基于主题模型的文档检索系统

说明书

华东师范大学

海量计算研究所

2014 年 3 月 14 日

1. 系统总体设计

1.1 目的意义

本软件的目的是给定一篇论文，找出已有的论文集中与此论文相似性较高的一系列论文集。
当期刊单位收到投稿时，希望能将收到的论文与已有的论文库中的论文进行比较，若相似度太高，则应该拒绝此论文。同时，期刊单位也希望，当读者在网上阅读一篇论文时候，能推荐出一系列相似的论文。同时，此软件也便于找出科研人员所研究的领域中相似的论文。

开发团队：贺珂珂、王科强、王晓玲、周傲英等。

1.2 主题模型的文档检索系统

主题模型的文档检索系统能够根据用户所提交的查询请求(即目标论文)，智能的为用户显示与当前论文相似的一系列论文集合。系统性能主要有以下特点：

- 高效性：多线程，系统能够根据用户的查询请求，快速给出相关的论文集合。
- 准确性: 系统采用 LDA 主题模型进行建模，能综合分析论文中词与词之间的关联，且利用 Lucene 建立倒排索引，检索得到的论文有较高的相似度。
- 用户交互特性：界面简单友好，能够提供便捷的交互式的查询模版定制。

系统主要由以下模块组成：

（1）数据预处理模块

论文集都是 pdf 格式的，先将 pdf 格式的文件转为文本形式，同时去除 pdf 中无关的信息，去除停用词，将单词词根化，标准化文本形式。

（2）数据建模模块

该模块是将预处理后的文档信息进行 LDA 主题建模，形成 LDA 训练后的文件，同时根据 tf-idf 算出每篇文档的关键词，将每篇文档的关键词利用 Lucene 建立倒排索引。

（3）文档搜索模块

该模块是接受用户查询的文档，进行查询处理，呈现用户查询结果。

（4）数据中心模块

该模块是数据建模模块，文档搜索模块的中间桥接模块，负责几个模块的数据交互工作。

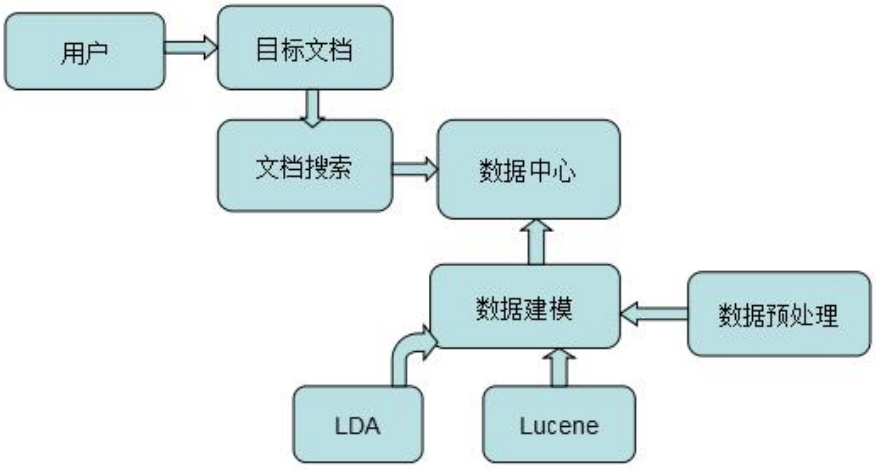


图 1 基于主题模型的文档检索系统的总体架构

1.3 全文组织结构

第一章：概述基于主题模型的文档检索系统的功能框架，系统中数据的处理，以及该系统的特点和优点。

第二章：该系统中采用的相关技术。

第三章：主题模型的文档检索系统各个功能模块的详细设计，说明每个模块中主要包含的类以及各类中成员的作用。

第四章：主题模型的文档检索系统的运行概况，系统的有效性的验证。

2. 相关基本技术

这一章我们将介绍主题模型的文档检索中涉及的两种主要的技术——LDA 主题模型和 Lucene 索引技术。

2.1LDA 主题模型

2.1.1LDA 简介

隐含狄利克雷分布简称 LDA(Latent Dirichlet allocation)，是一种主题模型，它可以将文档集中每篇文档的主题按照概率分布的形式给出。同时它是一种无监督学习算法，在训练时不需要手工标注的训练集，需要的仅仅是文档集以及指定主题的数量 k 即可。此外 LDA 的另一个优点则是，对于每一个主题均可找出一些词语来描述它。

2.1.2LDA 数学模型

LDA 是一种典型的词袋模型，即它认为一篇文档是由一组词构成的一个集合，词与词之间没有顺序以及先后的关系。一篇文档可以包含多个主题，文档中每一个词都由其中的一个主题生成

