
cute-jumper / BachelorThesis

Branch: master ▾ BachelorThesis / Reports / final / final\_report.org

Find fileCopy path

cute-jumper change Makefile and structure. Add final report.

147e040 on 13 Jun 2013

1 contributor

446 lines (438 sloc)19.3 KB

# 大数据环境下信息抽取模板自动聚类与发现

\begin{frame}<beamer>\frametitle{提纲}\tableofcontents\end{frame}

## 研究内容介绍

### 背景

- “大数据”时代来临，我们获得的数据越来越多，研究工作也受到了新的挑战
- 为了便于计算机处理这些数据，需要从非结构化和半结构化数据中提取出我们关心的内容，存储成结构化的数据
- 大部分网页是通过查询后台数据库，然后选择合适模板进行渲染方式生成的，因此可以通过模板来进行内容的提取。

```
<html>
<body>
  <h1>{{ news.title }}</h1>
  <p>{{ news.content }}</p>
  <div>
    {% for comment in news.comments %}
      <li>{{ comment }}</li>
    {% endfor %}
  </div>
</body>
</html>
```

### 工作内容和特点

#### 工作内容

- 已经获得大量新闻、博客等网页数据，这些网页可能由不同的模板生成，我们从中提取出所有可能的模板
- 根据提取出的模板，抽取新的网页中的有用的信息

#### 特点

- 由于我们的数据量比较大，因此需要设计一些高效的算法
- 海量数据提供了较多的冗余性，弥补算法的一些不精确性

## 系统框架与模块实现

### 整体框架

[./framework.pdf](#)

### 预处理模块-过滤无用网页

- 404等错误页大小一般比较小，可以通过设置文件大小阈值过滤
- 目录页的URL有一定的模式，可以用为每类网页分别建立正则表达式的方法过滤目录页。比如对于新浪博客来说,详细页的URL都以.html结尾,而目录页则没有。以下两个URL: \scriptsize

```
http://blog.sina.com.cn/u/1439351555
http://blog.sina.com.cn/s/blog_55cac30301016yb1.html
```

\normalsize 第一个URL对应的是某个博主的文章目录,第二个URL则是该博主的某篇文章。

## 预处理模块-过滤无用网页

- 为了完备性考虑,可能存在某些网站,目录页的 URL 和详细页的 URL 在模式没有太大的区别,或者人工设置规则的办法太麻烦了
- 目录页的 HTML 文档的主要部分是由列表环境 `<ul>` 或 `<ol>` 中的列表项 `<li>` 及其包含的超链接标签 `<a>` 组成。我们提取出网页中的所有 `<li>` 标签中的 `<a>` 标签,计算这些 `<a>` 标签的文本内容占网页所有文本内容的比重,超过一定的阈值则判定为目录页。

```
<ul>
  <li><a>红会社监委今日讨论是否重启调查郭美美事件</a></li>
  <li><a>北京今日14时起堵车 中度以上拥堵将持续7小时</a></li>
  <li><a>湖北黄冈拒卖地反投8亿建免费公园 市长称不亏</a></li>
  <li><a>湖南株洲市档案局工会主席杀害局长后跳楼</a></li>
  <li><a>陕西神木集资大王被查 结婚时曾空运20辆林肯</a></li>
  <li><a>内地学生扎堆报京沪港高校 在校平均恋爱0.78次</a></li>
</ul>
```

## 预处理模块-简化HTML文档

### 去除无用标签

- 与网页模块无关。去除这些标签时,可以将标签本身及标签的内容一同去掉。比如 `<script>`、`<link>`、`<style>` 等。
- 通常是深层次的 HTML 文档的节点,用于控制格式,在模板中意义不大。去除这些标签时只取出标签本身,保留标签所包含的内容。比如 `<br/>`、`<p>`、`<strong>`、`<em>` 等。
- 我们在实验中不考虑非文本标签,因此将 `<img>`、`<audio>` 等直接去除。
- 最后是去除一些在模板中变化很大的太复杂的标签,典型的是表格相关的一些标签,包括 `<table>`、`<th>`、`<tr>`、`<td>` 等。

