期末实验报告

在此次实验报告中将会分为五个部分进行单独分析，这五个部分分别是爬虫、界面、文本搜索、图片搜索以及音频搜索。

爬虫：

一、库

1. requests
2. urllib

二、实现功能

1. 能够通过 requests.get 获得网页上的内容，并使用 json.loads 进行解析
2. 能够快速定位歌曲、专辑的信息，包括专辑名、歌手、歌词、专辑简介、发行时间、流派、发行公司等
3. 能够从网页上下载歌曲和图片

三、实现过程

1．爬取 QQ 音乐网站的榜单，通过榜单下载歌曲

2. 分析发现搜索各首歌曲的 url 地址中的区别在于其中的一段内容代表的是搜索输入的文字的编码，利用这一特性模拟搜索从榜单上爬取下来的歌曲

3. 通过分析网页的 url 地址，发现每一首歌曲都有自己特定的”media_mid“属性，这个属性是各个歌曲的下载 url 地址唯一不同的地方。通过将该属性替换到 url 中的固定位置，可以完成歌曲、专辑、歌词定位

4. 定位到歌曲、专辑、歌词、图片地址后完成文本的写入

四、难点

1. reques 取得 url 地址是搜索页面下的单曲一栏，然而这里面并没有歌词，故将搜索歌曲的页面从单曲切换到歌词

2. 由于该页面中有大量中文字符，会出现 json 无法解析的编码，故使用正则表达式作类似处理
3. 专辑信息只能通过歌曲的 js 代码中获得的 media_mid 属性定位后，再通过正则表达式来提取

界面：

一、库

1.Wow.min.js

2.animate.css

3.jquery.min.js

二、实现的功能

1.由于文本搜索的结果较多，故使用 javascript 实现了分页的效果。设置最大显示量为 20 个结果，每页显示 4 个结果，共五页。 并实现了翻页、上一页、下一

页的功能。

效果如下：

代码如下:

```
<script>
    <script language="JavaScript" type="text/javascript" >
    var total="20";//最大数据量
    var pagesize="4";//每页的显示量
    var pagenum=Math.ceil(total/pagesize);//一共有多少页

    //生成导航
    function initpage(nowpage){
        var pagenav='<a href="#" onclick=up('+ nowpage+');>上一页</a>';
        for(i=1;i<=pagenum;i++){
        pagenav += (i==nowpage)?'<span class="now">' + i + '</span>':'<a class="number" href="#" onclick=goToPage(' + i + ');>' + i + '</a> ';
     }
    pagenav += '<a href="#" onclick=down('+ nowpage+');>下一页</a>';
    document.getElementById('pages').innerHTML=pagenav;
    }
    //翻页
    function goToPage(page){

        var start=(page-1)*pagesize;
        var max=page*pagesize;
        for(vs=0;vs<total;vs++){
            var obj=document.getElementById('item_' + vs);
            obj.style.display="none";
        }
        if(max>total)max=total;
        for(v=start;v<max;v++){
            var obj=document.getElementById('item_' + v);
            obj.style.display="block";
        }
        initpage(page);
    }
    //上一页
    function up(nowpage){
        if(nowpage==1)return false;
        if(nowpage>1)goToPage((nowpage-1));
    }
    //下一页
    function down(nowpage){
        if(nowpage==pagenum)return false;
        if(nowpage<pagenum)goToPage((nowpage+1));
    }
</script>
```

2.轮播图

可自动滚动播放海报，亦可点击左右切换按钮或下方圆形选择按钮切换海报。

效果如下：

代码如下：

```
        //轮播图函数
        function jAutoPlay() {
            if (curIndex < $jUlLis.length) {
                curIndex++;
            } else {
                curIndex = 1;
                $jUl[0].style.left = 0;
            }
            $jUl.stop().animate({
                left: -liWidth * curIndex
            });
            if (jOlIndex < $jOlLis.length-1) {
                jOlIndex++;
                console.log(jOlIndex)
            } else {
                jOlIndex = 0;
            }
            $jOl.children().eq(jOlIndex).addClass("current").siblings().removeClass("current");
        }

    });
})
```

