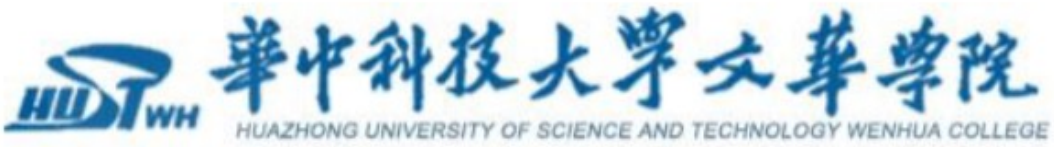


毕业设计（论文）

题目： 基于网络爬虫的 IT 专业需求信息挖掘系统

学 生 姓 名： 黎浔 学号： 120103021116

学 部 （系）： 信息科学与技术学部

专 业 年 级： 计算机科学与技术 2012 级 02 班

指 导 教 师： 鲁丽 职称或学位： 副教授

2016 年 5 月 10 日

目 录

基于网络爬虫的 IT 专业需求信息挖掘系统

摘　要

近些年来软件开发行业呈井喷式增长，各种领域内的互联网创业公司层出不穷，随之而来的是大学生的就业问题和学习方向的选择问题，很多公司同时也在面临一个如何优化招聘流程来更加高效地招聘合适的应届毕业生的问题。网络招聘这些年来获得了长足的发展，网上的招聘信息已经积累了非常大的一个数据量。个性化推荐系统又一直以来都是计算机技术中备受关注的一个领域，不断地都有新的技术出现，本课题将当前在电商、搜索及音乐领域内已经应用很广泛的机器学习和数据挖掘等技术应用在就业信息分类整理上，通过网络爬虫定向获取当前和过去的就业信息，挖掘出时下的就业需求信息，并在此基础上实现简单的就业推荐。

关键词： 网络爬虫；数据挖掘；信息推荐

IT Profession Demand Information Mining System Base On Web Spider

Abstract

In recent years the software development industry showed a growth spurt, in various fields within the emerging Internet start-ups, followed by a selection of college students employment problems and learning directions, while many companies are faced with an example of how to optimize the recruitment process more efficient recruiting suitable graduates problems. Online recruitment years gained considerable development, online jobs has accumulated a very large amount of data. Personalized recommendation system has always been a concern in the field of computer technology, continue to have new technologies emerge, the subject in the current electricity providers, search, and music has very broad application of machine learning and data mining technology in the employment information sorted and directed by web crawlers get current and past employment information, employment needs to dig out information nowadays, and on this basis to achieve a simple employment recommendation.

Key words: Web Spider; Data Mining; Information Suggest

前　言

随着大数据概念的热潮渐渐归于平静，对新技术的热捧时期已经过去，更多的焦点聚焦在思考有哪些地方可以使用现有的技术进行重组。对于绝大多数高校来说，计算机信息技术的教学依然相对落后，如何对专业未来几年的发展做一个规划，如何培养更符合社会需求的 IT 专业人才，是专业建设需要面临的问题之一。通过维护一个定向爬虫并且长时间积累数据，可以实现长久地自动跟踪当前热门的技术和热门的研究领域，为专业建设提供一定的参考信息。

目前大多数网站的信息相对闭塞，很少有网站会像豆瓣和微博那样提供开放的 API 来供用户查询数据，因此本系统选取了采用爬虫这个适应面相对较广的技术方向，通过爬虫可以把在某个领域内的不同来源的数据按照预先设定好的数据结构来存储，方便后期在此基础上进行进一步的数据挖掘以提取更多隐含的信息，同时也可以给学校教学方案改革提供一点参考建议。随着系统模型的不断优化，相信这一块一定会取得不错的成果。

在系统部署方面系统采用当下流行的 Docker 容器来提供完整的生产环境，希望通过容器来降低打包发布过程中的依赖问题，爬虫则采用业界已经非常成熟的 Scrapy 框架来构建爬虫系统，爬到的数据先缓存到 Redis，使用 Redis 作为爬虫的队列底层存储，存储过程中直接利用 Redis 的 Set 来进行判重，数据持久化方面则直接用 mysql 进行存储。

数据的运算平台使用 Hadoop 来提供分布式计算架构，方便后期的计算能力的拓展，同时还会提供一套 Restful 的 API 方便调用和集成。

1 绪论

1.1 背景介绍

连续三年来城镇新增就业的人数都在1300万人以上，而近年来经济又处于下行趋势，当前就业形势非常严峻。作为就业主力军之一的广大高校毕业生由于没有社会工作经验很难知道自己究竟适合什么样的职位；对于IT行业的需求而言，单是软件工程师这一个类别的职位就有数十种不同的分类。另一方面计算机专业的在校学生经常会有该如何开始学习编程之类的疑问，如果能从当前的市场需求中挖掘出当前的职位需求趋势，并利用这些数据为学校培养学生技能提供一定的指导依据，一定能让学生在探索并构建自己技能树的时候少走不少弯路。

同时，现在网络招聘信息急剧膨胀，单单靠几场招聘会已经很难满足学生的就业需求，但是如何甄别网上的招聘信息，如何快速从海量的招聘岗位中找到符合自己的岗位。如果采用传统的方法人工去筛选对比，不仅要耗费大量的时间可能还会错过最符合自己的公司岗位。

1.2 相关技术

1.2.1 网络爬虫

考虑到开发效率和维护效率，本系统选择使用Scrapy爬虫框架来完成数据爬取部分的任务。Scrapy是一个完全基于python编写的爬虫框架，框架设计得简洁优雅、易于拓展，例如基于其中间件机制可以极大地拓展爬虫的数据处理能力，更重要的是使用Twisted构建的异步非阻塞的编程模式，可以极大地提高爬虫的性能。Scrapy可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。其最初是为了页面抓取（更确切来说，网络抓取）所设计的，也可以应用在获取API所返回的数据(例如 Amazon Associates Web Services ）或者通用的网络爬虫。Scrapy的适用面非常广泛，可以应用在网络爬虫、数据变化监测、数据挖掘等方面。

用Scrapy框架可以直接利用成熟的设计模式和架构来构建一个健壮且易于维护的

爬虫项目，scrapy 的 pipeline 和 middleware 让整个爬虫的数据流变得很清晰，使得日后可以很方便地拓展爬虫的功能，再加上 Scrapy 使用了 Twisted 异步网络库来处理网络通讯，其整体性能也非常的优异。

1.2.2 中文分词

由于中文的词与词之间不像英文那样留有空格，在处理中文自然语言时所面临的第一个难题就是中文分词（Chinese text segmentation），做好中文自动分词是提取关键信息的前提。中文分词作为搜索引擎的基础研究领域，相关研究已经有不少成果，这里选择 python 的 jieba 作为分词工具。jieba 的分词算法采用基于前缀词典来扫描词图，并使用动态规划算法来计算最大概率路径从而找出基于词频的最大切分组合[1]，在没有引入相关领域词典的情况下效果并不是很好，为了进一步提高分词的精度与准确度还需要进一步优化。

精度（Precision）、召回率（Recall）、F 值（F-mesure）是用于评价一个信息检索系统的质量的 3 个主要指标，直观地说，精度表明了分词器分词的准确程度[2]；召回率也可认为是“查全率”，表明了分词器切分正确词汇的全面程度，F 值综合反映整体的指标，三个数值越大说明分词效果越好。

1.2.3 TF-IDF 词频计算与关键词提取

对一段中文描述做完分词处理后，接下来要做的就是用计算机提取它的关键词（Automatic Keyphrase extraction），通过公式 1-1 我们可以计算出（Term Frequency，缩写为 TF），但是单纯的计算词频往往会获得大量的噪音数据。通过给不同的词分配不同的权重可以计算出词汇的"逆文档频率"（Inverse Document Frequency，缩写为 IDF），他的计算公式如公式 1-2 所示。最后基于公式 1-3，利用之前由公式 1-1 和公式 1-2 计算出来的 TF 和 IDF 可以得到某个词的 TF-IDF 值，这个词对文章的重要性越高，它的 TF-IDF 值就越大。[3]

$$\text{词频(TF)} = \frac{\textit{某个词在文章中出现的次数}}{\textit{该文章出现频次最高的词的出现次数}} \qquad \text{公式 1-1}$$

$$\text{逆文档频率(IDF)} = \log\left(\frac{\textit{语料库的文档总数}}{\textit{包含该词的文档数}+1}\right) \qquad \text{公式 1-2}$$

$$TF - IDF = \text{词频(TF)} \times \text{逆文档频率（IDF）} \qquad \text{公式 1-3}$$

1.2.4 TextRank 算法提取关键词

TextRank 算法将文本素材进行分词切片处理，然后过滤掉无意义的词和字，还可以对切片出来的词汇集合进行词性的筛选过滤，最后输出结果集合。

每个单词作为 PageRank 中的一个节点。设定窗口大小为 K，假设一个句子依次由下面的单词组成：

$$\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \dots, \omega_n$$

$[\omega_1, \omega_2, \dots, \omega_k], [\omega_2, \omega_3, \dots, \omega_{k+1}], [\omega_3, \omega_4, \dots, \omega_{k+2}]$等都是一个窗口。在一个窗口中的任两个单词对应的节点之间存在一个无向无权的边[4]。基于上面构建的图，可以计算出每个单词节点的重要性。最重要的若干单词可以作为关键词。若原文本中存在若干个关键词相邻的情况，那么这些关键词可以构成一个关键短语。