## 预处理模块-简化HTML文档

### 对解析好的 DOM Tree 进行简化

- 树形结构上做相关的操作较为复杂,我们首先将树形结构转化为更便于处理的序列形式
- 采用先序遍历的方式,将 DOM Tree 转化成一个标签序列,在遍历的同时在每个节点处保存了该节点的深度信息
- 每个标签采用 标签名 + 深度 的表示方法,下图可以表示为: `<html0><body1><div2><a3><p3><div2>`

[./domtree.pdf](#)

## 预处理模块-简化HTML文档

### 检测重复记录-后缀树简介

- 网页模板中有不定个数的重复模式,通常由模板语言的for语句生成,我们需要将这些记录合并
- 后缀树是一种高效的数据结构,可以快速完成重复字符串的查找。
- 后缀树定义: 由序列所有的后缀组成的Trie树。后缀树的每一条边都代表着一个序列,从根节点到后缀树叶子节点的每条路径都对应着原序列的一个后缀。
- 后缀树普通的构造算法复杂度很高,系统实现中采用了Ukkonen在1995年提出了一个  $O(n)$  时间复杂度的在线构造算法。

## 后缀树示例

BANANA对应的的后缀树  $\begin{figure}[hb] \centering \includegraphics[width=0.5\textwidth]{./suffix-tree-banana.png} \end{figure}$

## 后缀树查找重复序列算法

### 原始算法

- 任意一条从根节点到内部节点的路径组成的序列都是原序列中重复出现的字串,且重复的次 数是以该内部节点为根的子树的 叶子节点个数。
- 不能处理：不同的重复子串之间可包含或者相交的关系；同一个重复子串在原序列上有交 集

### 新的算法要求

- 重复序列不能横跨两个子树
- 重复序列必须有公共的父亲
- 重复序列必须尽可能地长

## 后缀树查找重复序列算法

```
\begin{algorithm}[H] \caption{从根节点出发，找出所有的重复子序列\label{suffixtree:algo:fromroot}} \begin{algorithmic}[1]
\Require 已经构建好的后缀树，根为$\textit{root}$ \Ensure 该后缀树中所有的重复子序列 \State $//$从根节点出发，寻找所有的重复
子序列 \For{$\textit{edge} \leftarrow \textit{root.edges} \sim \mathbf{if} \sim \textit{edge.endNode.isNotLeaf}$} \State $//$取后缀树根节点的每条边的第一个元素作
为每个子树的根节点 \State $\textit{subTreeRoot} := \textit{edge.firstElement}$ \State $//$查找以该节点为根的所有重复子序列 \State
\textit{findAllRepetitions}$(\textit{root}, \mathbf{nil}, \textit{subTreeRoot})$ \EndFor \end{algorithmic} \end{algorithm}
```

## 后缀树查找重复序列算法

```
\floatname{algorithm}{\tiny 算法} \begin{algorithm}[H] \caption{\tiny 简化的findAllRepetitions实现}
\label{suffixtree:algo:findrep} \begin{algorithmic} \tiny \Require 一个内部节点$\textit{node}$，当前已经找到的重复序列$\textit{prefix}$，要
找的子树的根节点$\textit{subTreeRoot}$ \Ensure 所有经过该内部节点的符合要求的重复序列 \Function {findAllRepetitions}{$\textit{node}$,
$\textit{prefix}$, $\textit{subTreeRoot}$} \State $//$定义一个空集合 \State $\textit{results} := \text{Collection.empty}$ \State $//$对于该内部节点的每一条不连
接叶子节点的边 \For{$\textit{edge} \leftarrow \textit{node.edges} \sim \mathbf{if} \sim \textit{edge.endNode.isNotLeaf}$} \State $//$依次取出该条边上属于该根节
点子树上的点 \State $\textit{seq} := \textit{edge.takeWhile}(\textit{element} \in \textit{subTreeOf} \textit{subTreeRoot})$ \label{suffixtree:code:equals} \If
{$\textit{seq.length} == \textit{edge.length}$} \State $//$遍历完了该条边上所有元素， \State $//$则取该条边连接的下一个内部节点进行递归
查找 \State \textit{findAllRepetitions}$(\textit{edge.endNode}, \textit{prefix} + \textit{seq}, \textit{subTreeRoot})$ \Else \State $//$否则，将当前得到的序列加入到结
果集合中 \State \textit{addToResults}$(\textit{prefix} + \textit{seq}, \textit{results})$ \label{suffixtree:code:add} \EndIf \EndFor \State \Return{$\textit{results}$}
\EndFunction \State \end{algorithmic} \end{algorithm}
```