3.NEW SONGS 板块从上部飞入 slideInDown 的动画效果

（1）当下拉至 NEW SONGS 这一板块时,图片从左到右依次从上部向下飞入指定位置。

（2）当检测到鼠标滑过图片区域时，图片下部的歌曲信息由原先的一行专辑名

再增加一行歌手名。

4.PROMOTION 板块的 bounceInDown 弹跳飞入动画效果

当检测到鼠标进入图片区域时，浮于图片上显示图片的专辑名、歌手名等信息。

代码如下：

```
<!-- t-promot-start -->
<div class="t-promot w">
    <div class="title">
        <h2>PROMOTION<br><span>热门推荐</span></h2>
    </div>
    <div class="t-promot-t clearfix">
        <a href="javascript:;" class="t-a-imgs wow bounceInDown" data-wow-delay="0.2s"><span><i>你身边还有我<br/>阿里郎</i></span><img src="/static/images/
        t-promot1.jpg"/></a>
        <a href="javascript:;" class="t-a-imgs wow bounceInDown" data-wow-delay="0.3s"><span><i>别对我动情<br/>关晓彤</i></span><img src="static/images/
        t-promot2.jpg"/></a>
        <a href="javascript:;" class="t-a-imgs wow bounceInDown" data-wow-delay="0.2s"><span><i>新世界<br/>小贱</i></span><img src="static/images/
        t-promot3.jpg"/></a>
        <a href="javascript:;" class="t-a-imgs wow bounceInDown" data-wow-delay="0.2s"><span><i>星星照亮回家的路<br/>庞龙</i></span><img src="/static/images
        /t-promot4.jpg"/></a>
    </div>
</div>
<!-- t-promot-end -->
```

5.回到顶部按钮

按钮置于页面右侧，点击即可返回到页面顶部。

代码如下：

```
//返回顶部
    var timer = null;
    $(".t-btn7").click(function(){
        var leader = scroll().top;
        clearInterval(timer);
        timer = setInterval(function(){
            var target = 0;
            var step = (target-leader)/10;
            step = step>0 ? Math.ceil(step):Math.floor(step);
            leader = leader+step;
            window.scrollTo(0,leader);
            if(target == leader){
                clearInterval(timer);
            }
        },25)
    })
    function scroll() {  // 开始封装自己的scrollTop
        if(window.pageYOffset != null) {  // ie9+ 高版本浏览器
            // 因为 window.pageYOffset 默认的是  0  所以这里需要判断
            return {
                left: window.pageXOffset,
                top: window.pageYOffset
            }
        }
        else if(document.compatMode === "CSS1Compat") {    // 标准浏览器   来判断有没有声明DTD
            return {
                left: document.documentElement.scrollLeft,
                top: document.documentElement.scrollTop
            }
        }
        return {   // 未声明 DTD
            left: document.body.scrollLeft,
            top: document.body.scrollTop
        }
    }
})
```

6.上传图片、音频

使用 GET 方法提交表单，用户浏览得到文件及其文件名，将文件名传递给处理函数以进行进一步的处理。

代码如下：

```
<!--search部分 start-->
<form  action="/i_s" method="GET">
    <button type="submit">上传</button>
    <input type="file" name="uploadfile" id="uploadfile" value="Browse..." />
</form>
<!--search部分 end-->
```

三、问题及解决

1.遇到的问题主要是在界面的整合过程中。在运行代码过程中报错 “Codec with

name ‘Lucene410’ does not exist”查阅一些资料后，发现大部分出现此类问题的情况是由于缺少 META-INF/services 这样的文件夹，可以通过将 jar 包中的 META-INF 复制到 src 目录的方式解决。但多次尝试之后发现并不可行，再次研究思考报错内容后，认为问题在于

整合界面的同学所使用的 Lucene 的版本为 4.9.0 而写音频搜索的同学的 Lucene 的版本为 4.10.0，两者版本不一，使用版本低的 lucene 时会报错。整合界面的同学使用了 miniconda 安装了 4.9.0 版本，利用 conda update 更新 lucene 后，版本仍为 4.9.0，原因应是 miniconda 中的 lucene 最高版本号是 4.9.0，故采取了卸载原来安装的 4.9.0 版本的 lucene，通过 lucene PPT 中提到的第二种安装方式安装了 4.10.0 版本的 lucene 最终解决了这一问题。

