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

系统优化

初步结果

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
- 做法：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。
- 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，存储成以下格式，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>
```

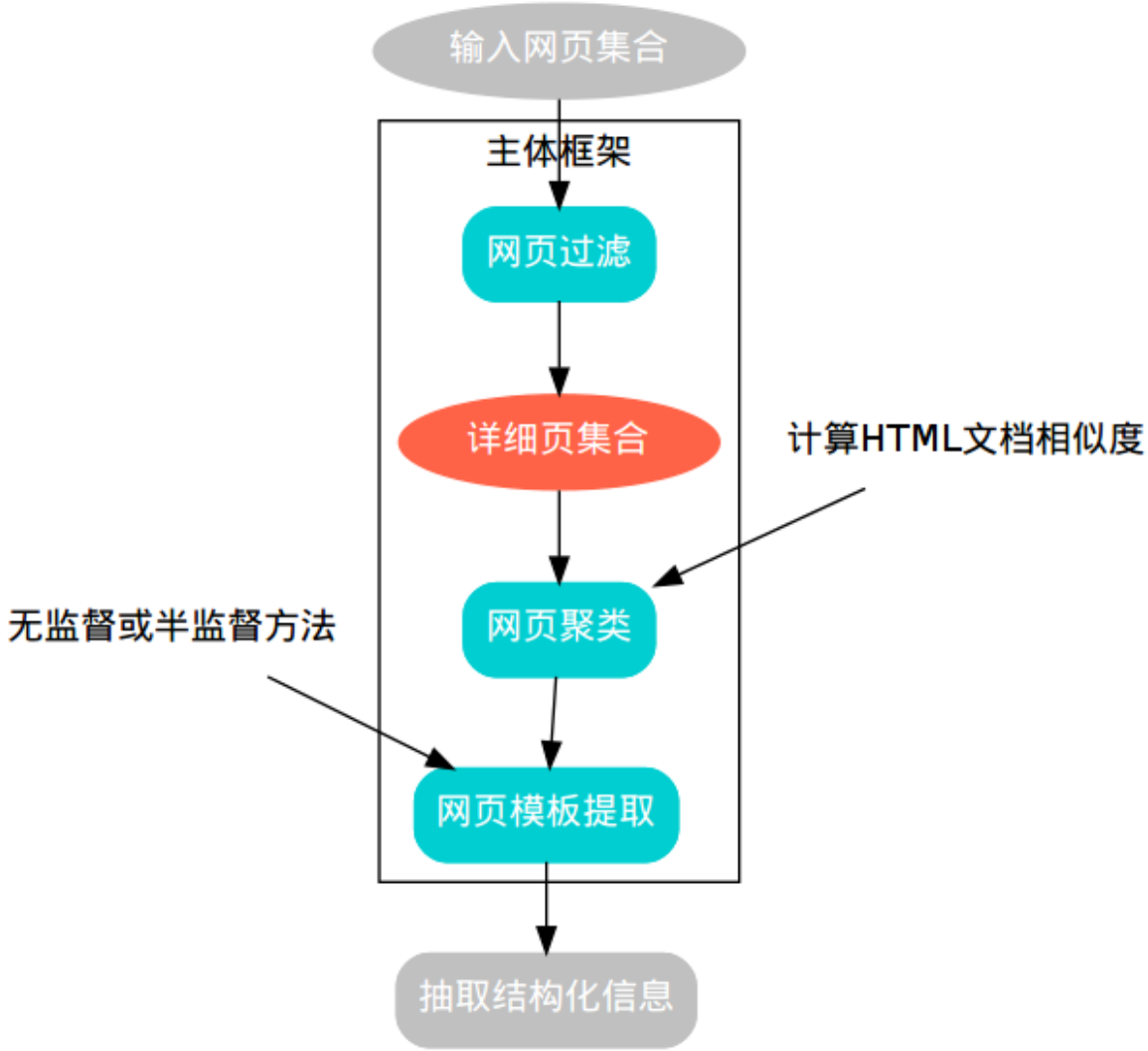
- 抽取的模板 2