1.2.5 特征向量

利用从招聘网站上搜集到的描述信息，可以对每个职位信息进行特征建模，根据提取出来的关键词按 TD-IDF 值排序并去除噪音词汇后作为特征向量，每个职位信息由于关键词及词频不一致会构成不同的词频向量，通过计算这些词频向量与用户的特征向量的余弦夹角的大小可以粗略判断该职位与用户的匹配程度。

1.2.6 热度排序算法

为了保证每个职位热度的时效性，需要给计算出来的热门职位加上一个时间相关因子，可以借鉴“牛顿冷却定律”（Newton's Law of Cooling）来构建一个“指数衰减”（Exponential decay）模型来构建一个算法[5]，牛顿冷却定律说的是温度随时间变化的曲线，其曲线如图 1-1 和图 1-2 所示。

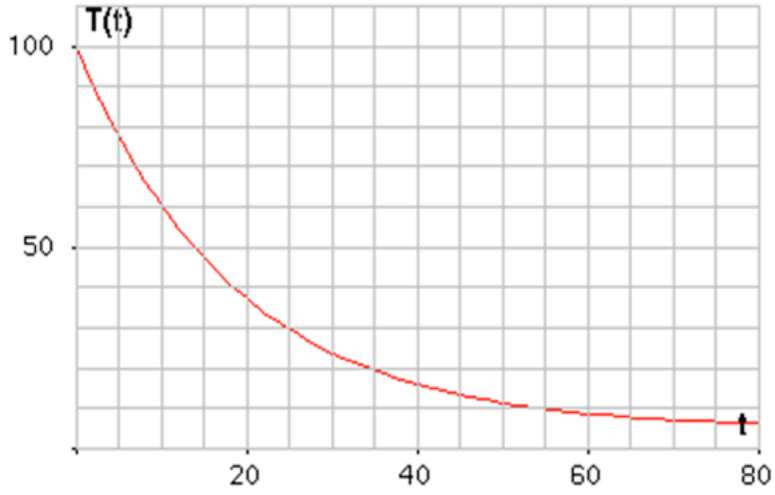


图 1-1 牛顿冷却曲线

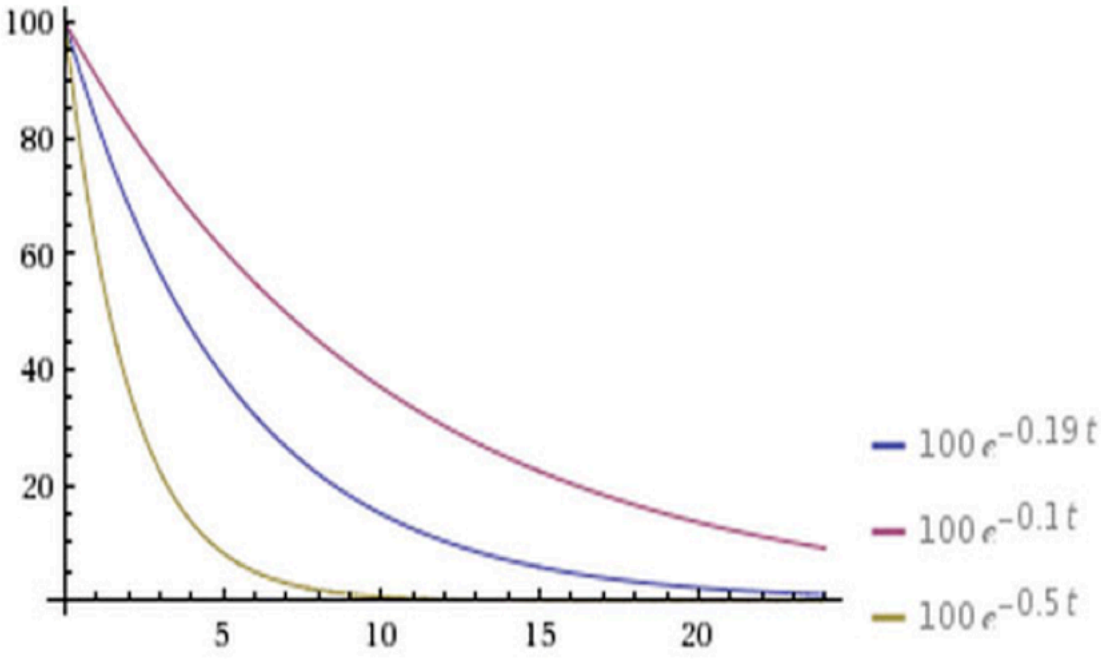


图 1-2 系数不同的牛顿冷却曲线

公式 1-4 是牛顿冷却曲线的微积分形式，其中 T 为热度的时间函数，H 代表当前热度，常数α $(\alpha > 0)$表示热度与时间之间的比例关系其负号代表热度在递减，最终可以利用公式 1-5 在程序中进行迭代计算。

$$\int \frac{T\prime(t)}{T(t)-H} dt = \int(-\alpha)dt$$ 公式 1-4

$$current = last * \exp(-\varepsilon * gapTime)$$ 公式 1-5

1.2.7 贝叶斯分类算法

在机器学习中，朴素贝叶斯分类器是一系列以假设特征之间强（朴素）独立下运用贝叶斯定理为基础的简单概率分类器。贝叶斯分类器（Bayes Classifier）的分类原理是通过某对象的先验概率，利用贝叶斯公式计算出其后验概率，即该对象属于某一类的概率，选择具有最大后验概率的类作为该对象所属的类。这里主要应用朴素贝叶斯进行文本分类，其中朴素贝叶斯最重要的特性就是假定各个特征值之间是相互独立的，其基本表示形式如公式 1-6 所示，用概率模型来表示的朴素贝叶斯公式如公式 1-7。[6]

$$\mathrm{P}(C|F_1,\dots,F_n) = \frac{P(F_1,\dots,F_n|C)\,P(C)}{P(F_1,\dots,F_n)}$$ 公式 1-6

$$classify(f_1,\dots,f_n) = argmax\ p(C=c)\prod_{i=1}^{n} p(F_i=f_i|C=c)$$ 公式 1-7

2 需求分析与系统设计

2.1 需求分析

整个系统可以拆分为分布式爬虫系统、数据挖掘分析系统、数据可视化系统、信息推荐系统。前期功能主要集中在分布式爬虫系统的构建与优化，主要定位于爬取拉勾网这类招聘网站上发布的就业信息，将爬取到的信息归类整理并进行中文分析处理，通过关键词词频来提取其中有价值的关键信息，并根据这些信息构建数学模型。

数据分析直接利用 Hadoop 的分布式计算框架来解决后期数据量增加导致的运算量不断变大的问题[7]，通过对不断增加的大量数据进行处理来判断当前的热门职位变化情况。

不断积累的数据同时也需要将数据以生动友好的方式展现出来，数据可视化系统可以根据不同条件的输入来展示不同类型的数据图形。

系统积累了一定的数据和用户之后，考虑利用协同过滤来推荐一些适合的技术方向和热门的技术领域给使用者。

根据调研及分析，本系统基本需求如下：

（1）爬虫系统需要尽可能过滤数据噪音，对于重复的数据需要尽可能有效的去除；

（2）不能因为某一个分支出现故障引起整个爬虫的崩溃，对于错误需要详细记录下必要的日志信息以供分析排查；

（3）绕开网站的限制与封锁的同时尽可能提升效率；

（4）对于爬取的数据要求尽可能的多样化，不能在某个分支消耗过多的资源；

（5）能应对数据量剧增的情况，保证分析结果的实效性；

（6）分析结果能够根据筛选条件以直观的方式展示出来；

（7）整个系统模块化，模块之间耦合尽可能低，数据能够以 api 方式调用。

2.2 系统设计

2.2.1 数据库设计

因为网站上的职位列表和职位详细描述是分开的页面，为了提高爬取效率，本系统对职位列表和职位详情的描述进行的是异步抓取，先遍历存储列表，同时把列表中的职位详情的链接压入爬取队列，这样即可实现两部分分开爬取。与此同时，在数据库层面也把列表数据结构和详情数据结构分了两张表进行存储以方便爬取过程中的数据库操作，两张表通过职位 id（position_id）外键来实现关联，数据库模型如图 2-1。