文本搜索：

一、库

1. lucene

2. jieba

二、原理

使用 jieba 与 lucene 对文本内容进行分词以此实现快速搜索

三、实现步骤

1. 建立索引

2. 实现搜索文件

图片搜索：

一、库

1.H5py

2.OpenCV

3. Numpy

二、原理

利用 SIFT 算法对图片进行特征提取，然后利用 Hash 对图片特征分类建立数据库，在搜索过程中对待检索图片做同样操作以完成匹配。

由于 sift 特征具有尺度不变性、旋转不变性等良好特征，故图片进行匹配时，两张图片相似度要求并不是很高。

三、实现步骤

1.对爬虫得到的文件中的所有图片用 OpenCV 的内置 sift 进行特征提取：

```
sift=cv2.SIFT()
```

2.将每张图片提取的 sift 特征做归一化处理

```
r,c=des.shape
argmaximum=np.argmax(des)
argminimum=np.argmin(des)
nmax=argmaximum/c
nmin=argminimum/c
maximum=des[nmax][argmaximum-nmax*c]
minimum=des[nmin][argminimum-nmin*c]
```

3.将归一化处理后的数据使用直方图统计，并将 bins 设为下图所示：

```
b=[0.07,0.1,0.13,0.16,0.19,0.22,0.25]
```

4.以得到的直方图为依据做 Hash 处理，将得到的矩阵变为字符串储存为一个字

典的 key 值，然后再将具有同样 Hash 值得图片信息作为 value 储存入该字典中：

```
if not dic.has_key(hashname):
    grphash=f.create_group(hashname)
    dic[hashname]=grphash

imgname="img"+str(count)
desname="des"+str(count)

grpimg.create_dataset(imgname,data=imgori)
grppath.create_dataset(imgname,data=imgpath)
dic[hashname].create_dataset(imgname,data=des)
```

5．在检索图片时，对待检索图片进行上述相同操作之后，从 H5py 中取得同样 Hash 值的图片信息，然后对这些图片一一使用欧氏距离匹配：

```
bf=cv2.BFMatcher()

keys=imggroup.keys()
paths=[]
matchimgs=[]

for k in keys:
    des1=imggroup[k][:]
    matches=bf.match(des1,des)

    d=0
    for j in matches:
        d+=j.distance

    if d<400:
        paths.append(imgpaths[k].value)
        matchimgs.append(imgs[k][:])
```

四、测试

从网上下载各类不同的专辑图片来对库中的图片进行匹配，可以得到：

1.所使用的 bins 划分是能够比较均匀的将库中的图片划分在各个不同的 Hash 值下面的

2.网上的图片与库中原有的图片的欧式距离是小于 400 的

3.库中有 4913 首歌曲，搜索到对应图片的时间需要 3s，但若将搜索嵌入到搜索引擎中，页面上出来结果大致需要 6s，猜测与电脑渲染页面速度有关

五、对比

与使用颜色空间创建 Hash 值相对比，该使用 sift 特征值来 Hash 的办法可以排除颜色的干扰，也就是即使亮度、灰度有变化，也不会影响图片的匹配。

音频搜索：

一、库

1.ffmpeg

2.wave

3.pyaudio

4.mysqldb

二、原理

基于内容的音频检索与基于内容的图片检索在实际实现上面临着类似的问题，包括巨大的信息量以及特征点相对位置对于搜索结果的巨大影响，对于检索的准确度要求以及对于检索速度的要求同样限制着音频检索的实现。这里参照图片检索的相关方式对音频检索进行了尝试，主要思想是在音乐的频谱中提取频率峰值特征点，基于这些特征点对不同的歌曲进行匹配。（主要用 lucene 解决了音频匹配速度问题）