## 合并重复记录

- 所有序列按实际的父节点分组，然后对每个分组进行合并，保证重复串有公共的父亲。
- 但是仍然存在序列相交的情况，如  $\langle \text{div3} \rangle \langle \text{div4} \rangle \langle \text{div3} \rangle \langle \text{div4} \rangle \langle \text{a5} \rangle \langle \text{img5} \rangle \langle \text{div4} \rangle \langle \text{a5} \rangle \langle \text{img5} \rangle$  和  $\langle \text{div4} \rangle$  和  $\langle \text{div4} \rangle \langle \text{a5} \rangle \langle \text{img5} \rangle$  都是符合算法要求的重复串，但互相之间有 交集  $\langle \text{div4} \rangle$  。我们采取的策略是只取更深的重 复序列。这里只取  $\langle \text{div4} \rangle \langle \text{a5} \rangle \langle \text{img5} \rangle$

## 网页聚类模块-计算结构相似度

- 序列  $S_1, S_2$  ,  $x_i$  和  $y_j$  分别表示  $S_1$  的第  $i$  个元素和  $S_2$  的第  $j$  个元素，  $f(\textit{depth})$  是根据深度加权的 函数，我们认为深度越深的节点，成为模板 的概率就越小。 
$$t(i)(j) = \begin{cases} 0 & i = 0, j = 0 \\ t(i-1)(j-1) + f(x_i.\textit{depth}) & i, j > 0, x_i = y_j \\ \max(t(i)(j-1), t(i-1)(j)) & i, j > 0, x_i \neq y_j \end{cases}$$
- 结构相似度计算公式 
$$\text{Sim}(D_1, D_2) = \frac{|\text{elcs}(S_1, S_2)|}{\max(\sum\limits_{n \in S_1} \{f(n.\textit{depth})\}, \sum\limits_{n \in S_2} \{f(n.\textit{depth})\})}$$

## 计算时间优化

- 文档数量多，计算量很大，需要一定的优化。
- 采用Actor库实现多线程计算，加快计算速度。将任务分割后交给每个Actor进行计算， Actor的调度的算法采用 RoundRobin。
- 任务分割示意图

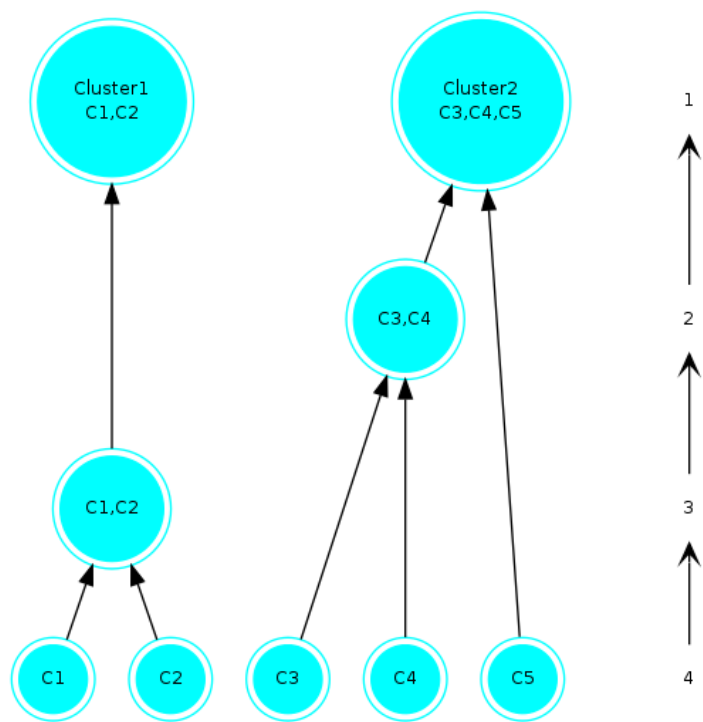
## 聚类算法

1. 初始时，让每个文档实例都单独为一类

2. 迭代时，每次选择距离最近的两个类合并

3. 直到任意两个类的距离都大于阈值时程序退出

4. 算法的结束条件由阈值决定，无需实现设定类的个数。



## 模板形式化定义

### 基本节点

- 两种组成方式

i. 单个不重复的HTML标签，即  $\langle \text{tag} \rangle$

ii. 由一个或多个HTML标签组成的序列，这些序列可以出现一次或多次，即  $(\sum_{i=1}^N \langle \text{tag}_i \rangle)^+$ ，其中  $N \geq 1$