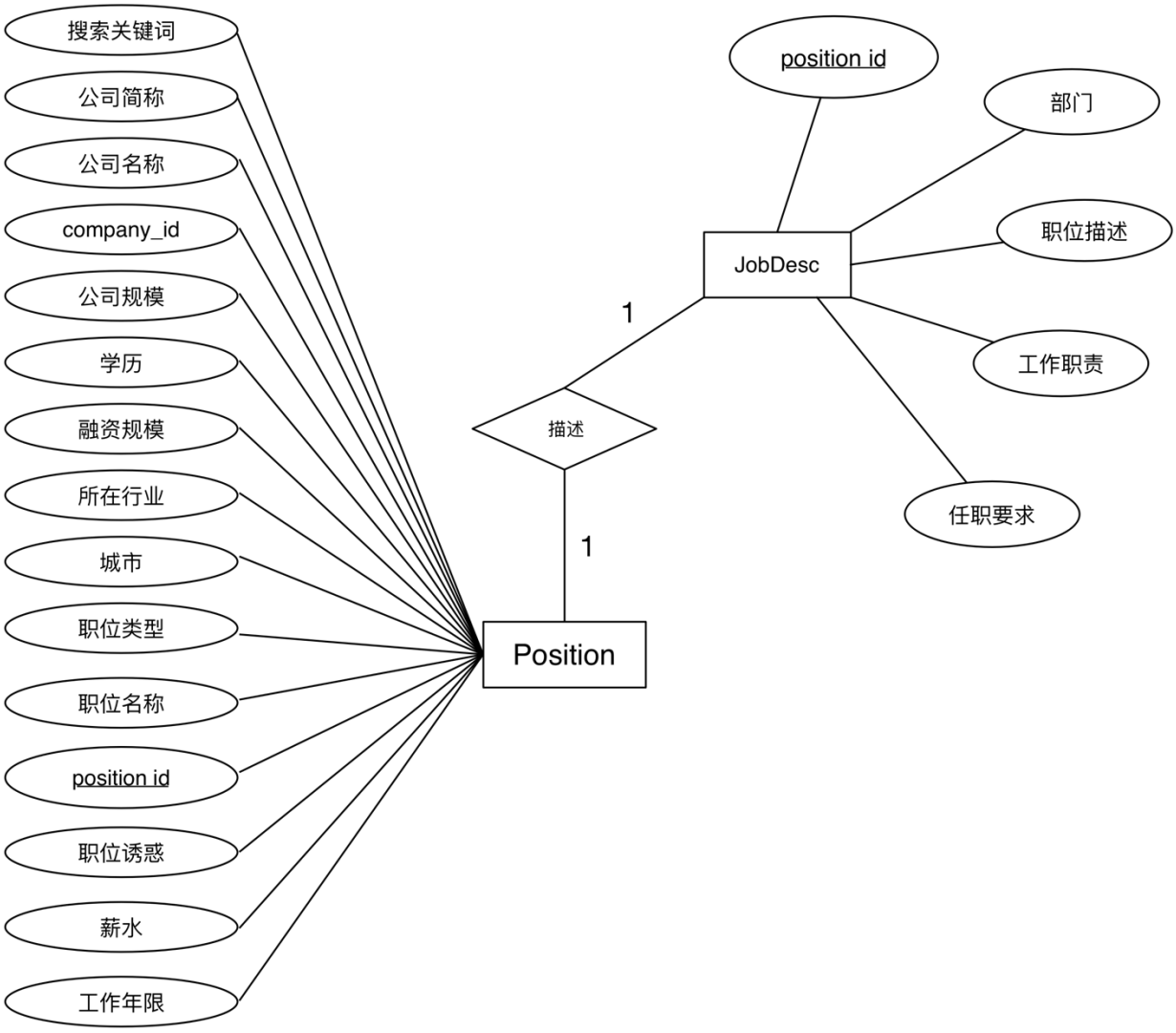


图 2-1 数据库 ER 图

2.2.2 架构设计

爬虫的架构如图 2-2 所示。

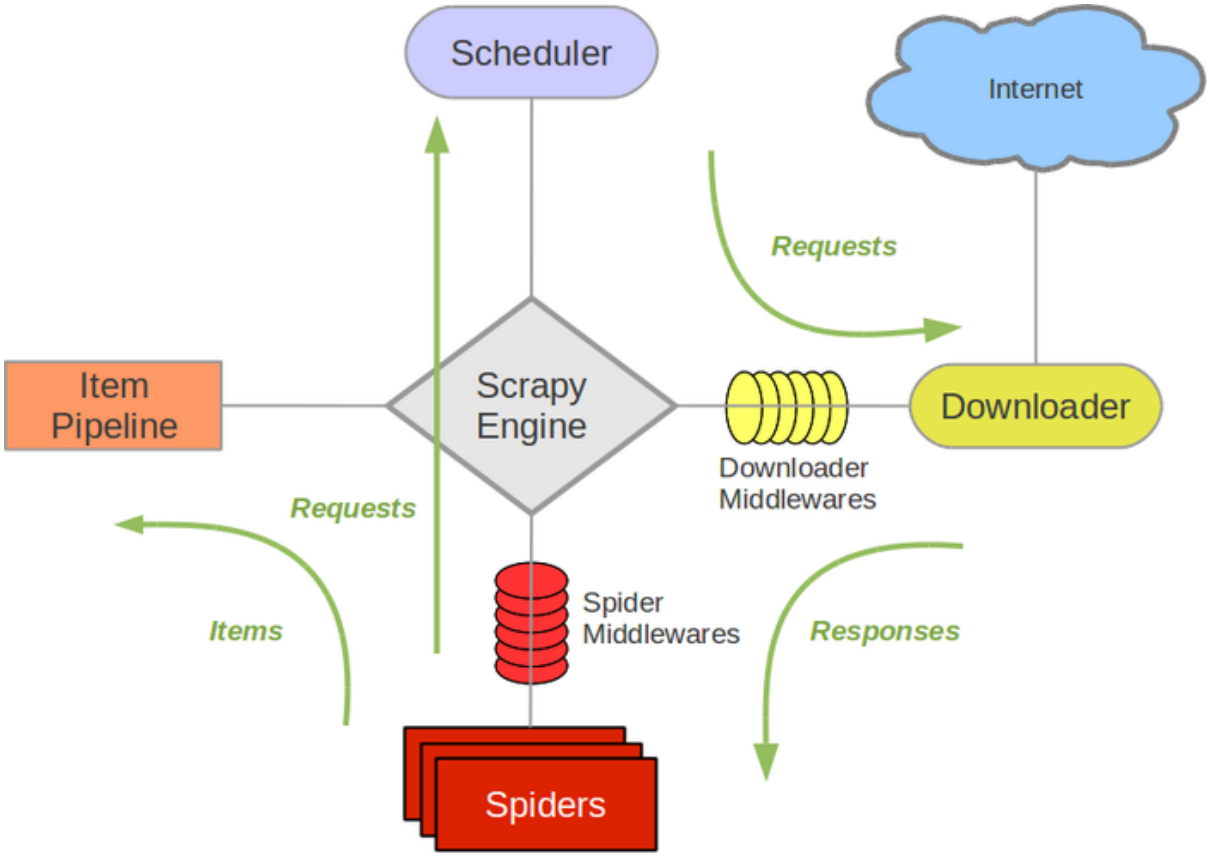


图 2-2 Scrapy 架构图

由 Spider 实例发起爬取请求，调度器（Scheduler）从 Scrapy 引擎接受请求，排序后将请求压入队列，并在 Scrapy 引擎发出请求后返还给他们[8]。

引擎打开一个网站，找到处理该网站的 Spider 并调用该爬虫的 start_request 方法获取初始爬取列表。

接下来 Engine 从 Spider 中获取到第一个要爬取的链接，调度器（Scheduler）从 Scrapy 引擎接受请求，排序后将请求压入队列，并在 Scrapy 引擎发出请求后返还给引擎进行 Request 请求的调度。

爬虫引擎会将 URL 链接通过下载中间件（Download Middleware）转发给下载器（Downloader），页面下载完毕之后下载器会将该页面的 Response 通过下载中间件发送给引擎。

爬虫引擎接到下载器返回来的 Response 之后通过 Spider 中间件交给 Spider，Spider 会从 Response 中利用设定的 xpath 规则提取 Items 同时提起深一层的 Request 请求给爬虫引擎，之后引擎会将提取出来的 Items 交给 Item Pipeline 将 Request 转交给调度器，在 Item Pipeline 中本系统将数据进行处理并存入数据库（或写入文件）。然后会一直重复，直到调度器队列中没有剩下的 Request 请求，至此爬虫结束。

2.2.3 爬虫设计

1．爬虫数据结构

爬虫的数据结构分为 PositionItem 和 JobDescItem，如图 2-3 所示，其中 Position-Item 是简要信息的数据结构，JobDescItem 是详细信息的数据结构。

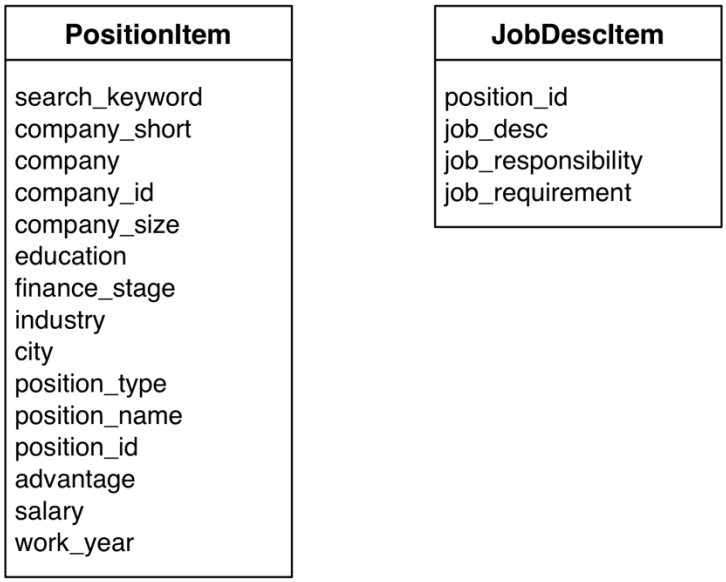


图 2-3 爬虫数据结构

2．爬虫管道设计

在爬虫运行的过程中间会链接数据库，并把用 xpath 解析出来的相关信息异步存储到数据库里，通过为不同的管道设置优先级可以确定它们的运行顺序，为了方便数据库连接操作，这里先定义了一个 ItemPipeline 基类，在基类中实现了有关数据库连接之类的操作，其余的管道类只需要继承这个基类就可以直接调用继承自父类的数据库操作方法。这样，不同的爬虫管道只需要实现各自的 process_item()方法就好了，

