律师擅长的领域和律师回答质量这两项进行平权统计得出在不同领域回答最好的前 10 位律师（这里只展示每个专业领域的综合第一）民事民法合适的律师是 1179 号，, 经济金融合适的律师是 1928 号，对刑法行政合适的律师是 2863 号，对涉外纠纷合适的律师是 353 号对公司企业合适的律师是 1886 号，对其他类别合适的律师是 1900 号。

对于法律机器人，我们采用，在客户输入的案情描述搜索我们在六个专业领域中挑选出的关键词，从而确定案情属于的专业领域。挑选若干位在该领域排名靠前的律师，在他们的回复中通过匹配关键词进行查找，选出最符合的回答并将其输出。经检验本方法具有一定的可行性。

关键字：中文分词　统计分析　特征关键词　平均回答长度

目录

一、问题重述

我们寻求通过“法律机器人”、智能法律服务等方式，为公众提供质优价廉的法律咨询服务，进而提高律师的服务效率，为法官提供精准的判决参考。

现在有这样一个连接公众法律需求与律师的互联网平台，公众可就自己遇到的法律问题进行咨询。

基于这样一个平台，我们要达到智能法律服务这一目标我们需要做三件事：

1. 建立识别律师历史回答的问题涉及的法律领域的模型，并且建立指标确定律师的擅长领域以及评价律师回答质量模型。
2. 对于特定问题如何将其进行法律领域的分类，并将该问题推送给最擅长解决这类问题的若干律师。
3. 根据客户输入的案情描述，设计数学模型和算法，自动给出一个质量较高的答复。

二、问题分析

在题目中，有这样一个连接公众法律需求与律师的互联网平台，公众可就自己遇到的法律问题进行咨询。如果某个律师感兴趣，可对咨询问题进行回复，进而争取客户资源。题中给出了从该平台上收集的一些公众客户提问，以及律师的回复。每一个文件代表一个律师的回复历史。

我们首先要做的是通过这些回复历史找出每个律师擅长的专业领域。从律师的角度来说，他选择回答一个问题，说明他认为自己会回答，认为自己擅长该领域。而在评估律师专业水准上，我们通过浏览大量回复发现，许多律师回复极其简短，很不用心。有的律师只简单给出不可以，没有任何阐释。有的律师直接把问题抛给有关部门。因而我们认为回复得越认真细致，即字数越长，从一定程度上能反应他的回复水平越高。首先我们利用机器学习处理 3000 个文档，进行了分词和词频统计。在这些基础上，我们建立了特征词语匹配模型，用以解决对律师擅长领域的分类问题以及评价律师回答的好坏。

然后我们依据提出的问题中的关键词对其涉及的法律方面进行归类，并将在相关领域的前几位律师推荐给客户。

最后我们需要综合前两问提出智能回答系统的建立过程。

三、基本假设

1. 由于网站接收问题后，会将问题推送给相关的律师，律师通过选择并对问题进行回答，律师在某个领域回答问题的数量与他在该领域的擅长程度成正比，因此律师擅长的方向仅从平台推送给律师的问题中获得。
2. 律师回复的字数长度与他的回复水平成正比，我们认为回答的长度越长，即对问题的回答越详细，即认为律师的回复水平高。

3. 选出的频数高的词语代表的法律专业领域没有重合，即我们认为我们选出的词具有极强法律领域的代表性。
4. 我们抽取的 3000 个律师，每个律师的回答问题的数目基本相同。

四、模型设计思路

在建立特征词语匹配模型前，首先需要找到合适的特征词汇。对于特征词汇的采集，我们首先利用基于最大熵隐马尔科夫模型的分词器对 3000 篇材料进行预分词处理，并且通过设计无效词排除、关键词统计程序筛选出接近目标特征词汇需求的高频备选词。

在得到筛选出的备选词之后，我们通过人工分类的方法筛选出符合各个领域的特征词汇，最后将各大领域中细分领域的特征词汇进行整合和评估，在六大领域中，每个领域选出十个核心关键词用于特征词汇匹配模型。

4.1 最大熵原理

当熵最大时，则表示该系统内各随机事件 (变量) 发生的概率是近似均匀的，等可能性的，根据这一性质我们可以将已知事件作为约束条件，作出最不偏倚的假设，求得可使熵最大化的概率分布。

我们先引入特征函数 $f(x,y)$，$f(x,y)$ 是一个二值函数，表示当 x,y 满足某一事实其特征函数值为 1。

$$f_i(x,y) \in 0,1, \quad i=1,2,3,\cdots,m \tag{1}$$

在真实的语言环境里，某一观测值对应的隐藏状态是由上下文环境 (观测，状态) 决定的，引入特征函数可使我们能够自由的选取特征 (观测或状态的组合)。可以说是用特征 (观测组合) 来代替观测，避免生成模型 HMM, naive bayes 的观测独立性假设的局限性。

我们可以根据大小为 T 的训练数据 $D=(x,y)$ 得到一个经验期望和模型期望。

$$\widetilde{E}(f_i)=\frac{1}{n}\sum_{x,y}p(x,y)f_i(x,y) \tag{2}$$

$$E(f_i)=\frac{1}{n}\sum_{x,y}p(x)(y|x)f_i(x,y) \tag{3}$$

我们假设经验期望与模型期望相等，那么就存在多个满足此约束的有关任意特征函数 f_i 的条件概率分布的集合 C，于是有:

$$C=P|E_p(f_i)=\widetilde{E}_p(f_i), i=1,2,\cdots,m \tag{4}$$

最大熵原理认为，从不完整的信息（例如有限数量的训练数据）推导出的唯一合理的概率分布应该在满足这些信息提供的约束条件下拥有最大熵值——即熵最大的分布在条件概率集合是最优的，那么最大熵模型变为凸函数的约束优化问题

$$\begin{aligned} \max_{P\in C} H(P) &= -\sum_{x,y} P(x)P(y|x)logP(y|x) \\ s.t. \quad E_p(f_i) &= \widetilde{E}_p(f_i), i = 1, 2, \cdots, m \\ s.t. \quad \sum_y p(y|x) &= 1 \end{aligned} \tag{5}$$

我们通常使用拉格朗日对偶原理来将原式变形为无约束的极值求解：

$$L(\omega, \alpha, \beta) = f(\omega) + \sum_{i=1}^{k} \alpha_i g_i(\omega) + \sum_{j=1}^{l} \beta_j h_j(\omega) \tag{6}$$

$$\Lambda(p, \widetilde{\lambda}) = H(y|x) + \sum_{i=1}^{m} \lambda_i (E_p(f_i) - \widetilde{E}_p(f_i)) + \lambda_{m+1}(\sum_{y\in Y} P(y|x) - 1) \tag{7}$$

在拉格朗日函数对 p 求偏导，并使之等于 0，求解方程，可得下式：

$$P^*_{\bar{\lambda}}(y|x) = \frac{1}{Z_{\bar{\lambda}}(x)} exp\left(\sum_{i=1}^{m} \lambda_i f_i(x, y)\right) \tag{8}$$

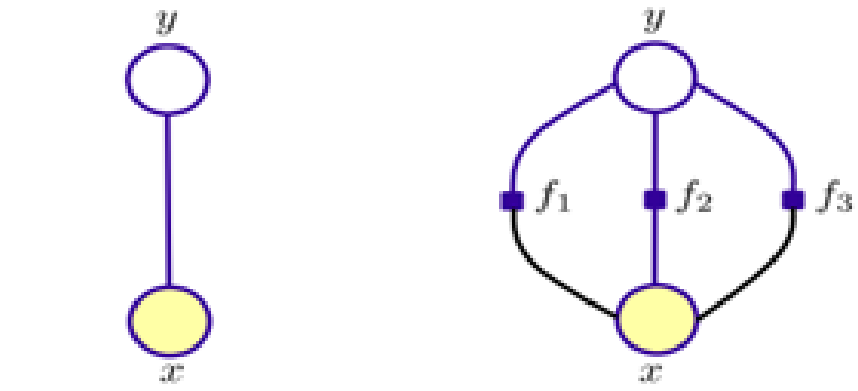
$$Z_{\bar{(\lambda)}}(x) = \sum_{y\in Y} exp\left(\sum_{i=1}^{m} \lambda_i f_i(x, y)\right) \tag{9}$$

其中是模型中各个特征函数的参数向量，Z 是以观测序列 X 为条件概率的归一化因子，其意义是将复杂的联合分布分解为多个因子的乘积 (最大团)，实质是得到归一化因子 $Z(x)$ 均衡给定 x 任意 y 的条件概率分布数值 (局部归一)，最大熵模型学习过程就是估计出这两种有关 x, y 的参数

$$Z_{\bar{(\lambda)}}(x) = \sum_{y\in Y} \prod_{i=1}^{m} exp(\lambda_i f_i(x, y)) \tag{10}$$

4.2 最大熵马尔可夫模型

最大熵模型是在已知经验分布的基础上求解有关特征函数 $f(x, y)$ 的最优的 $P(y|x)$ 概率分布，但它的随机变量 y 有相互独立的假设，所以不能很好的描述 y_i, x_i 与 y_{i-1} 的关系，而 HMM 又有观测独立性假设不能自由的选择特征，所以我们希望找到一个能同时服从马尔可夫性假设和服从最大熵假设的模型解决序列标注的问题。



(a) Independency graph　　(b) Factor graph

图 **1**　两种图模型

4.2.1　对比隐马尔可夫模型 (HMM)

$$P(X) = \sum_{y} \prod_{i=1}^{T} p(y_i|y_{i-1}) p(x_i|y_i) \tag{11}$$

状态序列 Y，观测序列 X，两个状态转移概率: 从 y_{i-1} 到 y_i 的条件概率分布，状态 y_i 的输出观测概率，初始概率。隐马尔可夫模型依赖于已知数据的概率分布，已经历史经验来决定现实决策，但实际能提供训练的数据是少量且稀疏的，我们不能枚举所有的数据分布状况，所以需要在数据稀疏的条件下估计未知 x, y 的条件概率。

4.2.2　最大熵马尔可夫模型 (MEMM)

$$P_{y_i-1}(y_i|x_i) = \frac{1}{Z_{\bar{(\lambda)}}(x_i, Y_{i-1})} exp\left(\sum_{a}^{m} \lambda_a f_a(x_i, y_i)\right), \quad i = 1, 2, \cdots, T \tag{12}$$

用分布来替代 HMM 中的两个条件概率分布，它表示从先前状态，在观测值下得到当前状态的概率，即根据前一状态和当前观测预测当前状态。每个这样的分布函数都是一个服从最大熵的指数模型。

