大数据环境下信息抽取模板自动聚类与发现

计 92 丘骏鹏 2009011282

指导老师：朱小燕 郝宇

提纲

选题回顾

系统设计与实现

初步结果

问题及后期工作

提纲

背景

- 已经获取到海量的新闻、博客、论坛等网页原始数据，需要从中提取结构化的信息
- 从已有数据中抽取模板，利用模板去抽取相似网页中的信息。
- 新的网页可能采取新的模板，需要自动检测，分类和抽取这些新的模板
- 目标：提取结构化信息，如新闻中的标题和正文，博客的标题和内容等，用于后续的处理。

```
<document>
  <news>
    <title>foobar</title>
    <content>blablabla</content>
  </news>
</document>
```

输入

- 文档集合

```
<html>                    |  <html>
  <body>                  |    <body>
    <h1>Title1</h1>       |      <h1>Title2</h1>
    <p>Content1</p>       |      <p>Content2</p>
  </body>                 |    </body>
</html>                   |  </html>
--------------------------|--------------------------
<html>                    |  <html>
  <body>                  |    <body>
    <div>                 |      <div>
      <div>foo1</div>     |        <div>foo2</div>
      <div>bar1</div>     |        <div>bar2</div>
    </div>                |      </div>
  </body>                 |    </body>
</html>                   |  </html>
```

输出

- 抽取的模板 1

```
<html>
  <body>
    <h1>?</h1>
    <p>?</p>
  </body>
</html>
```

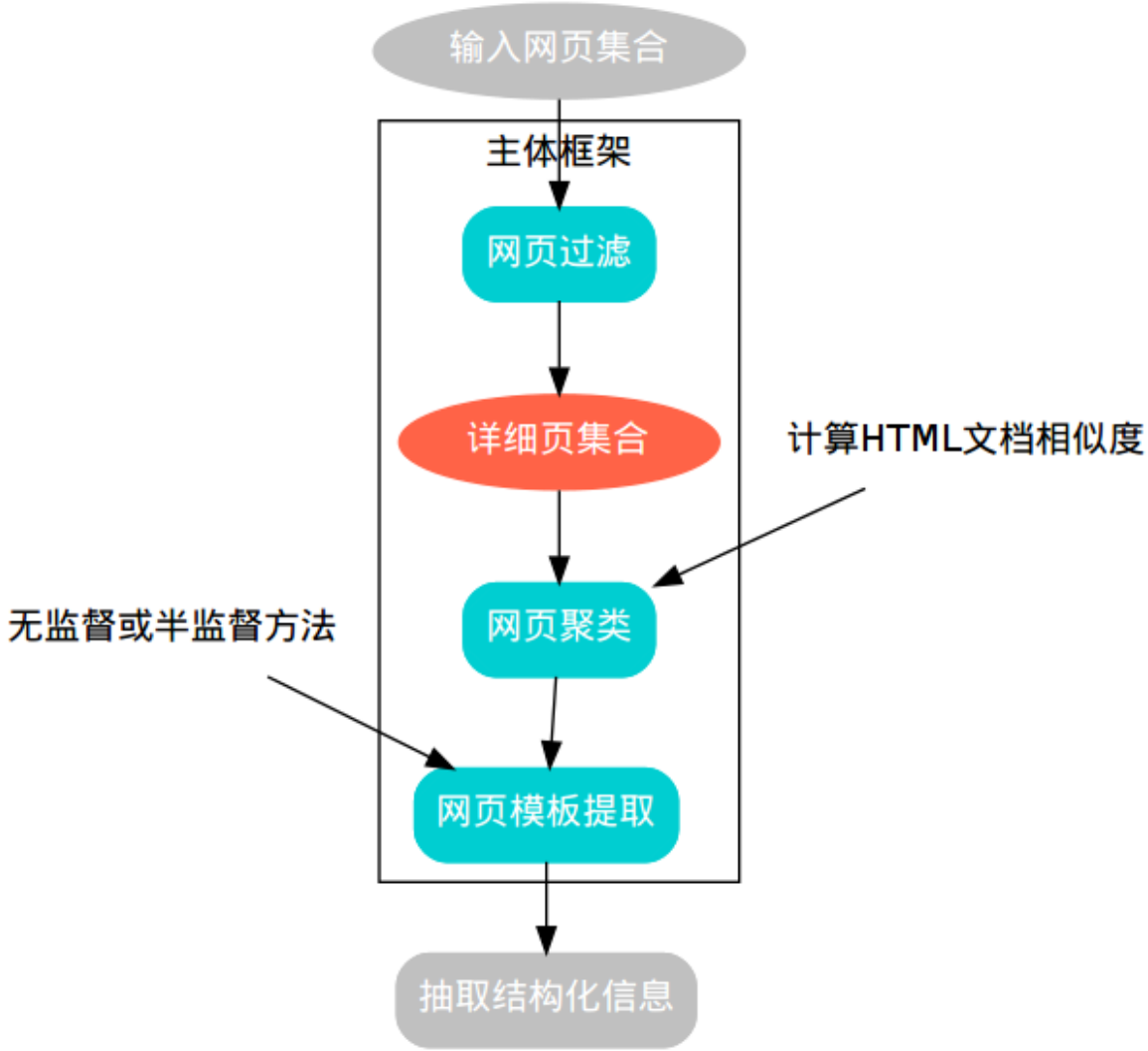
- 抽取的模板 2

```
<html>
  <body>
    <div>
      <div>?</div>
      <div>?</div>
    </div>
  </body>
</html>
```