数据库的操作实现细节对于积累来说是屏蔽的，后期就算要换数据库也只需要多实现一个 ItemPipeline 的父类，或者直接重载父类的数据库操作方法就好了。具体的继承结构可以参见图 2-4。

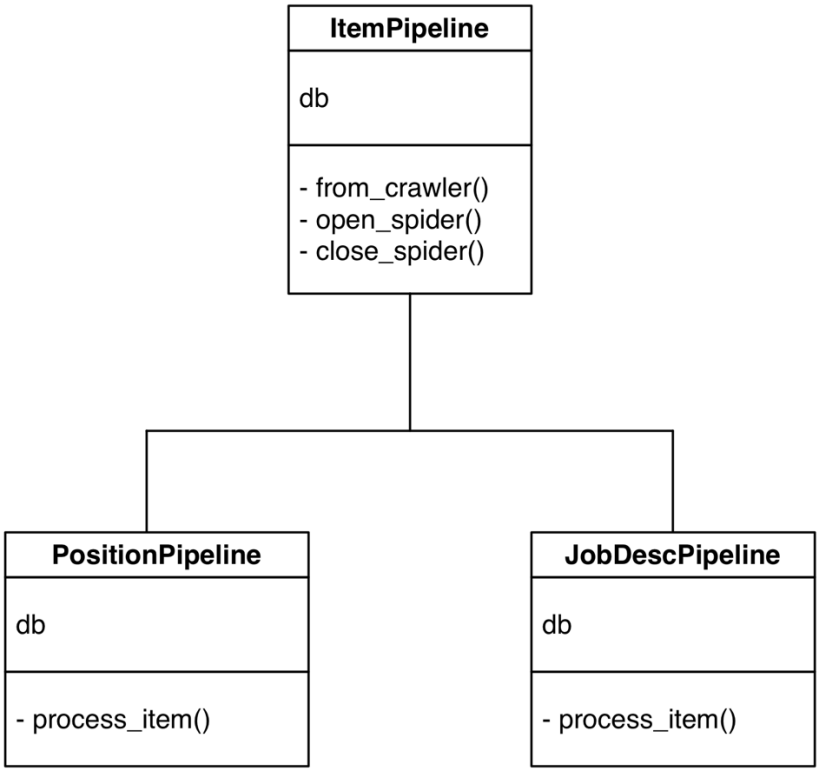


图 2-4 爬虫管道类 UML 图

3．管道过滤器

至此还需要构建一个去除重复数据的过滤器，丢弃那些已经被处理过的 item。例如 item 有一个全局唯一 id，此时可以直接根据这个 id 来进行去重操作，如下所示。

```
from scrapy.exceptions import DropItem

class DuplicatesPipeline(object):
def __init__(self):
    self.ids_seen = set()

def process_item(self, item, spider):
    if item['id'] inself.ids_seen:
        raise DropItem("Duplicate item found: %s"%item)
    else:
        self.ids_seen.add(item['id'])
        return item
```

3 系统实现

3.1 数据爬取

3.1.1 现有的招聘网站介绍

目前比较流行的招聘网站有前程无忧、智联招聘、拉勾网、猎聘网等等[9]，传统的招聘网站数据比较庞杂，数据分类也不太完善，这里出于简化模型的考虑决定主要爬取拉勾网的数据来进行建模分析，图 3-1 是拉勾网筛选工作的页面。

为了获得高的精度，效果是良好的模型中，大量的数据是必要的。只有数据的一定程度的量，该模型，以实现可接受水平的正确率，该模型的量和复杂性之间的关系，置信率数据（参考： VC 维），为了获得大量符合要求的数据，本系统在这里采用的是爬虫的手段，定向爬取的过程过可以把数据统一格式化后存入数据库，然后再对采集到的数据进行进一步处理。

图 3-1 拉勾网工作信息搜索页面

3.1.2 爬取算法

考虑到爬虫的性能，主要基于广度优先搜索的策略进行数据爬取[10]。因为拉勾网主站采用 AJAX 的方式实现数据加载，通过抓包分析可以直接找出其 API，通过变换参数请求其 API 结构获取数据列表，首先通过控制分页来请求完整的列表数据，之后将列表优先存储进数据库。

在爬取完当前关键词最后一页的列表后异步爬取列表中每一个职位的详细信息。爬取的过程中会检测是否有重复数据，若有重复数据则丢弃该数据。

查重算法是直接基于全局唯一 id 来判断，如果利用 redis 来进行存储，查重的时间复杂度可以控制在 O(1)。

3.1.3 爬取实现

本系统主要基于 pyhton 的 Scrapy 框架实现了一个较为简单清晰的爬虫，通过控制下表来读取预先设定的关键词列表和城市列表来发起 Request 请求，每一次 Request 请求之后计算当前请求数据的页数，然后通过页码与 url 拼接后压入队列来获取完整的数据。

```
keywords = ['javascript', 'ios', 'android', 'php', 'java', '.net',
'c', 'python']
cities = ['武汉', '深圳', '广州', '北京', '上海']
keyword = 'javascript'
city = u'武汉'
pn = 1  # 当前页码
k = 0  # keyword 下标
c = 0  # city 下标


totalSize =总共数据数目
pageSize =每页数据数目
pnum = totalSize / pageSize# 计算总页码


city = cities[c]
keyword = keywords[k]
```

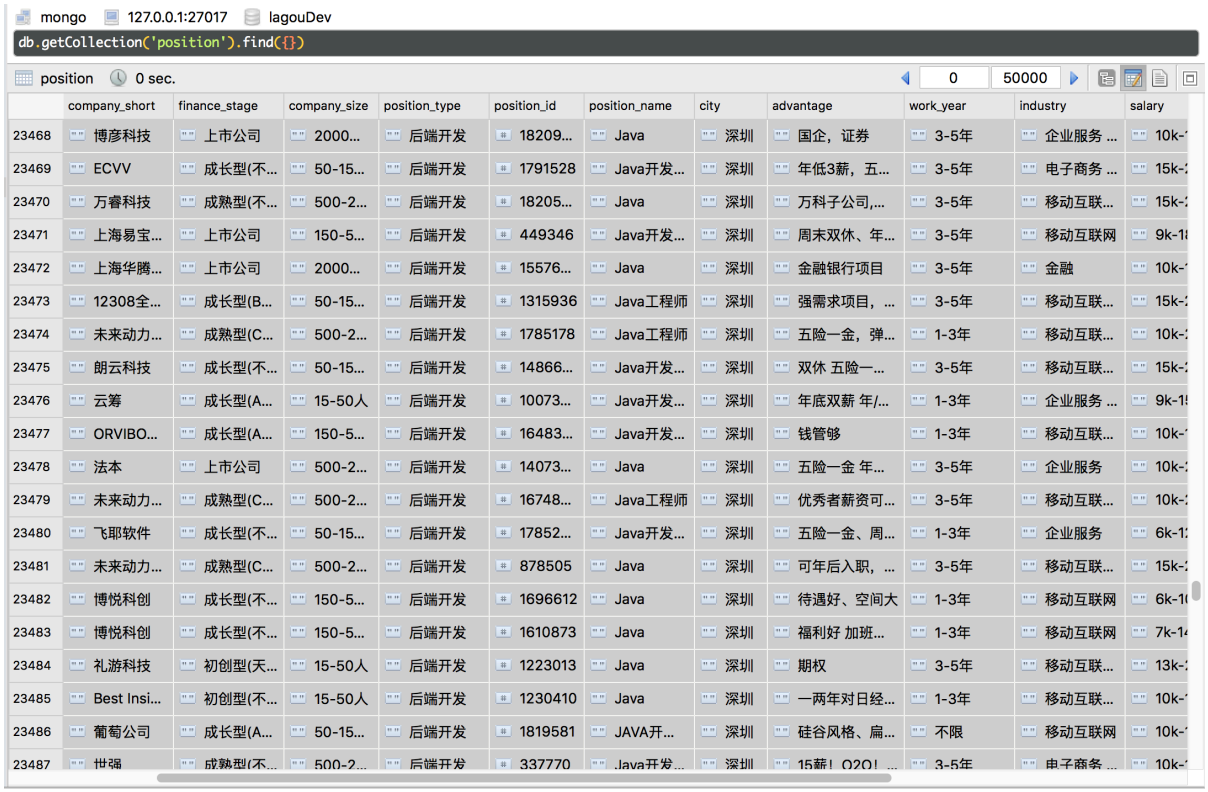
爬取过程的伪代码描述如下。

```
if 数据获取失败
    记录错误日志
    return
else
   for i in range(self.pageSize):
       构建 position 对象
       yield position
       将职位详情的 url 压入队列并设置用于解析职位详情的回调函数

pn += 1# 页码自增
if pn > pnum:
    if c == 4 and self.k == 关键词数组长度 - 1:
        记录错误日志
        return
    else if k == 关键词数组长度 - 1:
        k = 0
        c += 1
    else:
    k += 1
    拼接 url 并压入队列
更新 city、keyword
将下一页 url 压入队列
```