图 **2**　左为 **HMM** 模型，右为 **MEMM** 模型

4.2.3 总结

HMM：双状态转移条件概率的输出独立性假设，在训练过程通过统计的联合概率，在解码中通过联合概率反向递推的条件概率。

MEMM：限定条件求最优条件概率分布，在训练过程中收敛一组有关特征函数的参数向量，以及有关与的正则化因子，在解码过程中直接求得条件概率。

最大熵马尔科夫模型 [2] 把 HMM 模型和 Maximum-Entropy 模型 [?] 的优点集合成一个产生式模型 (如图 (2) 所示)，这个模型允许状态转移概率依赖于序列中彼此之间非独立的特征上，从而将上下文信息引入到模型的学习和识别过程中，提高了识别的精确度，召回率也大大的提高。

五、问题一模型建立与求解

为了建立数学模型，找出每个律师擅长的某个或几个专业领域，并对律师回复水平及专业水准进行评估，并用实例验证模型，我们进行了以下工作：

1. 将每一篇回复历史进行分词和词频统计。
2. 综合所有回复，人工挑选出最能代表六个专业领域且词频较高的关键词。
3. 分别将这些关键词与 3000 份律师的回复进行匹配，别分统计六个专业领域出现词频最高的前 50% 的回复文档，将它们对应的律师判定为擅长该专业领域。
4. 检索在某个领域排名靠前的若干位律师的文档，阅读他的回复，进行判断来验证模型。

5.1 词频统计

在词频统计中，我们首先通过 Python 中分词开源库搜索引擎模式完成材料的词汇划分与提取。再通过 C++ 编制了一套仿照 Python 中 nltk 包的词频特性统计程序完成词频的统计分析与无效字符剔除。

为了降低人工筛选特征词的工作强度，我们首先对 3000 份文档进行随机抽样，从 3000 个律师的材料中随机抽取 100 份文档进行词频分析，我们认为这 100 份文档产生的所有词可以代表 3000 份文档产生的大部分词语。我们的部分文档的词频分析结果如图 (3) 所示：

5.2 人工筛选

对于已经统计出每个词的频率，可见像诸如“你好”,“法律”等这些词虽然出现的频率极高，但是并不具有本文中六大法律领域的代表意义，因此我们需要明确我们的选词标准，我们对词语的选择标准是：具有明显法律背景，且仅在某一法律领域具有代表性。为提高我们选词的精度，人工筛选分为两次进行。

律师编号： 34
统计词数： 3282
总词数： 17757
你好/ 435 你/ 231 有/ 186 在/ 186 可以/ 162 不/ 153 建议/ 152 离婚/ 150 律师/ 129 给/ 120 起诉/ 117 没有/
115 和/ 112 孩子/ 108 现在/ 104 法院/ 101 赔偿/ 94 具体/ 93 要/ 92 咨询/ 85 如果/ 81 向/ 80 想/ 74 伤残/
72 年/ 69 怎么办/ 69 需要/ 67 被/ 66 委托/ 62 ；/ 61 还/ 61 详细/ 60 （/ 57 ）/ 54 钱/ 53 责任/ 52 怎
么/ 51 申请/ 51 由/ 51 法律/ 48 做/ 48 要求/ 48 工伤/ 47 计算/ 46 等/ 45 双方/ 45 女方/ 45 上/ 44 ，/
44 根据/ 44 去/ 44 工作/ 43 没/ 43 协商/ 43 把/ 43 认定/ 42 什么/ 42 一下/ 42 并/ 42 对/ 40 情况/ 40
自己/ 39 对方/ 38 会/ 38 结婚/ 37 朋友/ 37 当地/ 37 ！/ 37 男方/ 37 工资/ 35 这个/ 35 看/ 35 属于/ 35 进
行/ 34 又/ 34 抚养/ 34 一个/ 32 证据/ 32 儿子/ 31 解决/ 31 2/ 30 谁/ 30 鉴定/ 30 父亲/ 30 写/ 30 报警/
29 可/ 29 条件/ 29 案件/ 29 谢谢/ 29 能/ 29 规定/ 29 办理/ 29 多少/ 28 房子/ 28 3/ 28 过/ 28 当时/ 28
时/ 28 以上/ 28 交通事故/ 27 项目/ 27 因/ 27 这/ 27 不成/ 27 所有/ 26 打/ 26 一般/ 26 买/ 26 直接/ 26 一
直/ 26 结婚证/ 25 及时/ 25 户口/ 25 处理/ 24 地/ 24 支付/ 24 与/ 24 多/ 24 她/ 24 生活/ 23 经济/ 23 中/
23 一年/ 23 一/ 23 以及/ 23 详询/ 23 医院/ 23 单位/ 23 元/ 23 一方/ 22 归/ 22 才/ 22 赔偿金/ 22 只能/ 22
公司/ 22 让/ 22 还要/ 22 个/ 22 好/ 22 找/ 22 固定收入/ 22 按/ 22 这种/ 22 死亡/ 21 分割/ 21 交警队/
21 ?/ 21 标准/ 20 住院/ 20 知道/ 20 认定书/ 20 数额/ 20 下/ 20 同时/ 20 时候/ 19 抚养费/ 19 办案/ 19 房产
证/ 19 原则上/ 19 承担/ 19 1/ 19 劳动/ 19 发生/ 19 呢/ 18 时间/ 18 母亲/ 18 5/ 18 一种/ 18 级别/ 18 骨
折/ 18 交警部门/ 18 过户/ 18 20/ 18 需/ 18 周岁/ 17 土地/ 17 调解/ 17 手续/ 17 主要/ 17 以下/ 17 等级/ 17
收入/ 16 婚前/ 16 共同财产/ 16 构成/ 16 残疾/ 16 走/ 16 已经/ 16 同意/ 16 结果/ 16 费用/ 16 按照/ 16 如何/
16 你们/ 16 岁/ 15 老公/ 15 财产/ 15 结合/ 15 4/ 15 复杂/ 15 合同/ 15 来/ 15 中心/ 15 一次性/ 15 代理/
15 工地/ 15 两周岁/ 15 维护/ 15 生活费/ 15 再/ 15 案情/ 15 房屋/ 14 损害/ 14 很/ 14 10/ 14 或/ 14 跟/
14 老板/ 14 司法鉴定/ 14 父母/ 14 若/ 14 女儿/ 14 事故/ 14 整个/ 14 办/ 14 允许/ 14 车主/ 14 大/ 14 摩托车/
14 确定/ 14 ./ 14 材料/ 13 共同/ 13 另/ 13 民政局/ 13 内容/ 13 诉讼/ 13 不能/ 13 性质/ 13 三年/ 13 麻烦/
13 事故责任/ 13 全权/ 13 6/ 13 误工费/ 13 妻子/ 13 从/ 13 或者/ 13 前/ 13 流程/ 13 营养费/ 13 那/ 13 %/
13 万元/ 13 交通费/ 13 抚慰金/ 13 于/ 13 期间/ 13 涉嫌/ 13 精神/ 13 已/ 13 造成/ 13 注意/ 12 员工/ 12 补助
费/ 12 “/ 12 只有/ 12 9/ 12 欠/ 12 护理费/ 12 医疗费/ 12 协议书/ 12 ”/ 12 感情/ 12 怎样/ 12 回来/ 12
其/ 12 上班/ 12 用/ 12 夫妻/ 12 判决/ 12 那样/ 12 有效/ 12 去年/ 12 拿/ 12 金钱/ 12 浪费时间/ 12 更好/ 12
请/ 12 走弯路/ 12 免得/ 12 犯罪/ 12 诉状/ 12 代写/ 12 无/ 12 亲自/ 12 利益/ 11 综合/ 11 处罚/ 11 而且/ 11
家里/ 11 一起/ 11 通过/ 11 涉及/ 11 补助金/ 11 派出所/ 11 万/ 11 关于/ 11 算/ 11 搜集/ 11 提出/ 11 自愿/
11 投诉/ 11 超过/ 11 天/ 11 伙食/ 11 只/ 11 一名/ 10 两种/ 10 婚姻/ 10 方式/ 10 领取/ 10 上诉/ 10 表述/
10 今年/ 10 以/ 10 家/ 10 登记/ 10 实际/ 10 将/ 10 复核/ 10 借条/ 10 车/ 10 能否/ 10 日/ 10 证明/ 10
比较/ 10 像/ 10 男/ 10 大概/ 10 内向/ 10 按规定/ 10 合适/ 10 司机/ 10 知识点/ 10 这样/ 10 政策/ 9 得/ 9
抚养权/ 9 子女/ 9 公安局/ 9 不服/ 9 下来/ 9 医疗/ 9 而/ 9 清楚/ 9 妹妹/ 9 关系/ 9 老婆/ 9 》/ 9 拆迁/
9 房/ 9 当面/ 9 不要/ 9 着/ 9 很难/ 9 号/ 9 《/ 9 至/ 9 几年/ 9 判处/ 9 啊/ 9 手机/ 9 三/ 9 治疗/
9 及/ 9 咋办/ 9 回家/ 9 调查/ 9 很多/ 9 病/ 8 尽快/ 8 还款/ 8 无法/ 8 检查/ 8 由于/ 8 不想/ 8 介入/

图 **3**　部分分词统计结果

初步筛选：阅读 100 份词频统计结果，将符合上述结果的词添加至法律子项目，结果如图 (4) 所示：

民事民法		经济金融		刑事行政	涉外纠纷			公司企业	公司	其他类别	
离婚	离婚	房产纠纷	中介 业主 买房 房产证 开发商 房子 公积金 首付 按揭 分期	刑事辩护	抗诉 量刑	海事海商		法律顾问		个人独资	
交通事故	交通事故	建设工程	商品房 违建	取保候审	检察院	合资合作		公司解散	企业 裁员	工商查询	工商局
婚姻家庭	结婚证 户口 抚养费 抚养权 同居 赡养 出轨 怀孕 养老 计生办 家具	银行	银行 本金	暴力犯罪	故意伤害	涉外法律	大使馆 加拿大 美国 国外	公司法		合同审查	合同
工伤赔偿	协商 伤残 工伤 抚恤金 治疗费	知识产权	产权	毒品犯罪	犯罪，毒	涉外仲裁	仲裁	公司上市	营业执照 牌照	矿产资源	
劳动纠纷	误工费 劳动合同 包工头 拖欠	金融证券	交易 基金 利率	经济犯罪	冻结 传销	外商投资	台资	兼并收购		私人律师	律师函 律师费
土地纠纷	土地 自留地 国土局	反不正当竞争		死刑辩护	有期徒刑，全责	国际贸易		股权激励	份额	法律文书代写	委托 法律文书
债权债务	贷款 还款 还贷 借条 约定债务 租赁	经济仲裁	经济	刑事自诉	判决	招商引资		股权纠纷	股东	资信调查	造假，黑名单
医疗纠纷	护理费	加盟维权		公安国安	警察，公安局，			破产清算		电信通讯	短信 移动，
拆迁安置	房子 拆迁	票据	发票 收据	公司犯罪	监狱			改制重组		调解谈判	
抵押担保	高利贷 协议书 抵押	合伙联营	集资	国家赔偿	证据，政府			融资借款		高新技术	专家
合同纠纷	合同 乙方 房东 过户 雇工	招标投标	开发商	海关商检				税务	税 会计	广告宣传	
人身损害	骨折 轻伤 受伤 医药费 肇事伤害 交警			行政复议				新三板		环境污染	
保险理赔	赔偿金 保险 五保户			行政诉讼	败诉 检察院，驳回，打官司			资产拍卖	万元	经销代理	代理商，中介
消费权益	投诉									旅游	
继承	财产 去世									期货交易	
侵权	维权									倾销补贴	
其他										求学教育	
网络法律										水利电力	
行政	民政局 部门 主管部门 工商局 劳动局									WTO事务	
										污染损害	
										新闻侵权	
										移民留学	
										邮政储蓄	