提纲

框架

整体框架示意图


输入网页集合
主体框架
网页过滤
详细页集合
计算HTML文档相似度
无监督或半监督方法
网页聚类
网页模板提取
抽取结构化信息

系统实现概况

- 实现语言

  Java+Scala。Scala 是 JVM 上的静态类型语言，可以和 Java 之间无缝操作，支持面向对象和函数式等编程范式。
- 使用的第三方库
  1. icu4j：用于检测网页字符编码
  2. Jsoup：HTML Parser。可以自定义 Visitor 来访问树的节点。
  3. util-logging：twitter 包装的 java.util.logging 库，用于日志系统
  4. typesafe's config：完成配置文件读取
  5. akka：Java & Scala 的 Actor 模型库
- 主要模块实现
  - ☒ 网页过滤
  - ☒ 网页聚类
  - ☐ 模板抽取

系统设计（1）

- 实验数据统计

	blog	news	other
文件个数	59998	81561	183635
总大小	5.4G	7.9G	18G

前期主要针对 blog 数据做了一些实验

系统设计（2）

- 过滤目录页
  - 新浪博客的目录页和详细页可以用 URL 区分。比如某个博主的目录页为

    `http://blog.sina.com.cn/u/1439351555`

    他的某篇文章的 URL 格式为

    `http://blog.sina.com.cn/s/blog_55cac30301016yb1.html`

    因此对于博客数据可以用 URL 正则进行过滤
  - blog 中不带 html 后缀的文件有 23430，带有 html 后缀的文件有 36568。过滤出 blog 数据中有用的详细页数据为 36568。

系统设计（3）

- 预处理
  接下来我们需要对详细页进行预处理。
  - 去除空行
  - 去除无用标签

    `<script>`, `<link>`, `<style>`, `<br>`, `<img>`, `<em>`
  - 去除标签属性值，加速 Dom Tree 的构建速度
  - 去除文本 `<#text>` 和 `CDATA` 数据，只保留结构化的标签名
  - 将树结构平坦化：遍历 Dom Tree 得到 TreeNode 序列，在每个 TreeNode 保存节点的名字和深度

系统设计（4）

- 近似度计算
  为了方便计算以及后续的模板的抽取，采用 LCS 作为计算相似度的基础

$$c(i)(j) = \begin{cases} 0 & i = 0,\ j = 0 \\ c(i-1)(j-1) + 1 & i,\ j > 0, x_i = y_j \\ \max(c(i)(j-1), c(i-1)(j)) & i, j > 0,\ x_i \neq y_j \end{cases}$$

- Longest Common Tag Subsequence

$$d_{LCTS}(D_1, D_2) = 1 - \frac{|lcts(D_1, D_2)|}{\max(|D_1|, |D_2|)}$$

系统优化

1. LCS 的动态规划算法的时间复杂度为 O(mn)，空间复杂度也是 O(mn)，需要做一定的优化。
2. 假设每运行一次算法的时间为 t，以最小的文档集合 blog 为输入（数目约为 60000 篇），则计算文档集合中两两之间距离的总时间约为：

$$\frac{60000^2}{2 * 3600} * t = 5 * 10^5 * t$$

   取$t = 0.001s$，则总时间为$5 * 10^5 * 0.001 = 500h$。
3. 文档数很大，运行时需载入内存，也需要尽量减少空间复杂度

优化内存

- 动态规划原理式

$$c(i)(j) = \begin{cases} 0 & i = 0,\ j = 0 \\ c(i-1)(j-1) + 1 & i,\ j > 0, x_i = y_j \\ \max(c(i)(j-1), c(i-1)(j)) & i, j > 0,\ x_i \neq y_j \end{cases}$$

以行优先遍历为例：实际上我们在计算每一个点的值时，依赖的信息只包括这一行之前已计算出的点和前一行的点，所以只需要两个一维数组即可。空间复杂度降低为 `O(n)` 。

