实验报告

我做的第 2 题，用的是 SST 算法。
代码寄存在 github 上，开发语言为 Python2.7
Git: https://github.com/superjom/TDA.git
Pos: https://github.com/superjom/TDA/tree/master/pySST

实现内容

本实验实现了 SST 算法，通过扫描一个网页集在内存中维护一个风格树，然后通过统计，将原来网页中的相应正文以及噪音节点通过改变其背景颜色标识出来。

在后期测试期间，本实验自行爬取了 MSN.com 网站 164 个网页进行了处理。算法实现结果主要以直接标注 html 代码的方式返回，可以通过浏览器查看标注后的网页。测试结果表明原生的 SST 算法非常关注新鲜多变的内容作为正文，因此，如果把博客中的回复内容作为正文的话，那么 SST 效果可以，但是如果评论作为噪音的话，SST 效果并不会很好。

程序构成

自己实现的程序主要有三部分：

- 爬虫（reptile）
  单线程的 python 实现的简单爬虫，在爬取的时候，为了防止频繁访问被服务器屏蔽，采用 5-20s 的随机间隔时间进行爬取。爬取的目标网站是 MSN.com，主要考虑的是：
  1. 标签相对规整，降低 html 解析错误
  2. Utf-8 编码，减少 gbk 转码发生的乱码
- SST 算法（sst）
  基本上是论文中算法的实现，感觉算法的复杂度比较高
- 数据可视
  之前跟研二的学长讨论过，应该是要用人工标注的方式通过召回率等指标来验证具体的效果，但由于时间关系，退而求其次。
  为了方便测试，实现了一个简单的数据可视化模块，有两个功能：
  a) 将内存中的 SST 树以 pdf 的格式输出为图形
     采用 graphviz 工具，对于小数据效果非常出色（MSN 的网页支持 3 个，<2000 个节点），但是对于大点的数据失效（pdf 展开面积过大）
  b) 直接对原有的 html 节点用背景色标注
     算法会对 Html 源文件扫描两次，第一次建成 SST 树，第二次根据 sst 树标注 html 中的节点，将识别出的噪音节点背景色改为灰色，对识别出的正文部分背景色标注为蓝色

算法描述

爬虫（reptile）

路径：pySST/reptile/
由三个类组成：
Reptile　　爬虫主程序
Urlist　　判断 URL 是否重复访问
SourceParser　解析 html，抽取出 URL，并且判断筛选出合乎要求的 URL

SST

路径：pySST/sst
Styletree.py　定义了一些风格树相关的数据结构：
　　ElementNode　　普通标签
　　DataNode　　内容标签，包含若干 features，每个 ElementNode 默认有一个 DataNode
　　StyleNode　　风格节点
Sourceparser.py　　解析 html，调用 styletree 建立 SST
dic　　字典模块，在 SST 的 feature 熵计算中，用 md5 为此网页集合每个 feature 建立了一个字典
　　dic.py
　　nodedatas.py　　为每一个节点建立了一个局部的字典，便于统计
总之，一个 ElementNode 默认有一个 DataNode，可以有多个 StyleNode； 一个 StyleNode 可以有多个 ElementNode
初始化：为 body 标签建立一个 ElementNode，将此 ElementNode 及 HTML 树中的 body 节点压入栈，此后循环迭代，pop 出节点对，以深度优先方式便利 HTML 树，同时建立 SST。

数据可视化

doter.py　　将 SST 输出为 dot 文件，用 graphviz 画出树形图
temdec　　模板标注，采用 css 将正文标注为 blue，噪音标注为 gray
graphviz 将 SST 转化为树形图的 pdf 文件，方便小规模数据的测试和调试
temdec 以色块方式直接标注实验数据中的网页，并输出为 HTML 文件

不足

这次实现方面，总体上讲是比较仓促的。 尽管花了不少时间，但是回头看的话，的确走了很多弯路：

1. 最初没有认真看论文，盲目地按照自己的思路，写错了算法
2. 代码没有条理，应该多吸纳一些 OOP 的思想
3. 重复制造轮子，一些功能没有认真寻找现有的解决方案，比如 html 的解析，如果采用

webkit 的内核，应该比目前采用的 pyquery 的容错性高很多

收获

收获总会和挫折一样丰富。

从学术方面：

之前对如何做研究还是比较模糊的，看过的论文，也都是一知半解就草草收尾，而且涵盖领域比较广。这次写算法的这些弯路，确实让我对自己的能力有了客观的认识。

至少认识到人的精力总是有限的，只有充分认识到这一点，才能够知道自己什么应该做，什么不应该做。对于学术也是这样，各个领域浅尝则止是不会有深入认识的。至少彻底了解的话，需要去实现一个算法，这个时间就已经比较可观了。 需要戒骄戒躁。

从动手能力方面：

采用的是面向对象的语言，在开发的时候有意识地采用了一些 OOP 的思路，通过一些方法减少了代码的重复。相信如果有更加井井有条的架构方法，那么实验过程会是一个非常有乐趣的过程（暂时还不是）。 Github 之前就在用，但这次系统学习了用 git 进行版本控制，并且应用了，入门，相信对以后的开发会有很大帮助。

附录

处理完的一些标注网页
一个三个网页叠加的 SST 图