图 **4**　初步筛选结果

精细筛选：每一大的法律领域筛选出十个最具代表性的词或者字用来做为这一领域的特征词，比如像是“受伤”“打伤”“受伤”这类词，我们用“伤”这一个字来代表这一类民事民法，这样做可以降低检索所需时长。按这样的方法，我们每个法律领域选出 10 个词

代表该法律领域，选出的结果，如表 (1) 所示：

表 **1**　关键词筛选结果

民事民法	婚	交通	车	同居	出轨	疗	假	贷	借	拆
经济金融	房	公积金	首付	分期	产权	交易	基金	利率	发票	集资
刑事行政	刑	政	贿	判	狱	公安	局	委	暴	犯
涉外纠纷	仲裁	台资	贸易	美国	大使	国外	合资	外资	海	加拿大
公司企业	公司	企	股	收购	税	万元	资产	份额	执照	裁员
其他类别	函	文书	委托	短信	移动	联通	电信	广告	环境	代理

5.3　匹配选出律师擅长的领域

5.3.1　标准建立

我们已经选出代表每个领域的特征词，我们在对每个律师匹配之前，我们需要对擅长的领域建立一个标准。每个律师在选择他要回答的问题的时候就已经表明他在这六个领域所擅长的某一个领域或某几个领域，我们用每个律师在每个领域回答问题的数量做为律师在这几个领域的评分，通过将每一领域这 3000 名律师的评分选取前 1500 名的律师，我们认为这样的律师是擅长这一领域的。例如在民事民法这一领域，排名前 1500 名的律师我们认为，该律师擅长民事民法。

5.3.2　算法分析

该算法的思想是用已经选出的 6 个法律领域的 60 个词去匹配问题。具体算法是，首先先对这 3000 个律师中的所有问题用民事民法中的 10 个特征词进行筛选，只要出现十个词中的一个即在民事民法中记一分，并直接对下一个问题进行同样地分析，当分析完所有的民事民法，得到该律师在民事民法的得分。接下来对下一个律师进行同样的民事民法分析，做完 3000 个律师的民事民法分析，通过 3000 个律师得分的前 1500 名，我们认为这 1500 个律师是擅长这一领域的。同理对于其他领域可以进行同样地分析。表 (2) 给出每个领域排名前五的律师作为示例。

并对最擅长各个领域的律师研究了他们其他擅长的方向，如表 (3) 所示：（表中 1 代表擅长，0 代表不擅长）。

表 **2**　各领域律师排名（前五）

民事民法	编号	相关回答	经济金融	编号	相关回答	刑事行政	编号	相关回答
1	2899	383	1	1142	349	1	2315	389
2	2583	377	2	1287	315	2	998	379
3	893	342	3	497	295	3	538	369
4	809	341	4	647	290	4	1485	366
5	845	341	5	2711	275	5	180	360
涉外纠纷	编号	相关回答	公司企业	编号	相关回答	其他类别	编号	相关回答
1	2839	118	1	2583	325	1	2315	384
2	1798	103	2	2839	240	2	998	374
3	2373	102	3	2063	222	3	538	357
4	353	96	4	1245	185	4	162	341
5	1390	96	5	805	178	5	13	340

表 **3**　各领域排名第一的律师的领域分析

编号	民事民法	经济金融	刑事行政	涉外纠纷	公司企业	其他类别
2899	1	0	0	0	0	0
2583	1	1	1	1	0	0
2315	1	0	1	0	1	0
2839	0	1	0	1	1	1
2583	1	1	1	1	0	0
2315	1	0	1	0	1	0

5.4　对律师的回复水平进行评价

5.4.1　标准建立

在评价律师的回答问题的好坏时评价的标准仅从回答的长度来判断。我们选取这样的评价标准并没有具体考虑到回答的专业性，回答的详细程度，回答的严谨程度，我们

认为在这样的一个律师回答平台，每个律师回答的字数越多往往可以代表着，回答的比较专业，回答的详细，回答的比较严谨。另外我们不容易将这些专业性，严谨性，详细度这些指标无法在机器学习中得以很好的刻画。

5.4.2 算法分析

通过对每个律师回答的字数统计，按照由高到低的顺序排序，排序越靠前的则表明该律师的回复水平越高。

5.4.3 结果检验

对法律领域擅长方向的检验

首先每个律师回答问题均为 400 道，同时根据关键词筛选分类问答的总数小于 400 个，说明我们筛的关键词比较明确无歧义，一般不造成领域间的混淆，并且通过阅读文档实际找寻对律师的回复领域的检验发现律师确实对于该方面的回答很多。例如 2899 号律师的确主要解决民事问题。

我们认为只要排名在在前 1500 即可认为擅长这一领域，则每个律师平均擅长的领域有 3 个。

表 **4** 律师的擅长领域分析

擅长 0 个	擅长 1 个	擅长 2 个	擅长 3 个	擅长 4 个	擅长 5 个	擅长 6 个
60	325	672	887	684	300	72

擅长不同类别不同数量的人数呈现正态分布，说明我们用排名 1500 名作为分界线是较为合理的。

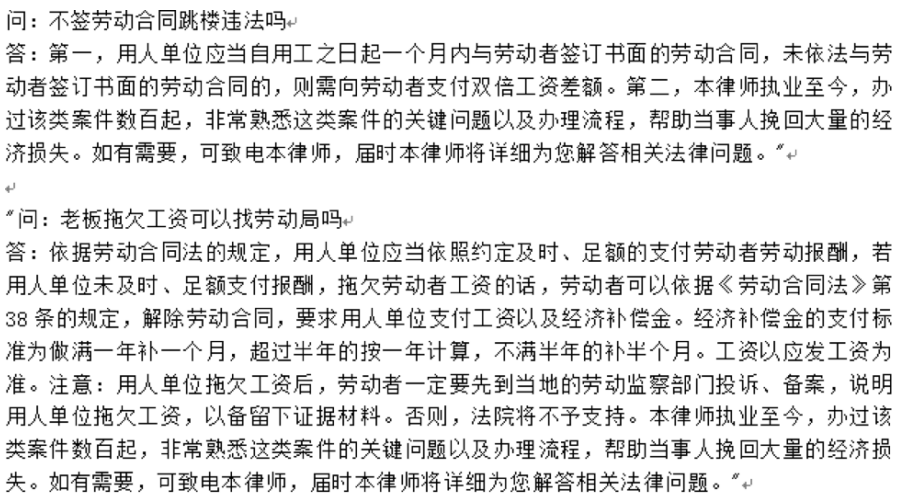
对律师的而回答质量和水平检验

通过排序我们发现平均回复长度前 5 名（括号内为平均回复字数）是 652 号律师 (516)，776 号律师 (774), 58 号律师 (294), 2114 号律师 (232), 436 号律师 (226) 。最后 5 名是 1300 号律师 (10), 1043 号律师 (9), 1218 号律师 (9), 2163 号律师 (8), 328 号律师 (7)。

注：通过我们实际查阅 652 号律师的问答，发现大部分的回答都是很长相同的话，故舍弃不用，选取排名第二的 776 号律师进行分析。

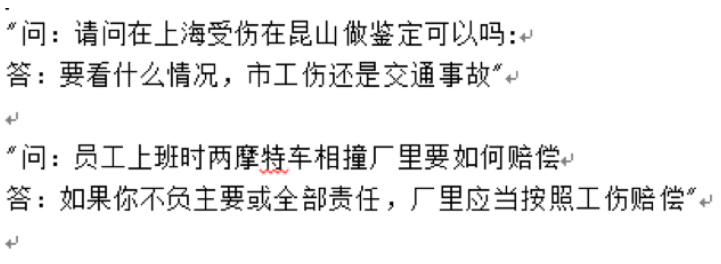
下面我们截取 776 号律师和 328 号律师的回答进行截取验证，验证如下：

由图 (5,6) 可见，776 号律师的回答要比 328 号律师详尽很多，且严谨很多，并且说话有依据，以及思考的方面也比 328 号全面的多


问：不签劳动合同跳楼违法吗
答：第一，用人单位应当自用工之日起一个月内与劳动者签订书面的劳动合同，未依法与劳动者签订书面的劳动合同的，则需向劳动者支付双倍工资差额。第二，本律师执业至今，办过该类案件数百起，非常熟悉这类案件的关键问题以及办理流程，帮助当事人挽回大量的经济损失。如有需要，可致电本律师，届时本律师将详细为您解答相关法律问题。"

"问：老板拖欠工资可以找劳动局吗
答：依据劳动合同法的规定，用人单位应当依照约定及时、足额的支付劳动者劳动报酬，若用人单位未及时、足额支付报酬，拖欠劳动者工资的话，劳动者可以依据《劳动合同法》第38条的规定，解除劳动合同，要求用人单位支付工资以及经济补偿金。经济补偿金的支付标准为做满一年补一个月，超过半年的按一年计算，不满半年的补半个月。工资以应发工资为准。注意：用人单位拖欠工资后，劳动者一定要先到当地的劳动监察部门投诉、备案，说明用人单位拖欠工资，以备留下证据材料。否则，法院将不予支持。本律师执业至今，办过该类案件数百起，非常熟悉这类案件的关键问题以及办理流程，帮助当事人挽回大量的经济损失。如有需要，可致电本律师，届时本律师将详细为您解答相关法律问题。"

图 **5**　**776** 号律师回答摘要


"问：请问在上海受伤在昆山做鉴定可以吗:
答：要看什么情况，市工伤还是交通事故"

"问：员工上班时两摩特车相撞厂里要如何赔偿
答：如果你不负主要或全部责任，厂里应当按照工伤赔偿"

