司法大数据分析说明文档

词法分析

实现功能

对案件进行分词处理，以便后续进行分词，标注等

实现工具

LTP**分词工具**

LTP实现自动分词

安装流程

```
pip install ltp
```

载入模型

```
from ltp import LTP
ltp = LTP() # 默认加载 Small 模型
# ltp = LTP(path = "base|small|tiny")
# ltp = LTP(path = "tiny.tgz|tiny-tgz-extracted") # 其中 tiny-tgz-extracted 是 tiny.tgz 解压出来的文件夹
```

用户自定义词典

在对样例直接进行分词时，发现效果并不理想

有许多我们希望得到的专有名词，例如：法院名称，公司名称，地址名称等会被拆分开
考虑到问题的复杂性和程序的复杂性，在这里没有选择机器学习等高级方法，选择使用添加自定义词典的方式
使用正则先从目标文本中筛选出我们想要的专有名词（具体实现见代码），加入到分词的自定义词典中

```
def nlp(text):
    ltp = LTP()
    my_word_list = gen_my_word_list(text)
    ltp.add_words(words=my_word_list, max_window=4)
```

词性标注

```
    # 分词
    # segments 是包含所有分词结果的 list
    # part_of_speech 是分词结果中每个词对应词性的 list
    seg, hidden = ltp.seg([text])
    part_of_speech = ltp.pos(hidden)[0]
    segments = seg[0]
```

客户端

实现功能

1. 前端界面
2. 保存文件和标注

实现工具

JQuery

客户端布局主要由JQuery实现

安装流程

- 从 jquery.com 下载 jQuery 库

使用流程

jQuery库就是一个JavaScript文件，我们可以在HTML中使用`<script>`标签引入jQuery库

```
    <script src="js/jquery/2.0.0/jquery.min.js"></script>
```

Particles

particles实现背景的动态效果

`particlesJS` 开源在Github上：https://github.com/VincentGarreau/particles.js

项目中直接引入particles.js文件和app.js的配置参数文件就可以直接调用

Blob对象实现保存标注

Blob 对象表示一个不可变、原始数据的类文件对象。

生成Blob对象

```
const blob = new Blob([content], { type: "text/plain; charset=utf-8" });
```

Blob下载数据

```
const blob = new Blob([content], { type: "text/plain; charset=utf-8" });
           const a = document.createElement("a");
           a.href = URL.createObjectURL(blob);
           a.download = "案件信息提取.json";
           a.click();
```

当结束使用某个 URL 对象之后，应该通过调用URL.revokeObjectURL()这个方法来让浏览器知道不用在内存中继续保留对这个文件的引用了整代码

服务端

实现功能

文件的传输，前后端的交互

实现工具

Express框架实现文件上传

Express 是一个方便开发者的 web 框架，可以让开发者可以方便地处理路由，Cookie, 静态文件，上传文件， RESTFULL风格等等常见操作。

安装流程

```
npm install express --save
```

```
npm install multer
```

此时node_moduels目录下的模块就是express间接要依赖的模块

创建package.json目录

```
npm init
```

静态文件准备

在public目录中装有html，css，js文件，即前端界面

app.js

```
var express = require('express');
var path = require('path')
var fs = require("fs");
var multer  = require('multer');

var app = express();
//指定静态文件位置
app.use(express.static(path.join(__dirname, 'public')))
app.use(multer({ dest: path.join(__dirname, 'tmp') }).array('file'));

//获取后缀名
function getExtName(fileName){
    var index1 = fileName.lastIndexOf(".");
    var index2 = fileName.length;
    var extName = fileName.substring(index1+1,index2);
    return extName;
}

app.post('/uploadFile', (req, res) => {
   //获取上传文件的后缀名
   var extName = getExtName(req.files[0].originalname);

   //随机数
//    var rundomNumber = Math.ceil(Math.random()*10000000);
   //以随机数作为文件名
   var randomFileName =  "case." + extName;

   //文件路径
   var fileFile = __dirname + "/public/file/" + randomFileName;

   //上传临时文件的路径
   var uploadedTempFilePath = req.files[0].path;
```

```
    //读取临时文件
    fs.readFile( uploadedTempFilePath, (err, data) => {
        //读取成功之后，复制到文件路径
         fs.writeFile(fileFile, data, (err) => {
             //写成功之后，返回 file元素显示上传之后的图片
             res.writeHead(200, {'Content-Type': 'text/html'});
             res.end("Save successfully!");
             // if (err) console.log(`[SERVER] error: ${err}`);
        });
    });
});
```

即可实现上传文件功能

WebSocket通信

WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。

WebSocket 使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在 WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行双向数据传输。

安装流程

安装node.js

安装ws模块

服务端实现

```
const WebSockect = require('ws')
const wss = new WebSockect.Server({port: 3000});
console.log("setup server...");

// 用于启动一个子进程执行python脚本
var exec = require("child_process").exec;
// 用于解决后端中文显示乱码问题
var iconv = require('iconv-lite');

wss.on('connection', function (ws) {
    console.log(`[SERVER] connection()`);
    ws.on('message', function (message) {
        console.log(`[SERVER] Received: ${message}`);
        // 执行本地python脚本
        exec('py .\\TextNLP.py '+ message, {encoding: "gbk"},
            (error,stdout,stderr) => {
                if(stdout.length > 1) {
                    var result = iconv.decode(stdout, "GBK");
                    // 后端输出pyhton的stdout
                    console.log('python stdout:', result);
                    // 将python的输出发送给前端
                    // 注意exec方法中的这个匿名函数作为参数是一个没有返回值的函数
                    // 一定要在该函数体内将结果传给前端，否则将拿不到数据
                    ws.send(result, (err) => {
                        if (err) console.log(`[SERVER] error: ${err}`);
                    });
                } else console.log('you don\'t offer args');
```

```
                if(error) console.info('stderr : '+ stderr);
        });
    });
});
```

爬虫

实现功能：

爬取100份中国司法案例网（https://anli.court.gov.cn/）案件

实现工具：

Python-->selenium

实现步骤

1 创建webDriver，打开指定网址，并进入第一个案例开始迭代

```
    # 创建 WebDriver 对象，指明使用chrome浏览器驱动
    wd = webdriver.Chrome()

    # 调用WebDriver 对象的get方法 可以让浏览器打开指定网址
    wd.get(web)

    # 设置最大等待时长为 10秒
    wd.implicitly_wait(10)

    # 进入此页面的第一个case
    wd.find_element(By.CSS_SELECTOR, 'body .ul_list li .li_h').click()
```

2 逐个爬取每个网页的信息，并写入txt文档

```
def spiderByCase(wd, rank):
    # 等待页面刷新
    sleep(1)

    # 找到所有文本的内容
    fullText = wd.find_elements(By.CSS_SELECTOR, '.caseDetailContent #content
.text p')

    # 定位到“裁判结果”，并保存之后的第二个webElement，用flag标识是否遇到了“裁判结果”
    flag = 0
    content = ""
    for paragraph in fullText:
        # 遇到了裁判结果栏目
        if paragraph.text == "裁判结果":
            flag = 1
        else:
            # 刚刚遇到这个栏目 需要经过一个空隙
            if flag == 1:
                flag += 1
            # 到达文本内容
            elif flag == 2:
                content = paragraph.text
```

```
                break

        write(content, rank)

        # 进入上一篇
        wd.find_element(By.CSS_SELECTOR, 'body .prev').click()

        return wd


def write(content, rank):
    path = 'res/' + str(rank) + '.txt'
    f = open(path, 'w', encoding='utf-8')
    f.write(content)
    f.close()
```