优化时间

- 本身 HTML 文档的表示方式是有冗余的。如果每个 HTML 标签都有对应的开始和结束标签，那么每一对这样的标签 `<tagName></tagname>` 可以用 `(tagName, depth)` 来表示。
- 类比：S-expression 。可以将严格的 XML 转换为 S-expression 而无信息丢失，因此只需要记录当前括号包含的第一个符号的名字（对应与 tagName）和嵌套层数（对应于深度）

```
<html>                                  <=>  (html
 <body>                                 <=>    (body
  <div>                                 <=>      (div
   <p></p></div></body></html>          <=>        (p))))
```

- 用途：减少 tag 序列长度，只保留开始 tag 和深度即可。大部分的标签都是成对的，因此这样大概可以减少一半的 tag 序列长度。
- 标签名 hash：直接比较计算好标签的 hash 值，避免字符串比较

优化计算方式（1）

- 在以上优化的基础上，两两之间进行一次计算需要的时间为$0.001 \sim 0.002s$。
- 之前已经计算过，在$t = 0.001s$ 的情况下，计算一次 blog 集合中所有文档相互之间的距离需要 500 小时。
- 优化计算方式：采用多线程进行计算。

优化计算方式（2）

- 采用 Actor 库进行实现
  - 一种并行计算的模型，与上世纪 70 年代提出
  - 在 Actor 模型里，每个 Actor 是完全独立的，相互间采用异步、非阻塞的消息传递进行通信
  - 比多线程的优点：可以避免使用全局状态、锁、信号量等一些低级的同步原语；有封装好的线程调度算法，不需要手动对线程进行管理，简化任务的分割。

优化计算方式（3）

- 具体实现
  - 计算一个文档集合两两之间的相似度相当于计算一个正方形的上半部分的三角形区域。
  - 将该区域用等距的横线和纵线分割，然后将这些区域通过调度器分发给每个可用的 Actor 进行计算。调度算法采用简单的 Round-Robin。

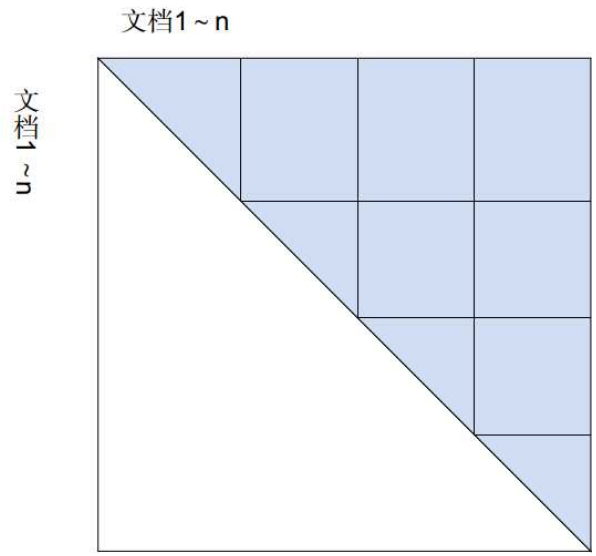


Figure: 示意图

聚类算法

- 实现了一个最简单的层次聚类算法
- 过程
  - 每个文档开始时单独为一类
  - 选择中心点距离最近的两个类进行合并
  - 更新类中心点：选择距离其他点距离之和最小的点作为类的新中心点，重复以上过程
- 算法特点
  - 只需要计算一次文档集合相互之间的相似度
  - 阈值较难设置

提纲

初步实验概况

- 目前只在实验室的电脑上进行过实验，机器配置为 16 个逻辑 CPU+24G 内存，尚未部署到 Hadoop 上。
- 由于以上限制，目前小数据量上做过实验：从 blog 中抽取出了 1000 个文档作为文档集合
- 计算相似度时间：约 100s
- 聚类所需时间：约 10s

初步实验结果

- 阈值为 0.1 时，聚成 71 类，效果很差
- 阈值为 0.5 时，聚成 8 类；同一博主的文章大部分被分成同一类，但也有分错
- 阈值为 0.7 时，聚成 2 类；通过手工检查，两个类别仍有分错现象
- 阈值为 0.9 时，聚成 1 类；比较符合目前手工检查的结果

初步分析

- 由于目前只能人工一个个检查实验结果进行评价，还缺乏比较系统可行的评价指标，以上结果也并不是非常全面
- 阈值难以设置，目前需人工修改，并不断检查来判断好坏
- 从聚类的结果来看，目前采用 LCS 的算法计算相似度效果不算非常好，在阈值为 0.7 时聚成 2 类时有相似网页聚类错误的情况

提纲

改进已有工作

- 需部署到 Hadoop 上对更大规模的文档集合进行计算
- 由于计算相似度做了大量近似，需要进一步评价实际效果，对分错的个例进行分析
- 目前简单的聚类算法需要进行改进

完成剩余模块

- 利用计算出的公共字串及 tag 的深度信息反向构建出树结构，作为该类的模板
- 对新的文档进行分类，利用该类的模板抽取文档内容
- 或者新的文档成为新的类，计算新的模板
- 优化系统的运行效率