图 **6**　**328** 号律师回答摘要

六、问题二模型建立与求解

在问题 1 中，我们已经有了两个指标: 第一个是通过分别将这些关键词与 3000 份律师的回复进行匹配，统计六个专业领域出现词频最高的前 50% 的回复文档，将它们对应的律师判定为擅长该专业领域。这个排序我们已知。第二个是我们已分别统计 3000 份回复中律师回答字数的平均值，排序我们也已知道。因而我们将这两个指标的平均作为我们推荐律师的判定综合指标，分数确定如下:

1. 3000 位律师中，六个专业领域出现词频最高的得 3000 分，第二名得 2999 分，依次递减。
2. 3000 位律师中，回答字数的平均值最高的得 3000 分，第二名得 2999 分，依次递减。
3. 3000 位律师在各个专业的指标为他在该专业出现词频得分与回答字数的平均值得分的平均。

通过这样一个综合指标，我们既考虑了律师回答某领域的问题数也考虑了他回答的水平。

综合指标排序结果如表 (5) 所示:（截取的各领域前 10）

指标确定后，我们按照以下的步骤处理:

1. 在客户输入的案情描述搜索我们在六个专业领域中挑选出的关键词，从而确定案情

表 **5**　各领域综合比指标排名

编号	民事民法	经济金融	刑事行政	涉外纠纷	公司企业	其他类别
2899	1	0	0	0	0	0
2583	1	1	1	1	0	0
2315	1	0	1	0	1	0
2839	0	1	0	1	1	1
2583	1	1	1	1	0	0
2315	1	0	1	0	1	0

属于的专业领域。

2. 挑选若干位在该领域总指标靠前的律师。
3. 手工查阅推荐编号的律师的问答记录，验证模型。

6.1　关键词提取

关键词提取与问题一类似，不同之处在于在对该客户输入案情描述的匹配过程中，可能出现大于一个领域的特征词语。因而可能推荐大于一个领域的律师。这也与实际生活是吻合的，有些案情可能涉及多个专业领域。

6.2　分配律师

关键词提取并匹配过后，我们就确定了这一案情涉及的领域。而我们也已经分别得到了六个领域律师综合指标的排序。根据用户需要的推荐律师的数目，只需按所需数量取出即可。

6.3　结果检验

综合指标的合理性检验

以民事民法综合指标排名第一的 1179 号律师为例，他在民事民法方面的回答均很完善。对于问题的回答不仅有针对性，而且基于事实，也基于法律。不管是分类说明还是阐述，他的回答均非常完整且清晰。如图 (7,8) 所示两个问答例子:

如图 (9) 经济金融领域综合指标排名第一的 1928 号律师，也条理清晰、有理有据。

相反, 查看涉外纠纷领域综合指标最后一名的 1043 号律师（如图 (10) 所示），得分 19。在其回答中仅出现“咨询”二字，没有任何有效信息。说明我们综合了回复题目数量

″问：我舅舅是地税局的领导人，很多应酬，每次喝完酒都会出点问题，上个星期喝完酒收了别人 4 万的红包，答应帮别人做事，现在很后悔，我想问一下受贿 4 万怎么判刑？
答：我国刑法第三百八十三条、第三百八十六条在规定贪污、受贿罪的定罪量刑情节时，适用了具体的数额表述：“行为人贪污、受贿满五千元的，构成贪污罪、受贿罪。”对犯贪污、受贿罪的量刑处罚明确规定为：“个人贪污数额在十万元以上的，处十年以上有期徒刑或者无期徒刑，可以并处没收财产；情节特别严重的，处死刑。″

图 **7**　**1179** 号律师回答摘要 **1**

″问：表弟在小的时候读书一直是班里的前几名，但不知道为什么，现在很喜欢跟社会上的人混在一起。总是出去惹事，就在上个月因为殴打他人、勒索他人被抓了。而且还是共同与他人犯罪，我想问下什么是共同犯罪啊？
答：共同犯罪分为一般共犯和特殊共犯即犯罪集团两种。一般共犯是指二人以上共同故意犯罪，而三人以上为共同实施犯罪而组成的较为固定的犯罪组织，是犯罪集团。组织、领导犯罪集团进行犯罪活动的，或者在共同犯罪中起主要作用的，是主犯。对组织、领导犯罪集团的首要分子，按照集团所犯的全部罪行处罚。在此之外的主犯，应当按照其所参加的或者组织、指挥的全部犯罪处罚。共同犯罪人主要分为主犯、从犯、胁从犯。″

图 **8**　**1179** 号律师回答摘要 **2**

″问：我哥哥在一个公司的财务部门工作，经常利用职务之便挪用一些钱出来炒股，累计大概有 20 多万元，请问挪用财务该怎么量刑呢？
答：挪用本单位资金归个人使用或者借贷给他人，数额较大、超过三个月未还的。这是较轻的一种挪用行为。其构成特征是行为人利用职务上主管、经手本单位资金的便利条件而挪用本单位资金，具用途主要是归个人使用或者借贷给他人使用，但未用于从事不正当的经济活动，而且挪用数额较大，且时间上超过三个月而未还。根据最高人民法院《关于办理违反公司法受贿、侵出、挪用等刑事案件适用法律若干问题的解释》的规定，挪用本单位资金一万元至三万元以上的，为“数额较大”。″

图 **9**　**1928** 号律师回答摘要

和回复字数的综合指标行之有效。

推荐律师检验

我们尝试向律师推荐程序键入问题，该问题中涉及到的特征词有房、拆，因而跟民事民法和经济金融有直接关联。程序运行后按照预期推荐了两大类共计 6 位律师，如图 (11) 所示。

例如检索 2583 号律师的文档发现，其针对这方面问题的回答较为优质，基本验证我们律师推荐算法的可行性。

″问：上个月我在上班途中发生交通事故住院费二万多全自出的，伤残鉴定为交通事故十级伤残，厂里有买社保，但他说不要走工伤那条路，因厂里也买了商业险走商业险这条路，到现在一分钱也没给我，请问住院费厂里该全出吗?能赔我多少钱。
答：咨询″
″问：你好，在交通事故中照成右手掌第五根骨折，事故责任一半，我该怎么理赔，谢谢
答：咨询″
″问：今天看医院的清单用的是进口钢板及固定大概费用将近 5 万元，保险是平安的，三者一百万及不计免都上了，保险公司说进口钢板只报 60%我想问下其余的是不是要我自己承担
答：对方起诉的话，很大可能保险全陪″
″问：我开朋友的车追尾了自己的车，对方保险公司拒赔我的车损，我怎么办？谢谢
答：咨询″
″问：委托律师去看守所看看我老公目前什么情况，需要支付律师费多少钱？
答：咨询″
″问：毕节是否可以鉴定交通伤残鉴定
答：咨询″
″问：请问事故中司机愿意负全责，交警非要行人负责，怎么办
答：咨询″
″问：交通事故后老板拖欠工资
答：咨询″
″问：我妈妈骑电动自行车冲红灯被公交车撞了重伤，责任有多少
答：咨询″
″问：大拇指粉碎性骨折能算十级伤残吗
答：咨询″

图 **10　1043** 号律师回答摘要

七、问题三模型建立与求解

7.1　方法与思路

方法 1

在客户输入的案情描述搜索我们在六个专业领域中挑选出的关键词，从而确定案情属于的专业领域。挑选若干位在该领域排名靠前的律师，在他们的回复中通过匹配关键词进行查找，选出最贴近的回答进行显示。

方法 2

根据客户输入的案情描述，通过爬虫登录搜索引擎，将访问量最高的回复返回给客户。

```
输入所需要推荐的律师数:
3
您的咨询内容是?（输入-1以结束咨询）
我家拆迁拿了一些补偿款，但是还是想要房，有什么途径可以换吗?
你的问题属于：民事类问题，向您推荐的3位律师如下所示:
第0位律师: 2899
第1位律师: 2583
第2位律师: 893
你的问题属于：经济类问题，向您推荐的3位律师如下所示:
第0位律师: 1142
第1位律师: 1287
第2位律师: 497
您的咨询内容是?（输入-1以结束咨询）
-1
感谢您的支持，祝您生活愉快
Program ended with exit code: 0
```

图 **11**　律师推荐程序实验

7.2　结果检验

本题中我们采用方法 1 进行验证。但由于我们是用匹配特征词语进行查找的，因而如果问题不带有特征词语的时候算法会失效。我们随机验证了两个情况，如图 (12) 所示。

```
您的咨询内容是?（输入-1以结束咨询）
我老公出轨了，我想离婚，那我们的财产怎么算呢
答：按照《婚姻法解释（二）》第11条的规定，婚姻关系存续期间，一方以个人财产投
资取得的收益为夫妻共同财产。
您的咨询内容是?（输入-1以结束咨询）
想买二手房，这个过户问题怎么处理
答：涉嫌非法集资
您的咨询内容是?（输入-1以结束咨询）
```

图 **12**　自动回答程序实验

第一个情况，非常的贴切，通过婚以及出轨都能定位出这是民事民法领域，再在推荐律师的文档中进行问题特征词语的匹配，最终得到答案，效果挺好。

但第二个情况，通过房定位到经济金融，但可能含有特征词语只有一个，因而得到了不理想的结果。暴露了该算法需要用户提出的问题中尽可能多地包含特征词语且过度依赖已有回答的弊端。

八、模型的优缺点

8.1 优点

1. 在选择律师擅长的方面时，用律师在某一领域回答问题的数量在 3000 个律师中的排名来做到，更具有客观性。
2. 在评价律师评价水平时选择用律师回答的长度来评价，思路简明，认为在一个律师专业咨询平台的回答，往往越长，即代表着回答的更加详细，更专业，更加严谨。这样便省去对多因素分配权重。
3. 在推荐律师时，考虑了出现不同专业领域关键词的情况，符合实际。现实中问题可能会涉及到大于一个专业领域。
4. 利用机器学习进行分词，再人工结合专业领域进行筛选，选出的词兼具概括性和高词频的特点。

8.2 缺点

1. 在找寻律师的擅长领域，按照我们的寻找标准，单个律师的回答特定领域问题数在所有 3000 个律师中的排名在前 1500 名认为擅长，有可能会存在一些律师没有擅长的领域存在，或者极个别律师擅长所有领域的情况。
2. 在找寻律师的擅长领域，我们用单个律师的回答特定领域问题数在所有 3000 个律师中的排名这一标准的一个前提假设是每个律师回答的总问题数是相同的，更精确可以将数量排名换成比例。
3. 在评价律师的回答问题的好坏时评价的标准仅从回答的长度来判断，没有证明回答的详细程度与专业性，严谨性有明显的正向相关关系。
4. 问题 3 中由于我们是用匹配特征词语进行查找的，因而如果问题所有特征词语一个都不带有的时候算法会失效。