另外，正如 Beta 分布是二项式分布的共轭先验概率分布，狄利克雷分布作为多项式分布的共轭先验概率分布。因此正如 LDA 贝叶斯网络结构中所描述的，在 LDA 模型中一篇文档生成的方式如下:

- 从狄利克雷分布α中取样生成文档 i 的主题分布θ_i
- 从主题的多项式分布θ_i中取样生成文档 i 第 j 个词的主题$z_{i,j}$
- 从狄利克雷分布β中取样生成主题$z_{i,j}$的词语分布$\phi_{z_{i,j}}$
- 从词语的多项式分布$\phi_{z_{i,j}}$中采样最终生成词语$w_{i,j}$

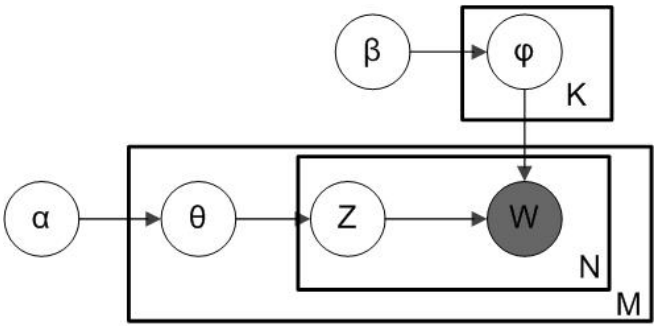
因此整个模型中所有可见变量以及隐藏变量的联合分布是

$$p(w_i, z_i, \theta_i, \Phi|\alpha, \beta) = \prod_{j=1}^{N} p(\theta_i|\alpha)p(z_{i,j}|\theta_i)p(\Phi|\beta)p(w_{i,j}|\theta_{z_{i,j}})$$

最终一篇文档的单词分布的最大似然估计可以通过将上式的θ_i以及Φ进行积分和对z_i进行求和得到

$$p(w_i|\alpha,\beta) = \int_{\theta_i}\int_{\Phi}\sum_{z_i} p(w_i, z_i, \theta_i, \Phi|\alpha,\beta)$$

根据$p(w_i|\alpha,\beta)$的最大似然估计，最终可以通过吉布斯采样等方法估计出模型中的参数。



图二. LDA 贝叶斯网络结构

2.2Lucene 索引技术

2.2.1 Lucene 简介

Lucene是apache软件基金会4 jakarta项目组的一个子项目，是一个开放源代码的全文检索引擎工具包，即它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎（英文与德文两种西方语言）。Lucene的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。

Lucene作为一个优秀的全文检索引擎，内部采用倒排索引，其系统结构具有强烈的面向对象特征。首先是定义了一个与平台无关的索引文件格式，其次通过抽象将系统的核心组成部分设计为抽象类，具体的平台实现部分设计为抽象类的实现，此外与具体平台相关的部分比如文件存储也封装为类，经过层层的面向对象式的处理，最终达成了一个低耦合高效率，容易二次开发的检索引擎系统。

2.2.2 Lucene 的结构和工作原理

Lucene总的来说具有一下特点：

- 一个高效的，可扩展的，全文检索库。
- 全部用Java实现，无须配置。
- 仅支持纯文本文件的索引(Indexing)和搜索(Search)。
- 不负责由其他格式的文件抽取纯文本文件，或从网络中抓取文件的过程。

Lucene 是有索引和搜索的两个过程，包含索引创建，索引，搜索三个要点，

其中被索引的文档用Document对象表示；IndexWriter通过函数addDocument将文档添加到索引中，实现创建索引的过程；Lucene的索引是应用反向索引；当用户有请求时，Query代表用户的查询语句；IndexSearcher通过函数search搜索Lucene Index；IndexSearcher计算term weight和score并且将结果返回给用户；返回给用户的文档集合用TopDocsCollector表示。索引过程：

（1）创建一个IndexWriter用来写索引文件，它有几个参数，INDEX_DIR就是索引文件所存放的位置，Analyzer便是用来对文档进行词法分析和语言处理的。

（2）创建一个Document代表我们要索引的文档。

（3）将不同的Field加入到文档中。我们知道，一篇文档有多种信息，如题目，作者，修改时间，内容等。不同类型的信息用不同的Field来表示，在本例子中，一共有两类信息进行了索引，一个是文件路径，一个是文件内容。其中FileReader的SRC_FILE就表示要索引的源文件。