3.1.3 数据存放

随着大数据的不断发展，非关系型数据库（NoSQL）已经变得越来越流行，MongoDB 作为一个文档型数据库，用文档来组织数据，不需要严格的结构，具有高性能易拓展等优点。考虑到本系统对数据库性能要求高，而对数据的完全一致性并没有严格的要求，所以系统采用 MongoDB 来作为数据库存储（也可以直接写入 json 文件），最后数据库查询结果如图 3-2。



	company_short	finance_stage	company_size	position_type	position_id	position_name	city	advantage	work_year	industry	salary
23468	博彦科技	上市公司	2000...	后端开发	18209...	Java	深圳	国企，证券	3-5年	企业服务 ...	10k-
23469	ECVV	成长型(不...	50-15...	后端开发	1791528	Java开发...	深圳	年低3薪，五...	3-5年	电子商务 ...	15k-
23470	万睿科技	成熟型(不...	500-2...	后端开发	18205...	Java	深圳	万科子公司,...	3-5年	移动互联...	15k-
23471	上海易宝...	上市公司	150-5...	后端开发	449346	Java开发...	深圳	周末双休、年...	3-5年	移动互联网	9k-1
23472	上海华腾...	上市公司	2000...	后端开发	15576...	Java	深圳	金融银行项目	3-5年	金融	10k-
23473	12308全...	成长型(B...	50-15...	后端开发	1315936	Java工程师	深圳	强需求项目，...	3-5年	移动互联...	15k-
23474	未来动力...	成熟型(C...	500-2...	后端开发	1785178	Java工程师	深圳	五险一金，弹...	1-3年	移动互联...	10k-
23475	朗云科技	成长型(不...	50-15...	后端开发	14866...	Java开发...	深圳	双休 五险一...	3-5年	移动互联...	15k-
23476	云筹	成长型(A...	15-50人	后端开发	10073...	Java开发...	深圳	年底双薪 年/...	1-3年	企业服务 ...	9k-1
23477	ORVIBO...	成长型(A...	150-5...	后端开发	16483...	Java开发...	深圳	钱管够	1-3年	移动互联...	10k-
23478	法本	上市公司	500-2...	后端开发	14073...	Java	深圳	五险一金 年...	3-5年	企业服务	10k-
23479	未来动力...	成熟型(C...	500-2...	后端开发	16748...	Java工程师	深圳	优秀者薪资可...	3-5年	移动互联...	10k-
23480	飞耶软件	成长型(不...	50-15...	后端开发	17852...	Java开发...	深圳	五险一金、周...	1-3年	企业服务	6k-1
23481	未来动力...	成熟型(C...	500-2...	后端开发	878505	Java	深圳	可年后入职，...	3-5年	移动互联...	15k-
23482	博悦科创	成长型(不...	150-5...	后端开发	1696612	Java	深圳	待遇好、空间大	1-3年	移动互联网	6k-1
23483	博悦科创	成长型(不...	150-5...	后端开发	1610873	Java	深圳	福利好 加班...	1-3年	移动互联网	7k-1
23484	礼游科技	初创型(天...	15-50人	后端开发	1223013	Java	深圳	期权	3-5年	移动互联...	13k-
23485	Best Insi...	初创型(不...	15-50人	后端开发	1230410	Java	深圳	一两年对日经...	1-3年	移动互联...	10k-
23486	葡萄公司	成长型(A...	50-15...	后端开发	1819581	JAVA开...	深圳	硅谷风格、扁...	不限	移动互联网	10k-
23487	世强	成熟型(不...	500-2...	后端开发	337770	Java开发...	深圳	15薪！O2O！...	3-5年	电子商务 ...	10k-

图 3-2 存储在 MongoDB 中的部分数据

3.2 数据的清洗

由于直接从网上爬下来的数据没有经过验证和筛选，其中必然会存在重复、残缺、失效甚至无意义的脏数据，为了保证分词以及关键词提取的准确度，在进行分词等操作之前需要先进行数据的清洗。

首先要做的就是数据的去重，首先根据职位 id 把整个数据库全部过滤一遍，判断是否存在重复数据。

去处重复数据之后，主要的问题是解决数据质量问题，也就是解决数据的各种问题，包括但不限于下列各项。

（1）数据的完整性——例如职位信息中缺少薪水、等公司信息；

（2）数据的唯一性——例如不同来源的数据出现重复的情况；

（3）数据的权威性——例如同一个指标出现多个来源的数据，且数值不一样；

（4）数据的合法性——例如获取的数据与常识不符，薪水超出社会普遍认知；

（5）数据的一致性——例如不同来源的不同指标，实际内涵是一样的，或是同一指标内涵不一致；

主要处理方法如下：

（1） 直接利用 sql 语句进行职位 id 去重；

（2） 对于 id 去重后的数据用代码进行遍历，检测字段是否为空、是否乱码、职位描述是否足够字数（职位描述如果字数过少说明职位抓取不完全或者该条数据质量过低），同时检测是否存在超出常规认知的数据（工资范围异常、工作地点不存在）等等；

（3） 对工作岗位进行 group 分类，对于相似的职位进行分类，统一职位描述字段以方便后期检索；

（4） 将工资进行数字提取，统一格式化为整数格式，分两个字断存储（工资区间）。

3.3 关键信息挖掘

由于不同的招聘主管对职位描述的语言组织风格都不太一样，甚至会出现大量网络流行语、口头语、专业信息缩写等各种问题，而想要对每个职位进行建模首先要做的就是对抓取到的各个职位信息进行关键词和关键短语的提取。

3.3.1 招聘描述关键信息提取

对 IT 行业热门需求进行数据挖掘，这里本系统主要利用 python 的 scikit-learn 和 jieba 分词进行中文分词切片操作,通过这两个非常成熟的工具库再根据一些特定需求可以较为快速的构建一个文本词汇分类器原型。使用 jieba 对文本进行分词，再使用 sklearn 的 feature extraction 模块对词袋进行 Hashing Trick 得到向量形式，然后再训练 bayes 分类器，最后使用 bayes 分类器对未知的文本进行分类[11]。分词运行过程如图 3-3，分词结果参见图 3-4。

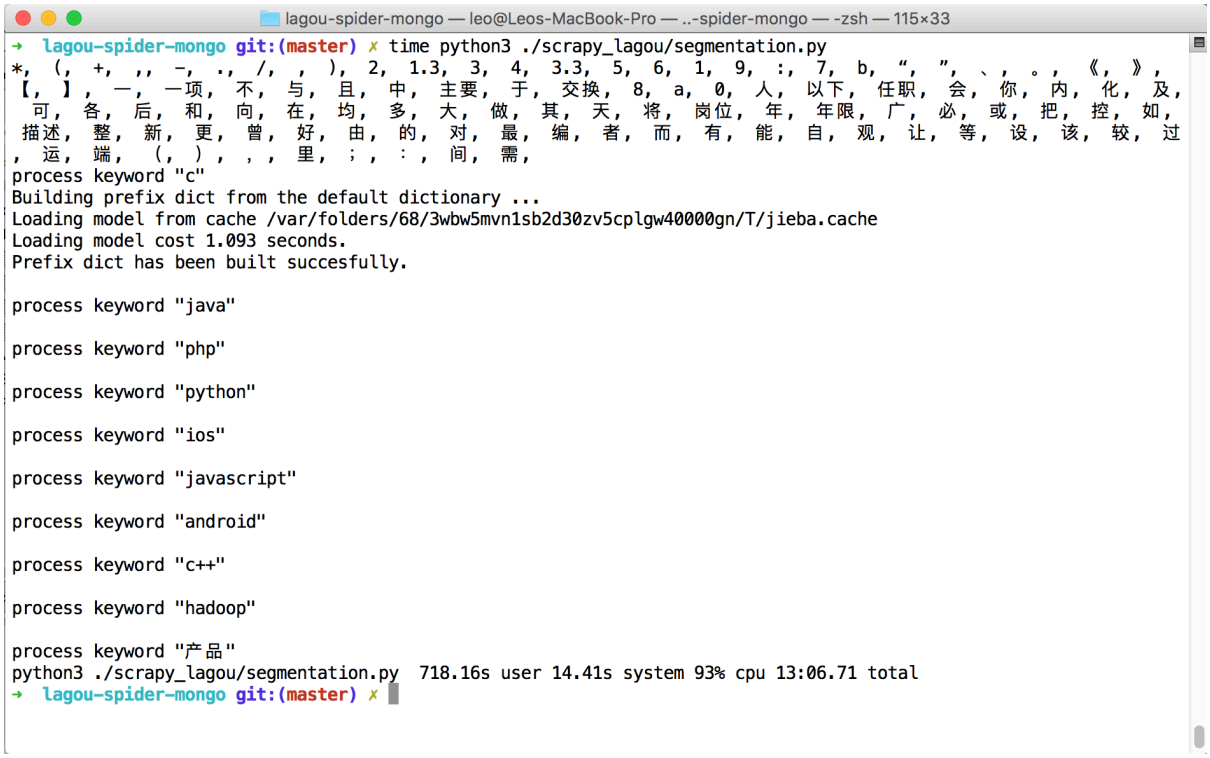


图 3-3 分词处理的结果

这一步，共有以下阶段：

(1)特征提取：对于文本分类，特征数据可以是：频率、TF-IDF，由于字典的规模会非常的大，可以使用 Hashing Trick 技巧来进行降维操作。当然，Hashing Trick 参数的选择是非常重要的，太大了会导致运行缓慢甚至内存溢出，太低了会导致碰撞的几率加大，降低模型准确性。

(2) 训练模型：接下来就是将进行特征提取之后的特征向量喂给分类器。 通常，这类数据的数据结构是 sklearn 内建的 spread 矩阵或者是 numpy 的 array. 关于 sklearn 的系数矩阵。