九、模型改进方向

1. 在第二文中我们推选适合的律师的时候我们只是考虑律师在第一问中的的擅长程度，即，律师回答某专业的问题的多少在全体律师中的排名，我们会选取排名靠前的若干律师推送给客户，但是并没有具体考虑律师回答问题的详细程度，即没有考虑律师回答专业水平这一指标，因此我们在模型改进中应该综合上述的这两个指标。

2. 如果需要提高精度，就需要考虑本文 6 大法律领域的子法律，分类细化，找到更适合的律师，做到更高效率。

十、参考文献

[1] http://wiki.swarma.net/index.php/

[2] http://blog.sina.com.cn/s/blog_8af106960102v0v1.html

附录 A

程序源码

分词处理**Input Python source:**

```
import jieba
import nltk
from nltk.corpus import *
n = 3000;
dir1 = escapeinside 'escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDownloads
    escapeinside/escapeinsidedataescapeinside/escapeinside';
dirn = escapeinside 'escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDownloads
    escapeinside/escapeinsidedata0escapeinside/escapeinside';
while n > 0:
    str1 = str(n);
    dir2 = dir1 + str1;
    dir3 = dirn + str1;
    f = open(dir2)
    tmp_line = f.read()
    f.close()
    tmp_line_decode = tmp_line.decode(escapeinside 'escapeinsideutf escapeinside-8escapeinside')
    jieba_cut = jieba.cut(tmp_line_decode)
    ans = escapeinside 'escapeinside/escapeinside'.join(jieba_cut)
    ans = ans.encode(escapeinside 'escapeinsideutf escapeinside-8escapeinside')
    f2 = open(dir3, escapeinside 'escapeinsidewescapeinside')
    f2.write(ans)
    f2.close()
    n = n-1;
```

模仿 NLTK 进行关键词筛选与统计**Input C++ source:**

```
//
//  main.cpp
//  建模校赛
//
//  Created by tinoryj on 2017/5/4.
//  Copyright © 2017年 tinoryj. All rights reserved.
//
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <stdlib.h>
```

```
#include <string>
#include <fstream>
#include <cerrno>
#include <vector>
#include <cstring>
#include <time.h>
typedef long long ll;
using namespace std;

#define MAX 30 //关键字个数设置
#define type 6
#define outNum 1500
struct timeval start, end;

string get_file_contents(const char *filename){

    std::ifstream in(filename, std::ios::in | std::ios::binary);
    if (in){

        std::string contents;
        in.seekg(0, std::ios::end);
        contents.resize(in.tellg());
        in.seekg(0, std::ios::beg);
        in.read(&contents[0], contents.size());
        in.close();
        return(contents);
    }
    throw(errno);
}
struct node{

    int name;
    int times;
};

bool cmp(node a, node b){
    if(a.times > b.times){
        return true;
    }
    else{
        return false;
    }
}

int main(int argc, const char * argv[]) {

    int start = clock();
    string typeName[6];
    typeName[0] = escapeinside"escapeinside民escapeinside事escapeinside";
    typeName[1] = escapeinside"escapeinside经escapeinside济escapeinside";
    typeName[2] = escapeinside"escapeinside刑escapeinside事escapeinside";
    typeName[3] = escapeinside"escapeinside公escapeinside司escapeinside";
    typeName[4] = escapeinside"escapeinside其escapeinside他escapeinside";
    typeName[5] = escapeinside"escapeinside涉escapeinside外escapeinside";
    string keyWord[type][MAX];
    keyWord[0][0] = escapeinside"escapeinside婚escapeinside";
    keyWord[0][1] = escapeinside"escapeinside交escapeinside通escapeinside";
```

```
keyWord[0][2] = escapeinside"escapeinside车escapeinside";
keyWord[0][3] = escapeinside"escapeinside同escapeinside居escapeinside";
keyWord[0][4] = escapeinside"escapeinside出escapeinside轨escapeinside";
keyWord[0][5] = escapeinside"escapeinside疗escapeinside";
keyWord[0][6] = escapeinside"escapeinside假escapeinside";
keyWord[0][7] = escapeinside"escapeinside贷escapeinside";
keyWord[0][8] = escapeinside"escapeinside借escapeinside";
keyWord[0][9] = escapeinside"escapeinside拆escapeinside";

keyWord[1][0] = escapeinside"escapeinside房escapeinside";
keyWord[1][1] = escapeinside"escapeinside公escapeinside积escapeinside金escapeinside";
keyWord[1][2] = escapeinside"escapeinside首escapeinside付escapeinside";
keyWord[1][3] = escapeinside"escapeinside分escapeinside期escapeinside";
keyWord[1][4] = escapeinside"escapeinside产escapeinside权escapeinside";
keyWord[1][5] = escapeinside"escapeinside交escapeinside易escapeinside";
keyWord[1][6] = escapeinside"escapeinside基escapeinside金escapeinside";
keyWord[1][7] = escapeinside"escapeinside利escapeinside率escapeinside";
keyWord[1][8] = escapeinside"escapeinside发escapeinside票escapeinside";
keyWord[1][9] = escapeinside"escapeinside集escapeinside资escapeinside";

keyWord[2][0] = escapeinside"escapeinside刑escapeinside";
keyWord[2][1] = escapeinside"escapeinside政escapeinside";
keyWord[2][2] = escapeinside"escapeinside贿escapeinside";
keyWord[2][3] = escapeinside"escapeinside判escapeinside";
keyWord[2][4] = escapeinside"escapeinside狱escapeinside";
keyWord[2][5] = escapeinside"escapeinside公escapeinside安escapeinside";
keyWord[2][6] = escapeinside"escapeinside局escapeinside";
keyWord[2][7] = escapeinside"escapeinside委escapeinside";
keyWord[2][8] = escapeinside"escapeinside暴escapeinside";
keyWord[2][9] = escapeinside"escapeinside犯escapeinside";

keyWord[3][0] = escapeinside"escapeinside公escapeinside司escapeinside";
keyWord[3][1] = escapeinside"escapeinside企escapeinside";
keyWord[3][2] = escapeinside"escapeinside股escapeinside";
keyWord[3][3] = escapeinside"escapeinside收escapeinside购escapeinside";
keyWord[3][4] = escapeinside"escapeinside税escapeinside";
keyWord[3][5] = escapeinside"escapeinside万escapeinside元escapeinside";
keyWord[3][6] = escapeinside"escapeinside资escapeinside产escapeinside";
keyWord[3][7] = escapeinside"escapeinside份escapeinside额escapeinside";
keyWord[3][8] = escapeinside"escapeinside执escapeinside照escapeinside";
keyWord[3][9] = escapeinside"escapeinside裁escapeinside员escapeinside";

keyWord[4][0] = escapeinside"escapeinside函escapeinside";
keyWord[4][1] = escapeinside"escapeinside文escapeinside书escapeinside";
keyWord[4][2] = escapeinside"escapeinside委escapeinside托escapeinside";
keyWord[4][3] = escapeinside"escapeinside短escapeinside信escapeinside";
keyWord[4][4] = escapeinside"escapeinside移escapeinside动escapeinside";
keyWord[4][5] = escapeinside"escapeinside联escapeinside动escapeinside";
keyWord[4][6] = escapeinside"escapeinside电escapeinside信escapeinside";
keyWord[4][7] = escapeinside"escapeinside广escapeinside告escapeinside";
keyWord[4][8] = escapeinside"escapeinside环escapeinside境escapeinside";
keyWord[4][9] = escapeinside"escapeinside代escapeinside理escapeinside";

keyWord[5][0] = escapeinside"escapeinside仲escapeinside裁escapeinside";
keyWord[5][1] = escapeinside"escapeinside台escapeinside资escapeinside";
keyWord[5][2] = escapeinside"escapeinside贸escapeinside易escapeinside";
keyWord[5][3] = escapeinside"escapeinside加escapeinside拿escapeinside大escapeinside";
```

```
keyWord[5][4] = escapeinside"escapeinside美escapeinside国escapeinside";
keyWord[5][5] = escapeinside"escapeinside大escapeinside使escapeinside";
keyWord[5][6] = escapeinside"escapeinside国escapeinside外escapeinside";
keyWord[5][7] = escapeinside"escapeinside合escapeinside资escapeinside";
keyWord[5][8] = escapeinside"escapeinside外escapeinside资escapeinside";
keyWord[5][9] = escapeinside"escapeinside海escapeinside";


char fileName[10];
string dir = escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/
    escapeinsideDownloadsescapeinside/escapeinsidedataescapeinside/escapeinside";
int fileCount = 3000;
int wordCount[fileCount][type];
memset(wordCount, 0, sizeof(wordCount));
for(int n = 1; n <= fileCount; n++){
    int countQ = 0;
    int cnt = 0;
    sprintf(fileName, escapeinside"escapeinside%escapeinsidedescapeinside", n);
    string file = dir + fileName;
    string dataRead = get_file_contents(file.c_str()); //文件绝对路径
    cout<<escapeinside"escapeinside文escapeinside件escapeinside名escapeinside:escapeinside escapeinside"<<fileName<<
        escapeinside"escapeinside escapeinside escapeinside总escapeinside字escapeinside符escapeinside数escapeinside:
        escapeinside escapeinside"<<dataRead.length()<<endl;
    string temp;
    for(int i = 0; i < dataRead.length(); i++){

        if(dataRead[i] == escapeinside'escapeinside\"escapeinside' && (dataRead[i+1] == escapeinside'escapeinside\
            escapeinsiderescapeinside' || dataRead[i+1] == escapeinside'escapeinside\0escapeinside')){

            for(int j = 0; j < type; j++){

                if(strstr(temp.c_str(),keyWord[j][0].c_str()) || strstr(temp.c_str(),keyWord[j
                    ][1].c_str()) || strstr(temp.c_str(),keyWord[j][2].c_str()) || strstr(temp.
                    c_str(),keyWord[j][3].c_str()) || strstr(temp.c_str(),keyWord[j][4].c_str())
                     || strstr(temp.c_str(),keyWord[j][5].c_str()) || strstr(temp.c_str(),
                    keyWord[j][6].c_str()) || strstr(temp.c_str(),keyWord[j][7].c_str()) ||
                    strstr(temp.c_str(),keyWord[j][8].c_str()) || strstr(temp.c_str(),keyWord[j
                    ][9].c_str())){

                    wordCount[n-1][j]++;
                    cnt++;
                }
            }
            //cout<<temp<<endl;
            temp.clear();
            countQ++;
        }
        temp.push_back(dataRead[i]);
    }
    /*
    cout<<"问答数: "<<countQ<<endl;
    cout<<cnt<<endl;
    for(int i = 0; i < type; i++){

        cout<<i<<":"<<wordCount[n-1][i]<<endl; //输出含有关键字的问答的数目
    }
     */
```

```
}

vector<node> type0;
for(int i = 0; i < fileCount; i++){

    node temp;
    temp.name = i;
    temp.times = wordCount[i][0];
    type0.push_back(temp);
}
sort(type0.begin(), type0.end(), cmp);
cout<<typeName[0]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
    escapeinside"escapeinside escapeinside频escapeinside数escapeinside"<<endl;
for(int i = 0; i < outNum; i++){
    cout<<type0[i].name<<escapeinside"escapeinside escapeinside"<<type0[i].times<<endl;
}

vector<node> type1;
for(int i = 0; i < fileCount; i++){

    node temp;
    temp.name = i;
    temp.times = wordCount[i][1];
    type1.push_back(temp);
}
sort(type1.begin(), type1.end(), cmp);
cout<<typeName[1]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
    escapeinside"escapeinside escapeinside频escapeinside数escapeinside"<<endl;
for(int i = 0; i < outNum; i++){
    cout<<type1[i].name<<escapeinside"escapeinside escapeinside"<<type1[i].times<<endl;
}

vector<node> type2;
for(int i = 0; i < fileCount; i++){

    node temp;
    temp.name = i;
    temp.times = wordCount[i][2];
    type2.push_back(temp);
}
sort(type2.begin(), type2.end(), cmp);
cout<<typeName[2]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
    escapeinside"escapeinside escapeinside频escapeinside数escapeinside"<<endl;
for(int i = 0; i < outNum; i++){
    cout<<type2[i].name<<escapeinside"escapeinside escapeinside"<<type2[i].times<<endl;
}
vector<node> type3;
for(int i = 0; i < fileCount; i++){

    node temp;
    temp.name = i;
    temp.times = wordCount[i][3];
    type3.push_back(temp);
}
sort(type3.begin(), type3.end(), cmp);
cout<<typeName[3]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
    escapeinside"escapeinside escapeinside频escapeinside数escapeinside"<<endl;
```

```
    for(int i = 0; i < outNum; i++){
        cout<<type3[i].name<<escapeinside"escapeinside escapeinside"<<type3[i].times<<endl;
    }
    vector<node> type4;
    for(int i = 0; i < fileCount; i++){

        node temp;
        temp.name = i;
        temp.times = wordCount[i][4];
        type4.push_back(temp);
    }
    sort(type4.begin(), type4.end(), cmp);
    cout<<typeName[4]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
        escapeinside"escapeinside escapeinside频escapeinside数escapeinside"<<endl;
    for(int i = 0; i < outNum; i++){
        cout<<type4[i].name<<escapeinside"escapeinside escapeinside"<<type4[i].times<<endl;
    }

    vector<node> type5;
    for(int i = 0; i < fileCount; i++){

        node temp;
        temp.name = i;
        temp.times = wordCount[i][5];
        type5.push_back(temp);
    }
    sort(type5.begin(), type5.end(), cmp);
    cout<<typeName[5]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
        escapeinside"escapeinside escapeinside频escapeinside数escapeinside"<<endl;
    for(int i = 0; i < outNum; i++){
        cout<<type5[i].name<<escapeinside"escapeinside escapeinside"<<type5[i].times<<endl;
    }
    int end = clock();
    cout<<(end - start)/1000000;
    return 0;
}
```