三、实现步骤：

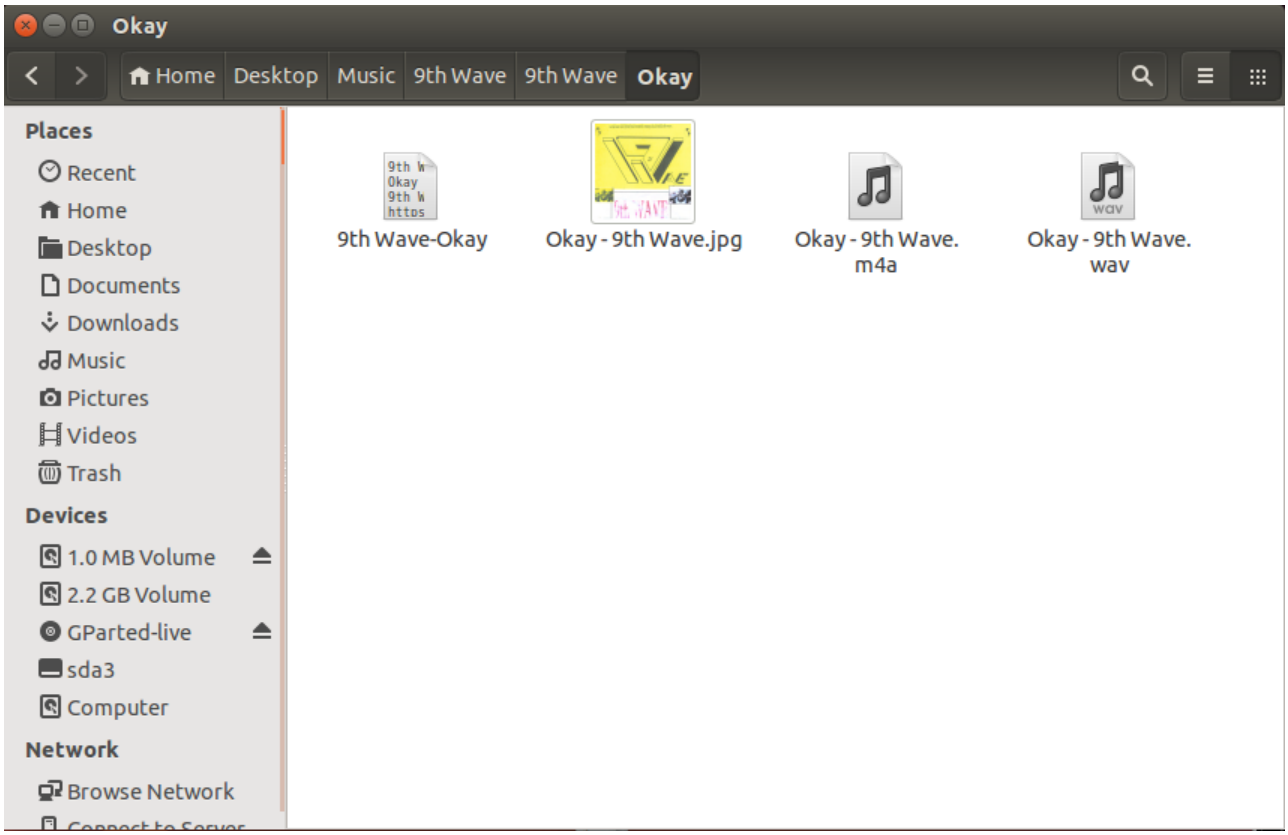
1. 音频格式转码

网页爬虫下载后的音频格式为 m4a，为使用 wave 库，需对下载的音频进行转码，将其转为 wav 格式，这里使用安装在 linux 中的 ffmpeg 进行转码，转码后的文件存入与原文件相同的目录下，与原文件只有后缀名不同。

代码：

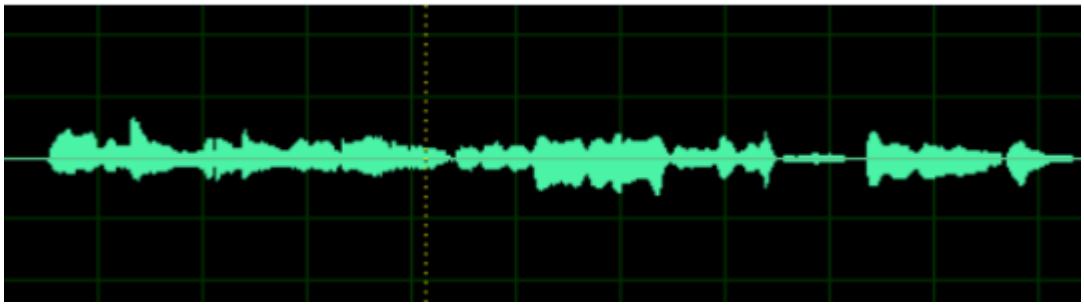
```
try:
    subprocess.call(["ffmpeg","-i",origin,newsr])
    sss.addsong(newsr2,extra,album)
except:
    continue
```