(3)交叉验证：使用 Cross Validate 模块可以验证不同的数据模型在处理同一个数据时的区别，这可以方便我们快速修正模型。

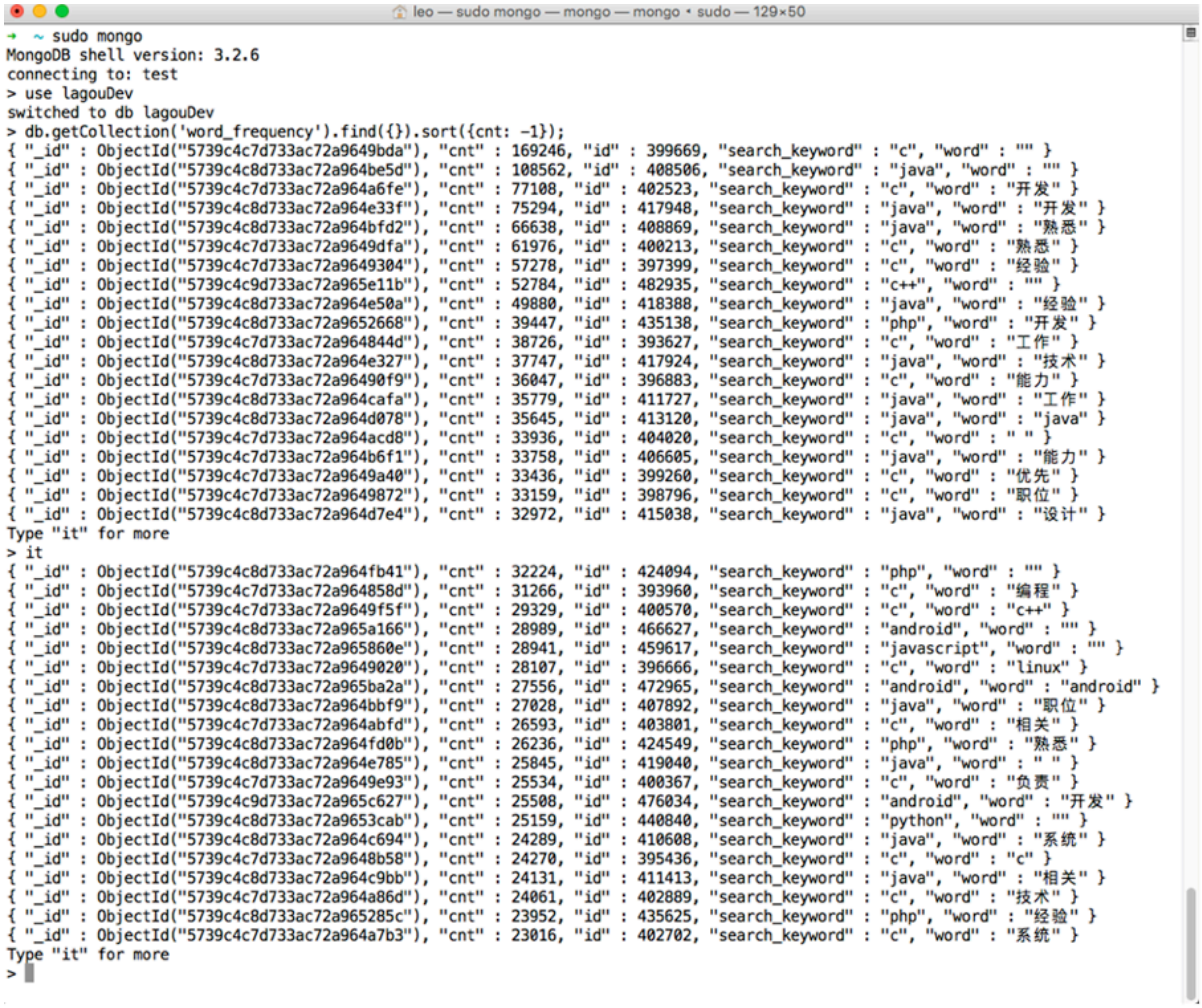


图 3-4 分词和词频计算后的结果从 MongoDB 中按词频降序查询结果

3.4 模式匹配

为了方便用户根据条件进行相关查询，接下来需要对数据和用户进行建模，通过模型的匹配算法可以算出最适合用户的职位集合，理论上模型越准确、基础数据量越大算出来的结果就会越精准[12]。

3.4.1 计算相似度

相似性度量（Similiarity），即评估个体之间的相似性，相似性度量的值越小，表明个体之间的相似性越小，越越大相似度值的个体差异。

对于一些不同的文本或短文本消息，关于如何计算它们之间的相似程度的对话，一个好的做法是将这些文本的话，映射到向量空间，使文本和矢量数据的文本形成一

个映射关系，并利用这个关系来计算文本的相似性。目前常用的算法有通过计算样本空间内特征向量的余弦距离、欧式距离（euclidean metric）或杰卡德距离（Jaccard Distance）来分析特征向量的相似程度大小[13]。

3.4.2 利用余弦相似性求相似度

求余弦相似度也就是求特征向量的余弦距离，直接计算两个特征向量间的余弦夹角作为两个职位间的相似程度的表示，再通过计算用户特征的向量与各个相似职位集合向量的余弦夹角则可以粗略算出用户与对应职位的匹配程度。

假定 A 和 B 是两个 n 维向量，A 是 [A1, A2, ..., An] ，B 是 [B1, B2, ..., Bn] ，则 A 与 B 的夹角 θ 的余弦值可以用公式 3-1 计算得出。

$$\cos\theta = \frac{\sum_{i=0}^{n}(A_i * B_i)}{\sqrt{\sum_{i=0}^{n}(A_i)^2} * \sqrt{\sum_{i=0}^{n}(B_i)^2}} \qquad \text{公式 3-1}$$

本系统具体做法如下：

（1）首先，把分词的结果进行过滤处理，去除掉不想关的和不符合语境的关键词（把搜狗的专业词库格式转换后作为 jieba 的本地自定义词库）；

（3） 把过滤之后的关键词作为一个 n 维特征向量；

（3）把待匹配的对象进行分词操作并提取关键词，并利用分词结果构建该对象的特征向量；

（4）对待匹配对象利用公式 3-1 求两个特征向量间的余弦距离，求出来的余弦距离值越接近 1 就说明两个特征向量越相似。

3.5 难点突破

3.5.1 调试

基于并发性考虑并且实现异步非阻塞的爬虫虽然性能上有了质的提高，运行中就算出了异常也不会终止其继续运行，但是这样随之而来的第一大问题就是调试也变难了。为了解决调试问题，在调试模式下运行时会把读取到的 json 格式原始数据直接写在文件里，同时在文件里记录下运行时的一些中间变量的值（包括当前请求的城市、查询关键词、当前查询的结果总数目、当前页码等），如图 3-5 所示。如果出了严重的问题则会直接把错误代码以及错误发生时候的详细信息写在日志文件里，这样以来可以直接写脚本程序批量去扫描日志文件和 json 文件就可以知道错误的具体位置和原因了，大大提高排错的效率和自动化程度。

图 3-5 调试时将中间变量写进文件

3.5.2 绕开限制

很多网站出于保护性措施，往往会对爬虫进行封锁，为了绕开网站的反爬虫机制可以采用以下方案来绕开限制：动态设置 user agent、禁用 cookies、设置延迟下载、使用 Google cache、使用 IP 地址池（Tor project、VPN 和代理 IP）、使用 Crawlera。其中最为关键的地方就在于动态代理即动态切换 IP。

综合衡量多种策略之后，本系统主要从动态随机设置 user agent、禁用 cookies、设置延迟下载和使用代理 IP 这几个方式来尝试，虽然可以解决被封锁 ip 导致不能访

问（大多数时候直接被强行重定向）的问题，但是如何获取大量高质量的代理又成为了另外一个问题，如果没有大量的资源提供而仅仅依靠单机，恐怕很难获取到大量的数据。增加代理池后的爬虫架构参见图 3-6。