第一问：按照回答长度划分律师回答水平**Input C++ source:**

```
//
//  main.cpp
//  20170506-3
//
//  Created by tinoryj on 2017/5/6.
//  Copyright © 2017年 tinoryj. All rights reserved.
//
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <stdlib.h>
#include <string>
#include <fstream>
#include <cerrno>
#include <vector>
#include <cstring>
#include <time.h>
typedef long long ll;
```

```
using namespace std;

string get_file_contents(const char *filename){

    std::ifstream in(filename, std::ios::in | std::ios::binary);
    if (in){

        std::string contents;
        in.seekg(0, std::ios::end);
        contents.resize(in.tellg());
        in.seekg(0, std::ios::beg);
        in.read(&contents[0], contents.size());
        in.close();
        return(contents);
    }
    throw(errno);
}
struct node{
    int name = 0;
    int sum = 0;
    int avr = 0;
};
bool cmp(node a, node b){

    if(a.avr > b.avr){
        return true;
    }
    else{
        return false;
    }
}
int main(int argc, const char * argv[]) {

    char fileName[10];

    freopen(escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDesktop
        escapeinside/escapeinsidezhuanyeescapeinside.escapeinsidetxtescapeinside", escapeinside"escapeinsidewescapeinside+
        escapeinside", stdout);

    string dir = escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/
        escapeinsideDownloadsescapeinside/escapeinsidedataescapeinside/escapeinside";
    int fileCount = 3000; //参与统计的文件数目
    vector<node> la;
    cout<<escapeinside"escapeinside律escapeinside师escapeinside编escapeinside号escapeinside escapeinside escapeinside"<<
        escapeinside"escapeinside回escapeinside答escapeinside总escapeinside长escapeinside度escapeinside escapeinside escapeinside"
        <<escapeinside"escapeinside平escapeinside均escapeinside回escapeinside答escapeinside长escapeinside度escapeinside"<<endl;
    for(int n = 1; n <= fileCount; n++){
        sprintf(fileName, escapeinside"escapeinside%escapeinsidedescapeinside", n);
        string file = dir + fileName;
        string dataRead = get_file_contents(file.c_str()); //文件绝对路径
        string temp;
        string len;
        node tmp;
        tmp.name = n;
        for(int i = 0; i < dataRead.length(); i++){

            if(dataRead[i] == escapeinside'escapeinside\"escapeinside' && (dataRead[i+1] == escapeinside'escapeinside\
```

```
                escapeinsider escapeinside' || dataRead[i+1] == escapeinside 'escapeinside\0escapeinside')){


                if(!strstr(temp.c_str(),escapeinside"escapeinside答escapeinside")){
                    continue;
                }
                len = strstr(temp.c_str(),escapeinside"escapeinside答escapeinside");

                tmp.sum += len.length();
                temp.clear();
                len.clear();
            }
            temp.push_back(dataRead[i]);
        }
        tmp.avr = tmp.sum/400;
        la.push_back(tmp);
    }
    sort(la.begin(), la.end(), cmp);
    for(int i = 0; i < la.size(); i++){
        cout<<la[i].name<<escapeinside"escapeinside escapeinside escapeinside"<<la[i].sum<<escapeinside"escapeinside
            escapeinside escapeinside"<<la[i].avr<<endl;
    }
    return 0;
}
```

第二问:综合领域分类关键词匹配度排序以及回答长度的综合指标处理Input C++ source:

```
//
//  main.cpp
//  jiaquanpaiming
//
//  Created by tinoryj on 06/05/2017.
//  Copyright © 2017 tinoryj. All rights reserved.
//
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <stdlib.h>
#include <string>
#include <fstream>
#include <cerrno>
#include <vector>
#include <cstring>
#include <time.h>
typedef long long ll;
using namespace std;

#define MAX 30 //关键字个数设置
#define type 6
#define outNum 3000
struct timeval start, end;

string getFileContents(const char *filename){

    ifstream in(filename, ios::in | ios::binary);
    if (in){
```

```
        std::string contents;
        in.seekg(0, ios::end);
        contents.resize(in.tellg());
        in.seekg(0, ios::beg);
        in.read(&contents[0], contents.size());
        in.close();
        return(contents);
    }
    throw(errno);
}
struct node{

    int name;
    int code;
};

bool cmp(node a, node b){
    if(a.code > b.code){
        return true;
    }
    else{
        return false;
    }
}

int main(int argc, const char * argv[]) {
    string typeName[6];
    typeName[0] = escapeinside"escapeinside民escapeinside事escapeinside";
    typeName[1] = escapeinside"escapeinside经escapeinside济escapeinside";
    typeName[2] = escapeinside"escapeinside刑escapeinside事escapeinside";
    typeName[3] = escapeinside"escapeinside公escapeinside司escapeinside";
    typeName[4] = escapeinside"escapeinside其escapeinside他escapeinside";
    typeName[5] = escapeinside"escapeinside涉escapeinside外escapeinside";

    freopen(escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDesktop
        escapeinside/escapeinsidepaiminescapeinside", escapeinside"escapeinsidewescapeinside+escapeinside", stdout);

    ifstream in1(escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDesktop
        escapeinside/escapeinsidezhuanyefenshuescapeinside");
    ifstream in2(escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDesktop
        escapeinside/escapeinsidezhuanyeescapeinside.escapeinsidetxtescapeinside");


    node zhuanye[3000];
    node leibie[6][3000];
    for(int i = 0; i < 6; i++){
        for(int j = 0; j < 3000; j++){
            in1>>leibie[i][j].name>>leibie[i][j].code;
        }
    }
    for(int i = 0; i < 3000; i++){
        int temp1, temp2;
        in2>>zhuanye[i].name>>temp1>>temp2;
        zhuanye[i].code = 3000-i;
    }
    vector<node> type0;
    for(int i = 0; i < 3000; i++){
```

```
    for(int j = 0; j < 3000; j++){
        if(zhuanye[i].name == leibie[0][j].name){

            node tmp;
            tmp.name = zhuanye[i].name;
            tmp.code = zhuanye[i].code + leibie[0][j].code;
            type0.push_back(tmp);
        }
    }
}
sort(type0.begin(), type0.end(), cmp);
cout<<typeName[0]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
    escapeinside"escapeinside escapeinsidecodeescapeinside"<<endl;
for(int i = 0; i < outNum; i++){
    cout<<type0[i].name<<escapeinside"escapeinside escapeinside"<<type0[i].code/2<<endl;
}
cout<<endl<<endl<<endl;
vector<node> type1;
for(int i = 0; i < 3000; i++){
    for(int j = 0; j < 3000; j++){
        if(zhuanye[i].name == leibie[1][j].name){

            node tmp;
            tmp.name = zhuanye[i].name;
            tmp.code = zhuanye[i].code + leibie[1][j].code;
            type1.push_back(tmp);
        }
    }
}
sort(type1.begin(), type1.end(), cmp);
cout<<typeName[1]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
    escapeinside"escapeinside escapeinsidecodeescapeinside"<<endl;
for(int i = 0; i < outNum; i++){
    cout<<type1[i].name<<escapeinside"escapeinside escapeinside"<<type1[i].code/2<<endl;
}
cout<<endl<<endl<<endl;
vector<node> type2;
for(int i = 0; i < 3000; i++){
    for(int j = 0; j < 3000; j++){
        if(zhuanye[i].name == leibie[2][j].name){

            node tmp;
            tmp.name = zhuanye[i].name;
            tmp.code = zhuanye[i].code + leibie[2][j].code;
            type2.push_back(tmp);
        }
    }
}
sort(type2.begin(), type2.end(), cmp);
cout<<typeName[2]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
    escapeinside"escapeinside escapeinsidecodeescapeinside"<<endl;
for(int i = 0; i < outNum; i++){
    cout<<type2[i].name<<escapeinside"escapeinside escapeinside"<<type2[i].code/2<<endl;
}
cout<<endl<<endl<<endl;
vector<node> type3;
for(int i = 0; i < 3000; i++){
```

```
        for(int j = 0; j < 3000; j++){
            if(zhuanye[i].name == leibie[3][j].name){

                node tmp;
                tmp.name = zhuanye[i].name;
                tmp.code = zhuanye[i].code + leibie[3][j].code;
                type3.push_back(tmp);
            }
        }
    }
    sort(type3.begin(), type3.end(), cmp);
    cout<<typeName[3]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
        escapeinside"escapeinside escapeinsidecodeescapeinside"<<endl;
    for(int i = 0; i < outNum; i++){
        cout<<type3[i].name<<escapeinside"escapeinside escapeinside"<<type3[i].code/2<<endl;
    }
    cout<<endl<<endl<<endl;
    vector<node> type4;
    for(int i = 0; i < 3000; i++){
        for(int j = 0; j < 3000; j++){
            if(zhuanye[i].name == leibie[4][j].name){

                node tmp;
                tmp.name = zhuanye[i].name;
                tmp.code = zhuanye[i].code + leibie[4][j].code;
                type4.push_back(tmp);
            }
        }
    }
    sort(type4.begin(), type4.end(), cmp);
    cout<<typeName[4]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
        escapeinside"escapeinside escapeinsidecodeescapeinside"<<endl;
    for(int i = 0; i < outNum; i++){
        cout<<type4[i].name<<escapeinside"escapeinside escapeinside"<<type4[i].code/2<<endl;
    }
    cout<<endl<<endl<<endl;
    vector<node> type5;
    for(int i = 0; i < 3000; i++){
        for(int j = 0; j < 3000; j++){
            if(zhuanye[i].name == leibie[5][j].name){

                node tmp;
                tmp.name = zhuanye[i].name;
                tmp.code = zhuanye[i].code + leibie[5][j].code;
                type5.push_back(tmp);
            }
        }
    }
    sort(type5.begin(), type5.end(), cmp);
    cout<<typeName[5]<<escapeinside"escapeinside:escapeinside"<<escapeinside"escapeinside编escapeinside号escapeinside"<<
        escapeinside"escapeinside escapeinsidecodeescapeinside"<<endl;
    for(int i = 0; i < outNum; i++){
        cout<<type5[i].name<<escapeinside"escapeinside escapeinside"<<type5[i].code/2<<endl;
    }
    return 0;
}
```