目录下的文件组成：

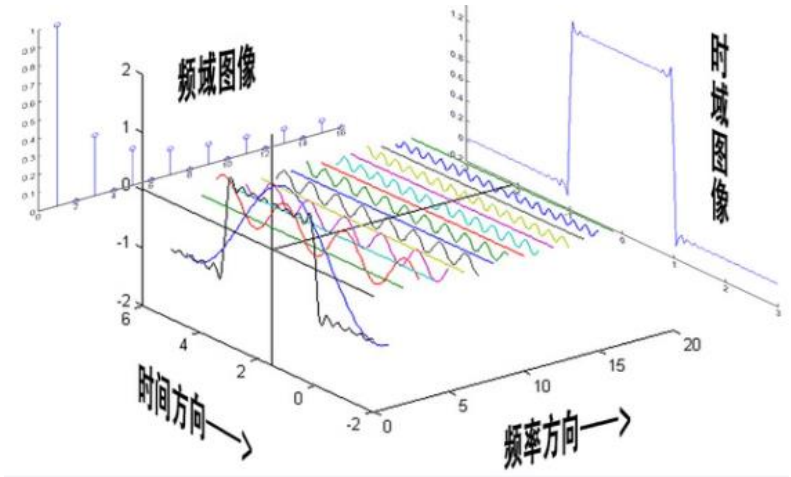


2. 提取音频特征

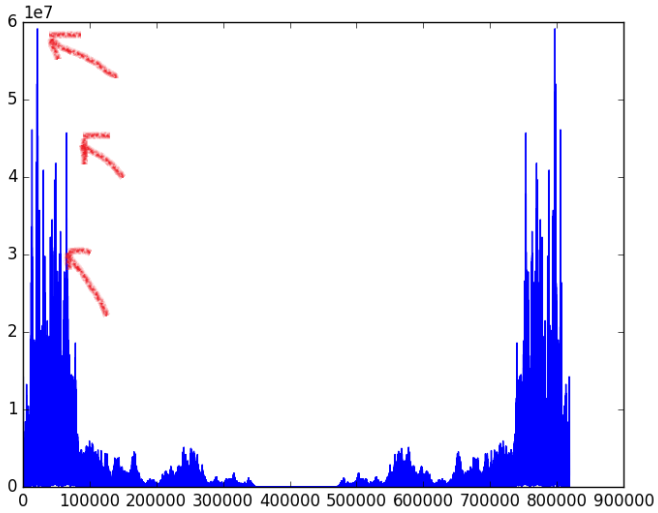
打开步骤 1 转码后的音频文件，该文件的一个声道是一个一维数组的形式，图形化如图：



这种时间对应幅值的一维数组信息量巨大，难以提取特征信息，故我们利用傅里叶变换将其转化到频域中：



转化的结果：



这样，音频信息转为了其对应的众多频率波形的叠加，而人耳正是靠着所听到的音乐的特定频率序列来区分音频的。在频域信号的众多特征中，我们认为峰值特征是最具代表性的，这种特征代表着这一频率能量最大，它容易提取，同时对于噪声有一定的容忍度。如果只是简单地提取整首歌曲中的一些峰值点，则我们会丧失重要的时间信息，故应当对整段音频进行分段，对每一段音频做傅里叶变换并提取特征峰值点，这样我们在得到特征点的同时也较好地保留了时间的信息。