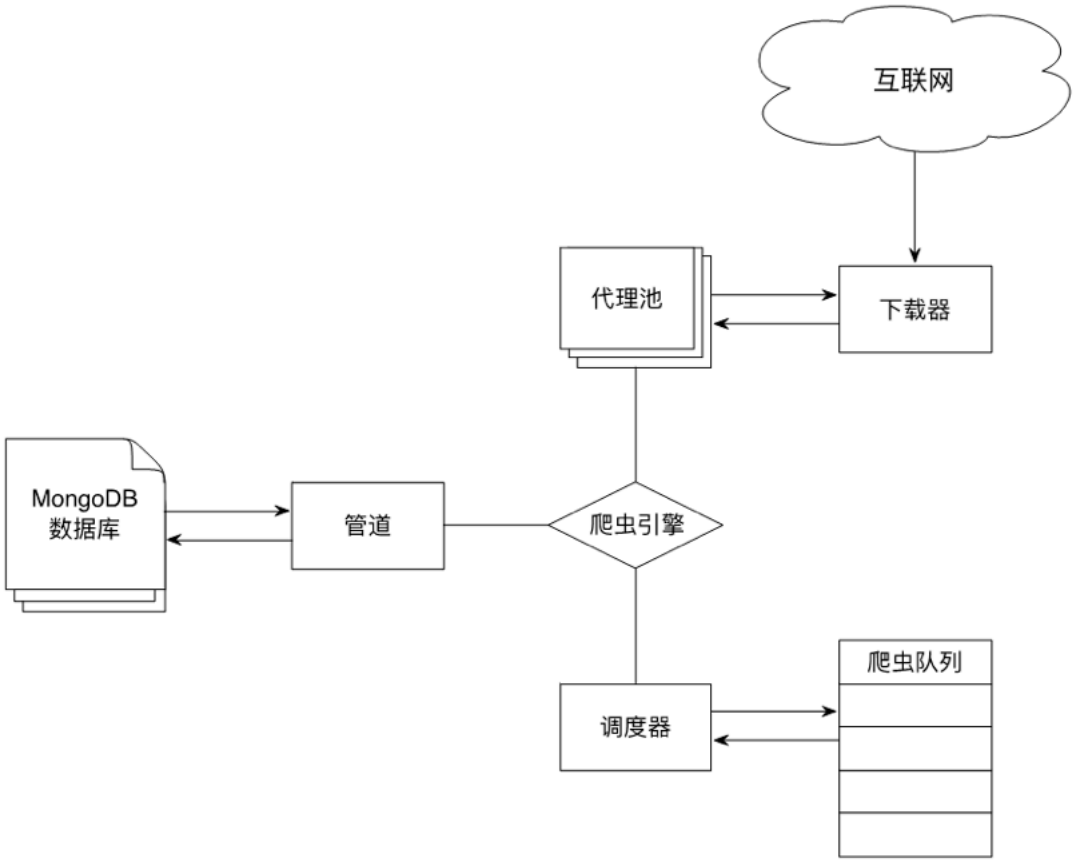


图 3-6 增加代理池后的爬虫架构图

3.5.3 高性能爬虫

从一开始每秒爬取接近 300kb 的网页，到后来被封 ip，再到后来利用代理池绕开封锁又接着爬了好几天的数据后慢慢觉得爬虫的速度直接就影响了获取数据量的大小。这个时候就需要考虑如何有效的提高爬虫的效率，简单来讲可以把 pyhton 换成 Go 语言这种静态语言，但是实际测试后发现爬虫的瓶颈更多是在 IO 尤其是网络 IO 上，从短板效应来看编程语言本身的性能反而没有太多影响。

最常见的做法就是构建一个分布式爬虫系统，利用分布式计算的吞吐能力去拓展爬虫的 IO 性能。简单点可以采用如图 3-7 所示的“主从式”分布式架构来进行爬虫性

能的拓展。

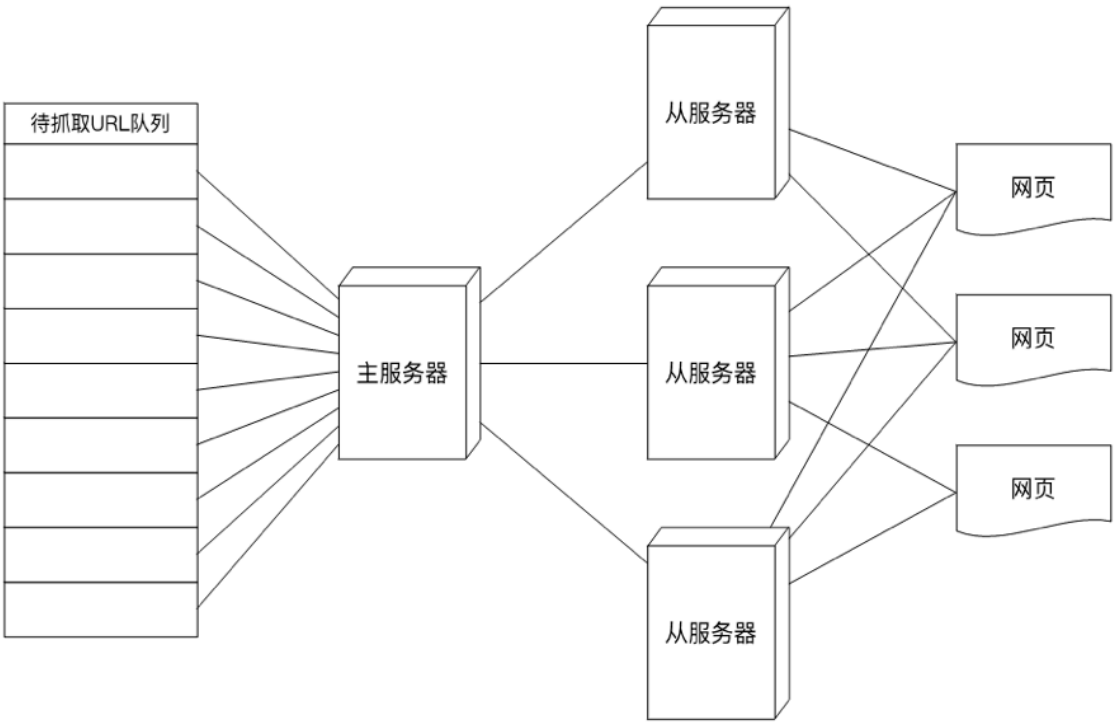


图 3-7 分布式爬虫架构图

4 数据可视化

由于爬虫爬取到的数据和分词处理后的数据全部都是直接存在数据库中，为了更加方便的查看数据，同时也为了更加直观的查看数据分析的结果，可以在爬虫和数据挖掘分析系统的基础上构建一个 Restful 风格的后端 API 模块和一个组件化的前端模块，将海量的数据统计分析结果采用图表的方式直接展示出来。出于跨平台和易于分享查看的考虑，决定优先采用基于网页的数据可视化方案，直接将分析结果的传给 Web 前端，由 Web 前端页面来负责绘制图表，具体形式类似于图 4-1。

图 4-1 国内某 JavaScript 数据可视化库

4.1 后端 API 设计

为了便于跨平台跨终端拓展，这里采用 Restful 风格的 API 来规范数据获取接口的标准，具体路由划分如表 4-1 所示。

4.2 前端组件化模块

配合后端的 API 接口，前端采用的是基于 React.js 的组件来构建 Web 页面。React.js 是由 Facebook 推出的一个前端框架，不仅使得前端页面可以彻底实现组件

化分离，还可以实现高性能的单页面渲染应用，非常适合数据展示这类对页面性能和体验要求非常高的应用场景。

表 4-1 Restful Api 接口规划

路由	含义
/api/v1/position/all	返回所有职位的摘要要信息
/api/v1/position/:id	返回某个职位的详细信息
/api/v1/keywords	返回关键词列表
/api/v1/user	返回用户信息
/api/v1/recommend	返回与用户偏好匹配的职位

最终实现的架构如图 4-2 所示，其中 API 接口层用的 Node.js + Express 来提供高性能的数据查询服务，充分利用了其异步非阻塞 IO 模型和 JavaScript 对 JSON 的友好支持。Web 层采用 React + ReactRouter 实现了高性能无刷新的单页应用，可以高效地用表格渲染出大量数据数据。数据库层用 MongoDB + Mongoose 来快速处理数据库查询请求，由于 NoSQL 数据库和 Node.js 的性能优势本系统可以解决 C10K 级别的并发请求，可以很好的解决拓展性问题，具体的实现效果图如图 4-2 和图 4-3 所示。