第二问：律师选择推荐程序**Input C++ source:**

```
//
//  main.cpp
//  20170505-2
//
//  Created by tinoryj on 2017/5/5.
//  Copyright © 2017年 tinoryj. All rights reserved.
//

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <stdlib.h>
#include <string>
#include <fstream>
#include <cerrno>
#include <vector>
#include <cstring>
#include <time.h>
typedef long long ll;
using namespace std;

#define MAX 30 //关键字个数设置
#define type 6
#define outNum 1500

int main(int argc, const char * argv[]) {

    ifstream in(escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDesktop
        escapeinside/escapeinsidedataescapeinside");//使用时请添加数据集data路径
    string typeName[6];
    typeName[0] = escapeinside"escapeinside民escapeinside事escapeinside";
    typeName[1] = escapeinside"escapeinside经escapeinside济escapeinside";
    typeName[2] = escapeinside"escapeinside刑escapeinside事escapeinside";
    typeName[3] = escapeinside"escapeinside公escapeinside司escapeinside";
    typeName[4] = escapeinside"escapeinside其escapeinside他escapeinside";
    typeName[5] = escapeinside"escapeinside涉escapeinside外escapeinside";


    string keyWord[type][MAX];
    keyWord[0][0] = escapeinside"escapeinside婚escapeinside";
    keyWord[0][1] = escapeinside"escapeinside交escapeinside通escapeinside";
    keyWord[0][2] = escapeinside"escapeinside车escapeinside";
    keyWord[0][3] = escapeinside"escapeinside同escapeinside居escapeinside";
    keyWord[0][4] = escapeinside"escapeinside出escapeinside轨escapeinside";
    keyWord[0][5] = escapeinside"escapeinside疗escapeinside";
    keyWord[0][6] = escapeinside"escapeinside假escapeinside";
    keyWord[0][7] = escapeinside"escapeinside贷escapeinside";
    keyWord[0][8] = escapeinside"escapeinside借escapeinside";
    keyWord[0][9] = escapeinside"escapeinside拆escapeinside";

    keyWord[1][0] = escapeinside"escapeinside房escapeinside";
    keyWord[1][1] = escapeinside"escapeinside公escapeinside积escapeinside金escapeinside";
    keyWord[1][2] = escapeinside"escapeinside首escapeinside付escapeinside";
    keyWord[1][3] = escapeinside"escapeinside分escapeinside期escapeinside";
    keyWord[1][4] = escapeinside"escapeinside产escapeinside权escapeinside";
    keyWord[1][5] = escapeinside"escapeinside交escapeinside易escapeinside";
```

```
keyWord[1][6] = escapeinside"escapeinside基escapeinside金escapeinside";
keyWord[1][7] = escapeinside"escapeinside利escapeinside率escapeinside";
keyWord[1][8] = escapeinside"escapeinside发escapeinside票escapeinside";
keyWord[1][9] = escapeinside"escapeinside集escapeinside资escapeinside";

keyWord[2][0] = escapeinside"escapeinside刑escapeinside";
keyWord[2][1] = escapeinside"escapeinside政escapeinside";
keyWord[2][2] = escapeinside"escapeinside贿escapeinside";
keyWord[2][3] = escapeinside"escapeinside判escapeinside";
keyWord[2][4] = escapeinside"escapeinside狱escapeinside";
keyWord[2][5] = escapeinside"escapeinside公escapeinside安escapeinside";
keyWord[2][6] = escapeinside"escapeinside局escapeinside";
keyWord[2][7] = escapeinside"escapeinside委escapeinside";
keyWord[2][8] = escapeinside"escapeinside暴escapeinside";
keyWord[2][9] = escapeinside"escapeinside犯escapeinside";

keyWord[3][0] = escapeinside"escapeinside公escapeinside司escapeinside";
keyWord[3][1] = escapeinside"escapeinside企escapeinside";
keyWord[3][2] = escapeinside"escapeinside股escapeinside";
keyWord[3][3] = escapeinside"escapeinside收escapeinside购escapeinside";
keyWord[3][4] = escapeinside"escapeinside税escapeinside";
keyWord[3][5] = escapeinside"escapeinside万escapeinside元escapeinside";
keyWord[3][6] = escapeinside"escapeinside资escapeinside产escapeinside";
keyWord[3][7] = escapeinside"escapeinside份escapeinside额escapeinside";
keyWord[3][8] = escapeinside"escapeinside执escapeinside照escapeinside";
keyWord[3][9] = escapeinside"escapeinside裁escapeinside员escapeinside";

keyWord[4][0] = escapeinside"escapeinside函escapeinside";
keyWord[4][1] = escapeinside"escapeinside文escapeinside书escapeinside";
keyWord[4][2] = escapeinside"escapeinside委escapeinside托escapeinside";
keyWord[4][3] = escapeinside"escapeinside短escapeinside信escapeinside";
keyWord[4][4] = escapeinside"escapeinside移escapeinside动escapeinside";
keyWord[4][5] = escapeinside"escapeinside联escapeinside动escapeinside";
keyWord[4][6] = escapeinside"escapeinside电escapeinside信escapeinside";
keyWord[4][7] = escapeinside"escapeinside广escapeinside告escapeinside";
keyWord[4][8] = escapeinside"escapeinside环escapeinside境escapeinside";
keyWord[4][9] = escapeinside"escapeinside代escapeinside理escapeinside";

keyWord[5][0] = escapeinside"escapeinside仲escapeinside裁escapeinside";
keyWord[5][1] = escapeinside"escapeinside台escapeinside资escapeinside";
keyWord[5][2] = escapeinside"escapeinside贸escapeinside易escapeinside";
keyWord[5][3] = escapeinside"escapeinside加escapeinside拿escapeinside大escapeinside";
keyWord[5][4] = escapeinside"escapeinside美escapeinside国escapeinside";
keyWord[5][5] = escapeinside"escapeinside大escapeinside使escapeinside";
keyWord[5][6] = escapeinside"escapeinside国escapeinside外escapeinside";
keyWord[5][7] = escapeinside"escapeinside合escapeinside资escapeinside";
keyWord[5][8] = escapeinside"escapeinside外escapeinside资escapeinside";
keyWord[5][9] = escapeinside"escapeinside海escapeinside";

int data[6][1500][2];
for(int i = 0; i < type; i++){

    for(int j = 0; j < 1500; j++){

        in>>data[i][j][0]>>data[i][j][1];
    }
}
```

```
    cout<<"escapeinside输escapeinside入escapeinside所escapeinside需escapeinside要escapeinside推escapeinside荐
        escapeinside的escapeinside律escapeinside师escapeinside数： escapeinside"<<endl;
    int num;
    cin>>num;
    if(num < 1){

        num = 1;
    }

    while(1){
        cout<<"escapeinside您escapeinside的escapeinside咨escapeinside询escapeinside内escapeinside容escapeinside是？
            escapeinside(escapeinside输escapeinside入escapeinside—1escapeinside以escapeinside结escapeinside束escapeinside咨
            escapeinside询escapeinside)escapeinside"<<endl;
        string temp;
        cin>>temp;
        if(strstr(temp.c_str(),"escapeinside—1escapeinside")){
            break;
        }
        bool flag = false;
        for(int j = 0; j < type; j++){

            if(strstr(temp.c_str(),keyWord[j][0].c_str()) || strstr(temp.c_str(),keyWord[j][1].c_str
                ()) || strstr(temp.c_str(),keyWord[j][2].c_str()) || strstr(temp.c_str(),keyWord[j
                ][3].c_str()) || strstr(temp.c_str(),keyWord[j][4].c_str()) || strstr(temp.c_str(),
                keyWord[j][5].c_str()) || strstr(temp.c_str(),keyWord[j][6].c_str()) || strstr(temp.
                c_str(),keyWord[j][7].c_str()) || strstr(temp.c_str(),keyWord[j][8].c_str()) ||
                strstr(temp.c_str(),keyWord[j][9].c_str())){
                flag = true;
                cout<<"escapeinside你escapeinside的escapeinside问escapeinside题escapeinside属escapeinside于：
                    escapeinside"<<typeName[j]<<"escapeinside类escapeinside问escapeinside题， escapeinside向
                    escapeinside您escapeinside推escapeinside荐escapeinside的escapeinside"<<num<<"escapeinside位
                    escapeinside律escapeinside师escapeinside如escapeinside下escapeinside所escapeinside示： escapeinside"<<
                    endl;
                for(int i = 0; i < num; i++){

                    cout<<"escapeinside第escapeinside"<<i<<"escapeinside位escapeinside律
                        escapeinside师： escapeinside"<<data[j][i][0]<<endl;
                }
            }
        }
        if(!flag){
            cout<<"escapeinside您escapeinside咨escapeinside询escapeinside的escapeinside问escapeinside题
                escapeinside伦escapeinside家escapeinside不escapeinside会escapeinside捏escapeinside(escapeinside￣escapeinside□
                escapeinside￣escapeinside)escapeinside， escapeinside对escapeinside不escapeinside起escapeinside"<<endl;
        }
    }
    cout<<"escapeinside感escapeinside谢escapeinside您escapeinside的escapeinside支escapeinside持， escapeinside祝
        escapeinside您escapeinside生escapeinside活escapeinside愉escapeinside快escapeinside"<<endl;

    return 0;
}
```