这里将音频的 1s 分成 40 个块，在每一块的 4 个对应区间中得到 4 个峰值点，这 4 个峰值点组成的序列即是在一个块中得到的音频指纹。

3. 数据存储

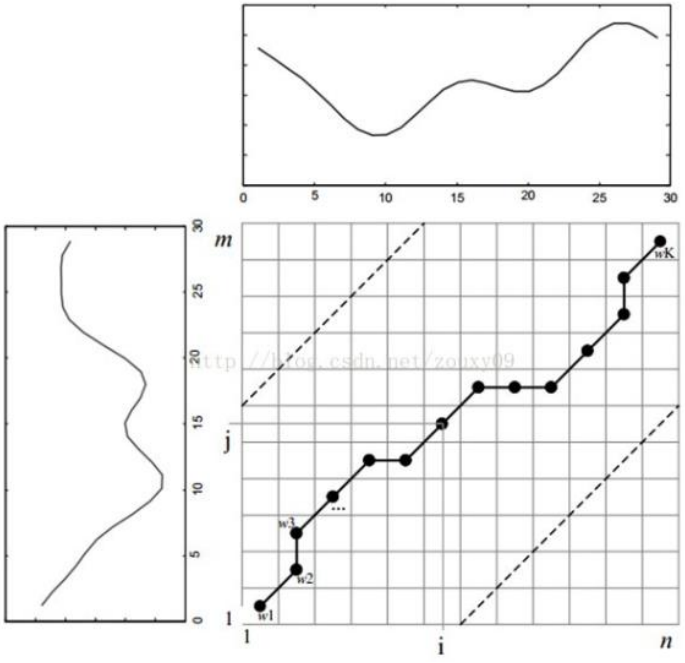
为了应用的方便，这里将得到的指纹序列存入了 mysql 数据库中，如图所示：

```
mysql> show columns from musicdata
    -> ;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| song_name | varchar(200) | YES  |     | NULL    |       |
| fp        | longtext     | YES  |     | NULL    |       |
| extra     | longtext     | YES  |     | NULL    |       |
| album     | longtext     | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

4. 匹配方法

这里设计了两种匹配方法，dtw 以及分词法匹配，考虑到对检验速度的要求，实际应用中选择了后者，但如果要求更高的精度，建议选择前者：

A. dtw，即动态时间规划，这种方法适于匹配两个不同长度的音频文件，主要思想是找出穿过 m*n 矩阵的最佳路径，图中上方和左方为我们要匹配的两个不同长度的音频文件，中间为这两个音频文件构造的矩阵，矩阵中每一个格点为两音频中对应点的相似度，中间的线即为找出的最佳路径：



具体实施原理及方法参照：

http://blog.csdn.net/zouxy09/article/details/9140207

B. 分词法匹配，这种方法的精度不高，但相比于 dtw 的匹配所花费的时间要大大减小。主要思想是把音频特征提取过程中提取出的特征点（即四个数字组成的序列）看作是文字检索中的一个词，如将 2,40,13,77 看作是文字搜索中的一个词，并用空格将这些词隔开，使用 lucene 中的 whitespace 分词器对特征点组成的“文章”进行分词以及语法分析，构建索引。这样，对特征点的检索就变成了简单的文字检索。

5. 搜索

主要搜索过程：

A. 从本地上传一个音频文件

B. 对该音频文件进行转码，将其转化为 wav 格式

C. 对转码后的音频文件提取特征点

D. 用逻辑与连接这些特征点，构建 query，与索引结果进行匹配，返回得分最高的 50 个结果

四、测试

测试结果显示，dtw 将测试音频与库中的音频进行匹配时平均每一首歌需花费 1.5s，这表明当库的容量非常大时其所消耗的时间将是难以想象的，因此，我们选择用 lucene 的分词系统对音频进行过滤。最终测试的结果是，其中百分之八十的正解将会出现在 lucene 打分结果的前五名中，有大概百分之 15 的正解将会出现在 5 至 40 名中，只有极少数的结果会出现在 40 名之后（出现这种现象的主要原因是不同的音频可能有相同的特征点，并且一些音频中含有的某种特征点的数量可能比正确结果的数量更大），故这种方法的精度可能不如 dtw，但在一定程度上可以满足检索的基本要求。

分工：

组长：

谢藩　负责内容：爬虫 文本搜索 图片搜索

成员：

鲁新钰 负责内容：音频搜索 界面细节完善

徐思慧 负责内容：界面 文本搜索