```
<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

- 主要模块实现
  - ☒ 整体框架搭建
  - ☒ 网页过滤
  - ☒ 网页聚类
  - ☐ 模板抽取
- 进度对比
  - 开题报告
    - 5-8 周：网页过滤，网页聚类和网页模板提取模块初步实现
    - 9-12 周：算法修正，系统改进，结果分析
  - 实际进度
    - 5-8 周：初步的模板提取模块尚未完全实现
    - 9-12 周：由于计算速度瓶颈，目前已进行了算法的优化

搭建系统框架

- 实现语言：Java+Scala
- 考虑到代码的重用性，采用了许多工业界广泛应用的第三方库：
  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

实验数据

- 实验数据统计

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

主要针对 blog 数据做了一些实验

系统设计（1）

- 网页过滤模块
  - 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 文档集合中目录页文件数为 23430，详细页文件数为 36568。

系统设计（2）

- 预处理
  - 去除空行、标签属性值、文本 <#text> 和 CDATA 数据以及无用标签

    <script>, <link>, <style>,
, <img>,

  - 将树结构平坦化，降低计算复杂度。通过前序遍历 Dom Tree 得到 tag 序列：

```
<html>
  <body>
    <div>
      <p></p>
      <a></a>
    </div>
    <div></div>
  </body>
</html>
```

    转化成:

```
<html><body><div><p></p><a></a></div><div></div></body></html>
```

系统设计（3）

- 相似度计算
  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

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

重复记录的处理

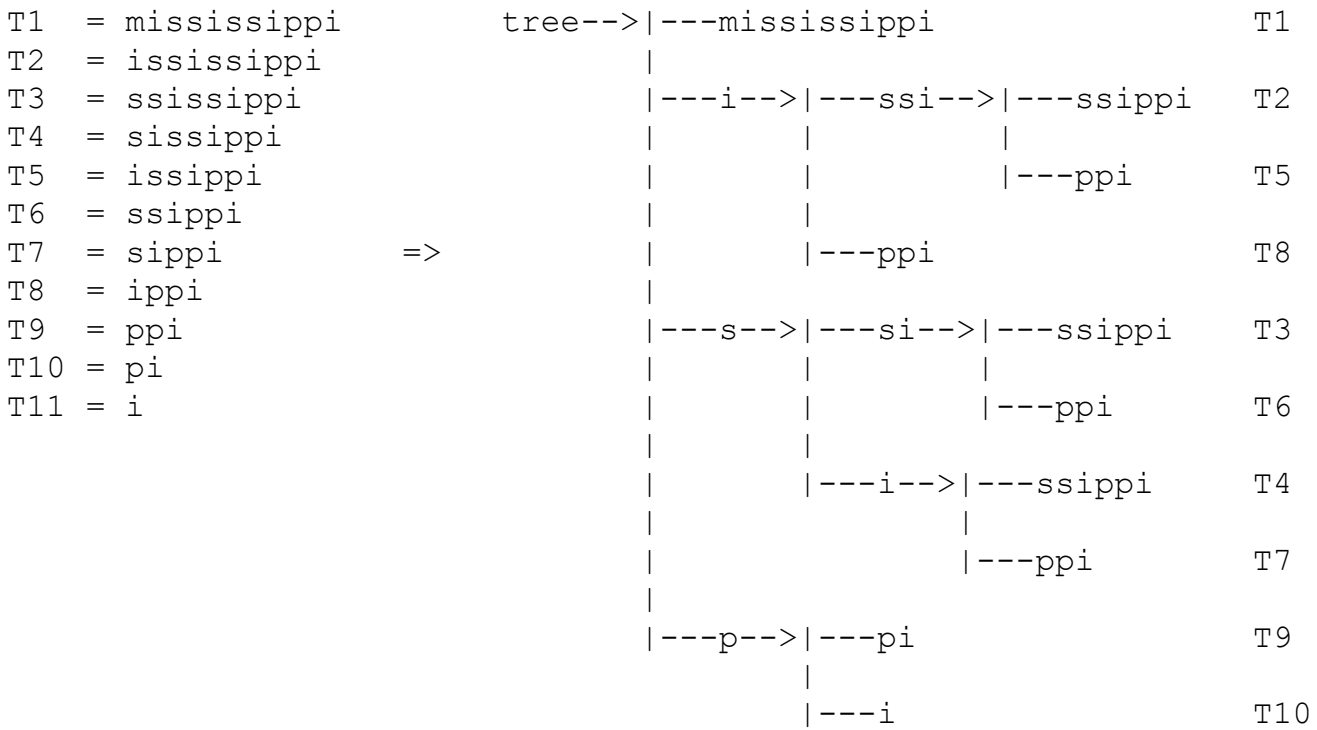
- 网页中含有部分重复元素，这些部分是由网站后台动态生成的 (Python Django)：

```
{% for file in file_list %}
  <li>{{file}}</li>
{% endfor %}
```