第三问：自动回答系统的尝试**Input C++ source:**

```
//
//  main.cpp
//  20170506-4
```

```
//
//  Created by tinoryj on 2017/5/6.
//  Copyright © 2017年 tinoryj. All rights reserved.
//

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <stdlib.h>
#include <string>
#include <fstream>
#include <cerrno>
#include <vector>
#include <cstring>
#include <time.h>
typedef long long ll;
using namespace std;

#define MAX 30 //关键字个数设置
#define type 6
#define outNum 1500
string getFileContents(const char *filename){

    ifstream in(filename, ios::in | ios::binary);
    if (in){

        std::string contents;
        in.seekg(0, ios::end);
        contents.resize(in.tellg());
        in.seekg(0, ios::beg);
        in.read(&contents[0], contents.size());
        in.close();
        return(contents);
    }
    throw(errno);
}
int main(int argc, const char * argv[]) {

    ifstream in(escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/escapeinsideDesktop
        escapeinside/escapeinsidedataescapeinside"); //使用时请添加数据集data路径
    string typeName[6];
    typeName[0] = escapeinside"escapeinside民escapeinside事escapeinside";
    typeName[1] = escapeinside"escapeinside经escapeinside济escapeinside";
    typeName[2] = escapeinside"escapeinside刑escapeinside事escapeinside";
    typeName[3] = escapeinside"escapeinside公escapeinside司escapeinside";
    typeName[4] = escapeinside"escapeinside其escapeinside他escapeinside";
    typeName[5] = escapeinside"escapeinside涉escapeinside外escapeinside";


    string keyWord[type][MAX];
    keyWord[0][0] = escapeinside"escapeinside婚escapeinside";
    keyWord[0][1] = escapeinside"escapeinside交escapeinside通escapeinside";
    keyWord[0][2] = escapeinside"escapeinside车escapeinside";
    keyWord[0][3] = escapeinside"escapeinside同escapeinside居escapeinside";
    keyWord[0][4] = escapeinside"escapeinside出escapeinside轨escapeinside";
    keyWord[0][5] = escapeinside"escapeinside疗escapeinside";
    keyWord[0][6] = escapeinside"escapeinside假escapeinside";
```

```
keyWord[0][7] = "escapeinside贷escapeinside";
keyWord[0][8] = "escapeinside借escapeinside";
keyWord[0][9] = "escapeinside拆escapeinside";

keyWord[1][0] = "escapeinside房escapeinside";
keyWord[1][1] = "escapeinside公escapeinside积escapeinside金escapeinside";
keyWord[1][2] = "escapeinside首escapeinside付escapeinside";
keyWord[1][3] = "escapeinside分escapeinside期escapeinside";
keyWord[1][4] = "escapeinside产escapeinside权escapeinside";
keyWord[1][5] = "escapeinside交escapeinside易escapeinside";
keyWord[1][6] = "escapeinside基escapeinside金escapeinside";
keyWord[1][7] = "escapeinside利escapeinside率escapeinside";
keyWord[1][8] = "escapeinside发escapeinside票escapeinside";
keyWord[1][9] = "escapeinside集escapeinside资escapeinside";

keyWord[2][0] = "escapeinside刑escapeinside";
keyWord[2][1] = "escapeinside政escapeinside";
keyWord[2][2] = "escapeinside贿escapeinside";
keyWord[2][3] = "escapeinside判escapeinside";
keyWord[2][4] = "escapeinside狱escapeinside";
keyWord[2][5] = "escapeinside公escapeinside安escapeinside";
keyWord[2][6] = "escapeinside局escapeinside";
keyWord[2][7] = "escapeinside委escapeinside";
keyWord[2][8] = "escapeinside暴escapeinside";
keyWord[2][9] = "escapeinside犯escapeinside";

keyWord[3][0] = "escapeinside公escapeinside司escapeinside";
keyWord[3][1] = "escapeinside企escapeinside";
keyWord[3][2] = "escapeinside股escapeinside";
keyWord[3][3] = "escapeinside收escapeinside购escapeinside";
keyWord[3][4] = "escapeinside税escapeinside";
keyWord[3][5] = "escapeinside万escapeinside元escapeinside";
keyWord[3][6] = "escapeinside资escapeinside产escapeinside";
keyWord[3][7] = "escapeinside份escapeinside额escapeinside";
keyWord[3][8] = "escapeinside执escapeinside照escapeinside";
keyWord[3][9] = "escapeinside裁escapeinside员escapeinside";

keyWord[4][0] = "escapeinside函escapeinside";
keyWord[4][1] = "escapeinside文escapeinside书escapeinside";
keyWord[4][2] = "escapeinside委escapeinside托escapeinside";
keyWord[4][3] = "escapeinside短escapeinside信escapeinside";
keyWord[4][4] = "escapeinside移escapeinside动escapeinside";
keyWord[4][5] = "escapeinside联escapeinside动escapeinside";
keyWord[4][6] = "escapeinside电escapeinside信escapeinside";
keyWord[4][7] = "escapeinside广escapeinside告escapeinside";
keyWord[4][8] = "escapeinside环escapeinside境escapeinside";
keyWord[4][9] = "escapeinside代escapeinside理escapeinside";

keyWord[5][0] = "escapeinside仲escapeinside裁escapeinside";
keyWord[5][1] = "escapeinside台escapeinside资escapeinside";
keyWord[5][2] = "escapeinside贸escapeinside易escapeinside";
keyWord[5][3] = "escapeinside加escapeinside拿escapeinside大escapeinside";
keyWord[5][4] = "escapeinside美escapeinside国escapeinside";
keyWord[5][5] = "escapeinside大escapeinside使escapeinside";
keyWord[5][6] = "escapeinside国escapeinside外escapeinside";
keyWord[5][7] = "escapeinside合escapeinside资escapeinside";
keyWord[5][8] = "escapeinside外escapeinside资escapeinside";
```

```
keyWord[5][9] = escapeinside"escapeinside海escapeinside";

int data[6];
data[0] = 2899;
data[1] = 1142;
data[2] = 2315;
data[3] = 2583;
data[4] = 998;
data[5] = 2839;


string dir = escapeinside"escapeinside/escapeinsideUsersescapeinside/escapeinsidetinoryjescapeinside/
    escapeinsideDownloadsescapeinside/escapeinsidedataescapeinside/escapeinside";
char fileName[10];
while(1){
    cout<<escapeinside"escapeinside您escapeinside的escapeinside客escapeinside询escapeinside内escapeinside容escapeinside是？
        escapeinside(escapeinside输escapeinside入escapeinside-1escapeinside以escapeinside结escapeinside束escapeinside客
        escapeinside询escapeinside)escapeinside"<<endl;
    string temp;
    cin>>temp;
    int select = -1;
    if(strstr(temp.c_str(),escapeinside"escapeinside-1escapeinside")){
        break;
    }
    for(int j = 0; j < type; j++){

        if(strstr(temp.c_str(),keyWord[j][0].c_str()) || strstr(temp.c_str(),keyWord[j][1].c_str
            ()) || strstr(temp.c_str(),keyWord[j][2].c_str()) || strstr(temp.c_str(),keyWord[j
            ][3].c_str()) || strstr(temp.c_str(),keyWord[j][4].c_str()) || strstr(temp.c_str(),
            keyWord[j][5].c_str()) || strstr(temp.c_str(),keyWord[j][6].c_str()) || strstr(temp.
            c_str(),keyWord[j][7].c_str()) || strstr(temp.c_str(),keyWord[j][8].c_str()) ||
            strstr(temp.c_str(),keyWord[j][9].c_str())){

            select = j;
            break;
        }
    }
    if(select == -1){
        cout<<escapeinside"escapeinside抱escapeinside歉，escapeinside没escapeinside能escapeinside匹escapeinside配
            escapeinside到escapeinside合escapeinside适escapeinside的escapeinside答escapeinside案escapeinside"<<endl;
        continue;
    }
    sprintf(fileName, escapeinside"escapeinside%escapeinsidedescapeinside",data[select]);
    string file = dir + fileName;
    string dataRead = getFileContents(file.c_str()); //文件绝对路径
    string tempans;
    for(int i = 0; i < dataRead.length(); i++){

        if(dataRead[i] == escapeinside'escapeinside\"escapeinside' && (dataRead[i+1] == escapeinside'escapeinside\
            escapeinsidernescapeinside' || dataRead[i+1] == escapeinside'escapeinside\0escapeinside')){


            if(!strstr(tempans.c_str(),escapeinside"escapeinside答escapeinside")){
                continue;
            }
            string len = strstr(tempans.c_str(),escapeinside"escapeinside答escapeinside");
            if(strstr(len.c_str(),keyWord[select][0].c_str()) || strstr(len.c_str(),keyWord[
```

```
                    select][1].c_str()) || strstr(len.c_str(),keyWord[select][2].c_str()) || strstr(
                    len.c_str(),keyWord[select][3].c_str()) || strstr(len.c_str(),keyWord[select
                    ][4].c_str()) || strstr(len.c_str(),keyWord[select][5].c_str()) || strstr(len.
                    c_str(),keyWord[select][6].c_str()) || strstr(len.c_str(),keyWord[select][7].
                    c_str()) || strstr(len.c_str(),keyWord[select][8].c_str()) || strstr(len.c_str()
                    ,keyWord[select][9].c_str())){
                    cout<<len<<endl;
                    break;
                }
                tempans.clear();
                len.clear();
            }
            tempans.push_back(dataRead[i]);
        }
    }
    cout<<escapeinside"escapeinside感escapeinside谢escapeinside您escapeinside的escapeinside支escapeinside持， escapeinside祝
        escapeinside您escapeinside生escapeinside活escapeinside愉escapeinside快escapeinside"<<endl;
    return 0;
}
```