• 第二种形式通过合并重复记录得到，对应的模板语言为：

• 还有一种模板生成形式：

## 模板形式化定义

### 必选和可选节点

- 必选节点  $EN$  对应着由基本节点组成的一个序列，若模板节点用  $tn_i$  表示，则必选节点可以表示为  $\backslash [ EN=tn_1tn_2...tn_n ] \backslash$

• 可选节点  $ON$  同时对应多个序列，每个序列由不同的基本节点组成，同时每个序列 还对应着一个出现概率  $p_k$ ，即： $\backslash [ ON=tn_{11}tn_{12}...tn_{1n_1},p_1|tn_{21}tn_{22}...tn_{2n_2},p_2|...|tn_{k1}tn_{k2}...tn_{kn_k},p_k ] \backslash$

• 定义模板  $TP$  必须由这两种节点交替组成： $\backslash [ TP=[ON_0]EN_1ON_1EN_2ON_2.....EN_n[ON_n] ] \backslash$

## 模板生成算法

- 找出所有组成必选节点的基本节点：对于一个聚类的全部序列，从聚类中心点开始，依次计算一次最长公共子序列，得到  $n$  个序列的公共子序列  $S_{common}$

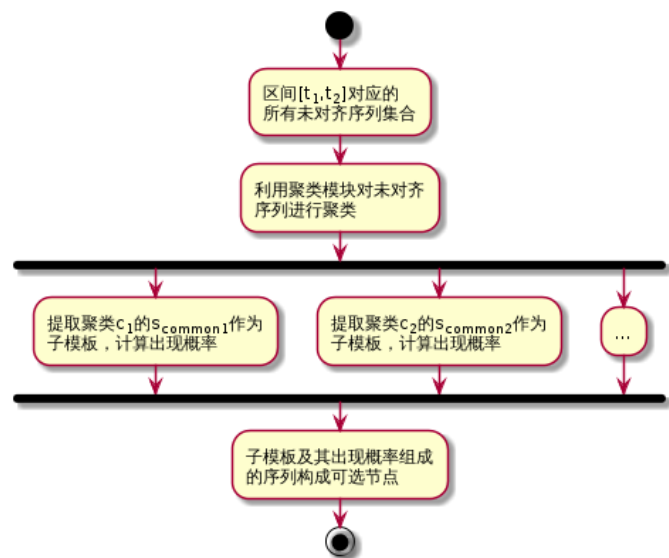
• 每个序列和  $S_{common}$  对齐，得到一些未对齐的区间，每个未对齐的区间计算一个 可选节点