- 简单的方法：去掉这些标签，在比较过程中不予考虑。但这种方法无法处理更复杂的情况。如一个 标签下的子树（Data Record）。
- 后缀树（SuffixTree）
  - Trie 的一个变种，可以快速找到字符串中的重复子串
  - 快速算法可以在$O(n)$ 时间内构建:
    Ukkonen, Esko. "On-line construction of suffix trees." Algorithmica 14.3 (1995): 249-260.
- 由于采用前序遍历，可以保证子树在序列上是连续的

后缀树

- 对于字符串 mississippi：



- 任一到内部节点的路径都是字符串中重复的子串

聚类算法

- 实现了一个简单的层次聚类算法
- 过程
  - 每个文档开始时单独为一类，并作为该类的中心点
  - 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

动机

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$O(mn)$
2. 文档数很大，运行时需载入内存，需要尽量减少空间复杂度
3. 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$5 * 10^5 * 0.001 = 500h$。

优化空间

- 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

- 减小运行时间的有效办法是压缩 tag 序列长度。
  1. 预处理已经去掉了很多无用标签
  2. `<tagName></tagname>` 可以用 `(tagName, depth)` 来表示, 相当于将 HTML 转换为 S-expression。

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

- 用途：由于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。
- 标签比较时加入深度信息：

```
node1.tagName = node2.tagName && node1.depth = node2.depth
```

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，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
  - 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。
- 具体实现
  - 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。

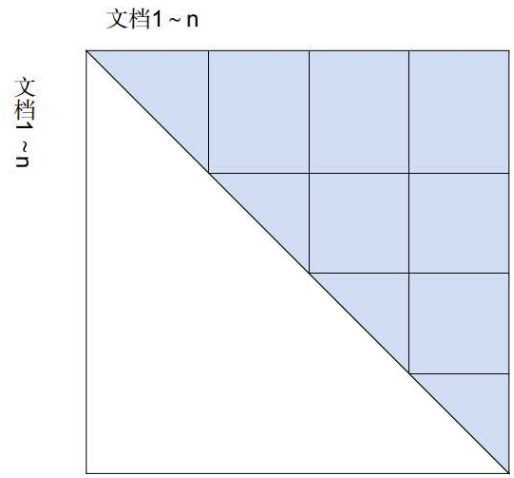


Figure: 示意图

优化距离计算

- 考虑深度的影响，离根节点越近的标签权重越大
- 修改 LCS 算法，$f(x)$ 是一个与当前节点深度x 有关的函数：

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

- 同时修改距离计算公式

$$d_{LCTS}(D_1, D_2) = 1 - \frac{|lcts(D_1, D_2)|}{\max(\sum_{n \in D_1} f(n.depth), \sum_{n \in D_2} f(n.depth))}$$

提纲

实验设置

- 在实验室的服务器上进行实验，机器配置为 16 个逻辑 CPU+24G 内存
- 由于以上限制，目前在小数据量上做实验：从 blog 中抽取出了 1000 个文档作为实验的文档集合

实验结果

- 直接聚类：在阈值为 0.3 的情况下，所有文档聚成一类，通过手工可以大致确定正确性
- 验证可行性：加入噪音
  - 在详细页文档中加入目录页、404 错误页等噪音
  - 聚类结果：在阈值为 0.3 的情况下，文档被聚成 3 类，分别是详细页，目录页和 404 错误页
- 初步分析：
  - 阈值设置：目前暂无法确定合适阈值将两个文档分为不同的两类
  - 评价：需要根据最后的模板抽取结果来判断

提纲

后期工作目标

- 实现模板抽取
  - 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
  - 采取少量标注进行半监督学习
- 新文档分类
  - 归为已有的一类，利用该类的模板抽取文档内容
  - 归为新的类，计算新的模板
- 结果评价
  - 根据模板抽取结果，调整实验参数