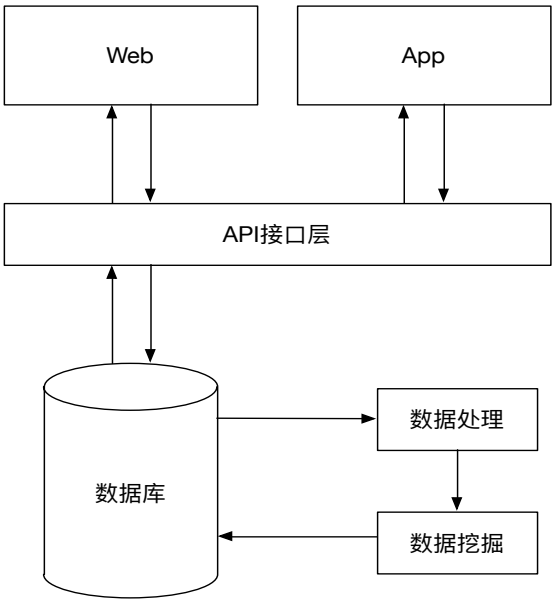


图 4-2 前后端通信架构图

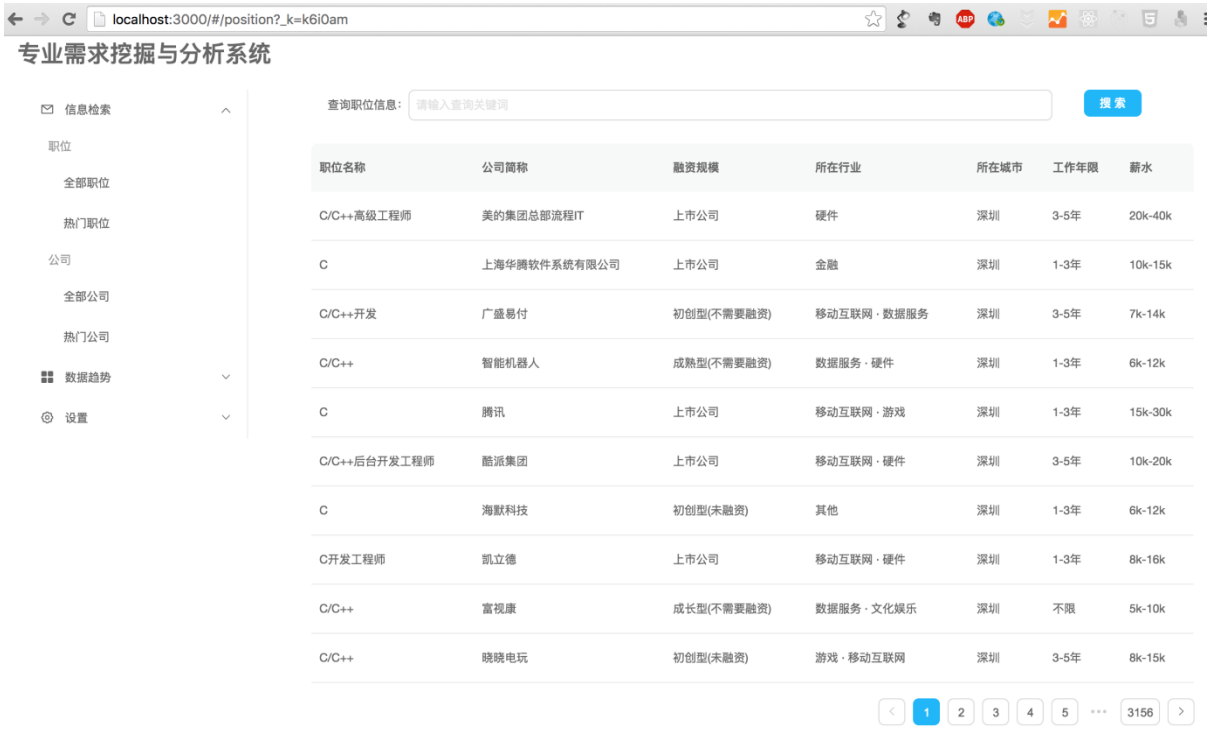


图 4-3 Web 端查看职位信息

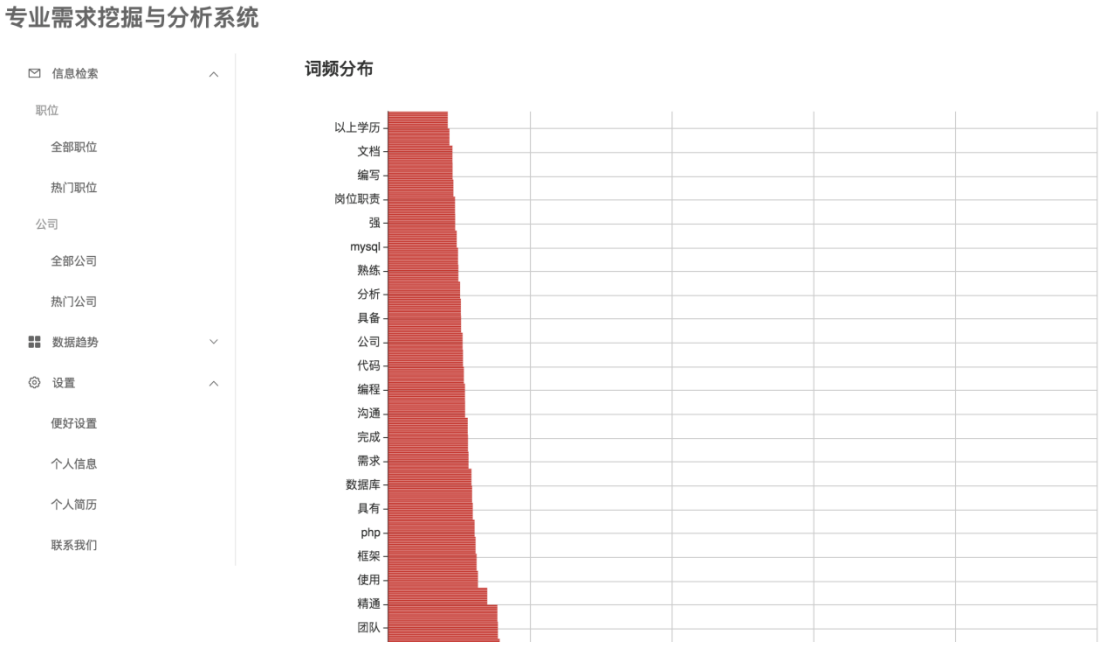


图 4-4 词频统计结果可视化

5 总结与展望

本文描述了定向网络爬虫及其相关数据处理架构设计与实现，爬虫部分使用的是 Python 的 Scrapy 框架，数据库使用的是 MongoDB，中文分词用的 jieba，数据挖掘用的是 scikit-learn 封装的朴素贝叶斯分类器。

网络招聘从上世纪 90 年代末发展到现在已经进入了一个转折点，瞄准细分领域的各类垂直招聘网站开始兴起，现在大量垂直细分领域的招聘数据可以直接从互联网上获取到，随着 Web 技术和大数据技术的飞速发展，垂直领域的定向爬虫配合大数据处理技术再加上 Web 前端数据可视化逐渐成为一种趋势。

最后实现的 IT 专业需求信息挖掘系统分为三个大部分：

1. 爬虫系统：分为数据爬取部分和数据存储部分，将招聘网站上的数据爬取并存储进数据库
2. 数据处理分析系统：分为分词系统和模式匹配系统，对职位需求信息进行分词和关键词提取，并利用关键词进行数据建模并利用建立的模型进行相关性匹配
3. 前端数据可视化系统：分为后端 API 部分和前端可视化组件部分，实现了后端 Restful 风格 API 接口和前端组件化分离，目前可以按条件展示爬取到的数据并显示出关键词分布的图表

但是由于时间以及个人能力所限，本系统在分词及关键词提取部分做的不是很好，用于匹配的数据模型也过于简单，数据爬取的速度也有待优化。总之，最然此次毕业设计没能完美的实现预期的效果，但是对现有的技术整合与探索依然是一次不错的尝试。随着大数据技术的发展，利用不断积累的数据信息，从海量工作岗位信息中快速找到最符合自己的工作一定会变得越来越容易，从中分析热门技术的趋势也会越来越方便。

参考文献

[1]刘为怀, 才华, 何东杰. 一种基于中文分词和数据聚合的餐饮行为特征挖掘方法[J]. 软件产业与工程, 2015(4),47-51

[2] 滕秋霞,杨金霄,方永佳. 基于投票混合模型的中文地址分词研究[J]. 工业控制计算机,2015(11),105-106

[3] 胡伟伟,孙逊,王婷婷. 基于向量空间模型的项目申报书查重系统设计[J]. 天津科技, 2015(8), 33-34

[4] 汪红林,王红玲,周国栋. 语义分析中谓词标识的特征工程[J]. 计算机工程与应用,2010(9),134-137

[5] 耿升华. 新词识别和热词排名方法研究[D]. 重庆:重庆大学,2013.

[6] 梁艳. 基于案例推理的职位推荐[D]. 石家庄:河北师范大学, 2013.

[7] 闻剑峰,石屹嵘. 以分布式计算实现电信数据分析业务加速的研究[J]. 电信科学, 2012(2),22-26

[8] 林伟坚. 基于 Scrapy 框架的新闻实时抓取及处理系统的设计与实现[D]. 天津:南开大学, 2012

[9] 杨宇轩. 面向软件专业高校毕业生招聘系统的设计与实现[D]. 石家庄:河北师范大学, 2015

[10] 刘伟光. 开源网络爬虫在垂直搜索引擎应用[J]. 智能计算机与应用, 2015(4), 75-77

[11] 王丹. 基于向量空间模型的中文文本分类技术研究[D]. 沈阳:东北大学, 2006

[12] 潘昊,郑苗,杨俊. 基于就业信息服务的个性化推荐系统设计与应用[D]. 北京:北京邮电大学 , 2015

[13] 陈珊珊. 基于语义的大学生就业推荐系统[D]. 武汉:武汉科技大学, 2014

文献综述

文献［1］介绍了介绍餐饮行为挖掘流程，其次介绍分词、清洗，以及数据聚合以及在聚合数据基础上进行的特征挖掘。

文献［2］介绍了针对现有的中文分词算法在特殊领域的分词性能并不理想的问题，在基于 CRF 分词器的基础上，结合传统的基于字典的分词方法，以及支持向量机（Support Vector Machine，SVM）分词工具，实现了一种基于投票混合模型的地址分词方法，并使用非标准地址数据对该模型进行训练与测试。实验结果表明，在对中文地址数据的分词中，该分词器比几种常用的分词工具具有更好的分词性能，为基于分词的地址数据清洗做了一个重要的基础。

文献［3］、文献［11］介绍了有关向量空间在文本内容的相似度计算中的应用。

文献［4］、文献［5］介绍了有关中文分词和关键词排序相关算法及应用。

文献［6］、文献［12］、文献［13］介绍了有关职位信息的关键信息提取与建模，和在已有模型基础之上的推荐算法，其中文献［12］还简单介绍了服务器接口和移动端 App 的设计与构建。

文献［7］介绍了有关分布式系统的研究与应用。

文献［8］、文献［10］介绍了常见的爬虫系统的架构与实现方法，其中文献［10］介绍了爬虫在搜索引擎中的具体应用。

文献［9］概括介绍了现有的网络招聘平台之间的特点与区别和高等学校招聘系统的设计与实现。

致　谢

准备毕业设计这段时间以来，查了不少资料，而网络爬虫、数据挖掘、模型匹配每一个部分都很费时费力，这里首先要感谢互联网，正是因为有 Google、WikiPedia、Github、Stackoverflow 这样优秀的网站，使得我们非常容易的快速获取到我们想要的知识，感谢那些一直以来贡献代码的开源工作者们，没有他们长期以来的工作就不会有今天繁荣共享的互联网生态。最后，要感谢我的指导老师鲁丽老师，她是我见过最为负责任的老师之一，在我准备毕业设计的过程过给予了我非常多的指导与帮助。