• 已对齐的基本节点组成必选节点

• 必选节点和可选节点交替出现，组成最终的模板

## 模板生成流程图

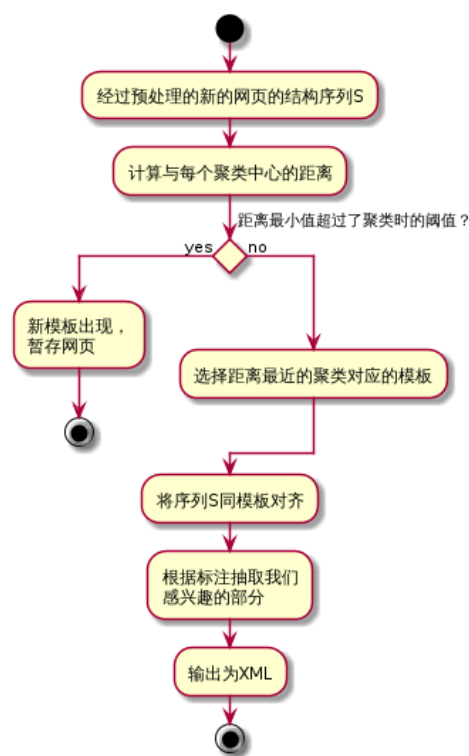




## 内容提取模块

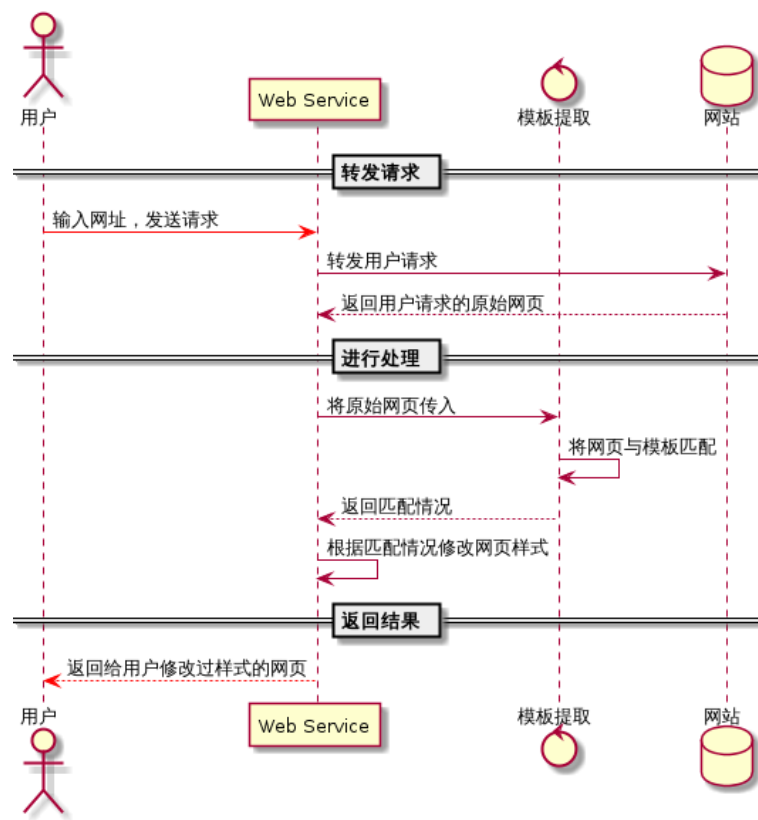
### 内容提取的过程

1. 针对每个模板做一些简单的人工标注，如新闻网页模板的标题，正文，评论等
2. 判断新输入的网页属于哪个聚类，选择对应的模板
3. 模板与网页的序列对齐，根据标注抽取我们感兴趣的部分



## 模板匹配演示系统

- 基于Play! Framework(Scala)实现了一个Web Service



## 实验和分析

### 实验环境和数据

- 机器的配置情况是：16个逻辑核的CPU，24G内存，操作系统为64位的CentOS。
- 数据统计

### 预处理模块

- 过滤目录页的规则
- 训练集和测试集分割

### 预处理模块

#### 无用标签去除规则

#### 检测出的重复记录统计

### 聚类 and 模板提取

- 选择news数据集做实验，调整聚类时的阈值，观察聚类结果
- TODO: 结果分析

### 提取效果

- 我们在测试集上运行我们的程序，抽取出我们需要的信息，并将其保存为XML格式输出。
- TODO: 结果统计和分析

### 演示系统

- 模板匹配演示系统的运行效果如下：

# 总结和展望

---

## 总结

---

1. 设计并实现了一个完整的系统，做了很多优化；实现了较高度的自动化处理，减小人工 参与的工作量。
2. 高效地实现了后缀树这个数据结构，在此基础上设计了一套适用于在树的先序遍历序 列中查找重复子树的算法。
3. 实现并改进了最长公共子序列算法，将其用于计算文档的结构相似度；实现了简单的 凝聚层次聚类算法，并将其用于文档的聚类
4. 实现了一个无监督的模板生成算法，通过人工指定模板中某些部分对应的语义，可以 使用模板提取新的由该模板生成的网页中对应的信息。
5. 实现了一个Web Service，可以直观地看到网页和模板的匹配情况。

## 未来的工作

---

1. 预处理模块：改进Ukkonen原有的在线构造算法，在构造树的时候就考虑一些原本的结构 信息。查找重复时利用后缀树中其他的一些信息，比如后缀链。
2. 网页结构相似度计算和网页聚类模块：可以进一步改进计算相似度的算法，加入一些其他 的除了深度之外的更多结构信息；可以考虑使用更复杂但是鲁棒性更好的一些聚类算法。
3. 模板生成和内容提取：对模板生成算法做一些修改，设计一些更高级的机器学习算法。在 内容提取的时候，可以采用树的匹配算法而不是序列的匹配算法。