（4） IndexWriter调用函数addDocument将索引写到索引文件夹中。

➢ 搜索过程：

（1） IndexReader将磁盘上的索引信息读入到内存，INDEX_DIR就是索引文件存放的位置。

（2） 创建IndexSearcher准备进行搜索。

（3） 创建Analyer用来对查询语句进行词法分析和语言处理。

（4） 创建QueryParser用来对查询语句进行语法分析。

（5） QueryParser调用parser进行语法分析，形成查询语法树，放到Query中。

（6） IndexSearcher调用search对查询语法树Query进行搜索，得到结果TopScoreDocCollector。

Lucene 的系统由基础结构封装、索引核心、对外接口三大部分组成，其中直接操作索引文件的索引核心又是系统的重点。Lucene 的将所有源码分为了 7 个模块（在 java 语言中以包来表示），各个模块所属的系统部分如图 9 所示。需要说明的是 org.apache.lucene.queryPaser 是做为 org.apache.lucene.search 的语法解析器存在，不被系统之外实际调用，因此这里没有当作对外接口看待，而是将之独立出来。

从面向对象的观点来考察，Lucene 应用了最基本的一条程序设计准则：引入额外的抽象层以降低耦合性。首先，引入对索引文件的操作 org.apache.lucene.store 的封装，然后将索引部分的实现建立在（org.apache.lucene.index）其之上，完成对索引核心的抽象。在索引核心的基础上开始设计对外的接口 org.apache.lucene.search 与 org.apache.lucene.analysis。在每一个局部细节上，比如某些常用的数据结构与算法上，Lucene 也充分的应用了这一条准则。在高度的面向对象理论的支撑下，使得 Lucene 的实现容易理解，易于扩展。

Lucene 在系统结构上的另一个特点表现为其引入了传统的客户端服务器结构以外的的应用结构。Lucene 可以作为一个运行库被包含进入应用本身中去，而不是作为一个单独的索引服务器存在。这自然和 Lucene 开放源代码的特征分不开，但是也体现了 Lucene 在编写上的本来意图：提供一个全文索引引擎的架构，而不是实现。

3. 模块详细设计

3.1 模块一览表

模块	功能
数据预处理模块	论文集都是 pdf 格式的，先将 pdf 格式的文件转为文本形式，同时去除 pdf 中无关的信息，去除停用词，将单词词根化，标准化文本形式。
数据建模模块	该模块是将预处理后的文档信息进行 LDA 主题建模，形成 LDA 训练后的文件，同时根据

	tf-idf 算出每篇文档的关键词，将每篇文档的关键词利用 Lucene 建立倒排索引。
文档搜索模块	该模块是接受用户查询的文档，进行查询处理，呈现用户查询结果。
数据中心模块	该模块是数据建模模块，文档搜索模块的中间桥接模块，负责几个模块的数据交互工作。

3.2 各模块的介绍

3.2.1 数据预处理模块

该模板主要是对文本信息进行过滤处理，标准化文本形式

（1） 论文集都是 pdf 格式的，先将 pdf 格式的文件转为文本形式

（2） 去除停用词

（3） 将单词词根化。

主要包含的类：

1）PDF2TXT 类：将 pdf 文档转化为 txt 文本，便于分析处理。

```
public class PDF2TXT {
    public static String getText(String pdffile)
    public static String getTextlist(List<String> pdffileList);
    public static String abstractExtractor(String contents) ;
    public static String titleExtractor(String[] content_arrays) ;
    public static String keywordsExtractor(String[] content_arrays) ;
    public static String referencesExtractor(String contents);
}
```

主要的函数说明：

```
public static String getText(String pdffile) {}
```

给定单一 pdf 文档，抽取内容信息。

```
public static String getTextlist(List<String> pdffileList){}
```

给定 pdf 文档的集合，抽取内容信息。

public static String abstractExtractor(String contents) {}

给定文档的内容，抽取文档的概要信息。

public static String titleExtractor(String[] content_arrays) {}

给定文档的内容，抽取文档的标题信息。

public static String keywordsExtractor(String[] content_arrays) {}

给定文档的内容，抽取文档的关键词信息。

public static String referencesExtractor(String contents) {}

给定文档的内容，抽取文档的引用信息。

2）ContentClean 类：过滤原 pdf 的信息。

public class ContentClean {

public static Analyzer analyzer;

public static boolean dic_load_judge = true;

public static String clean(String contents) ;

private static String stemmerParserString(String termsString) ;

private static String analyzeText(String text) throws IOException;

}

主要的函数说明：

public static String clean(String contents) {}

整合分词，去停用词，词根化。

private static String stemmerParserString(String termsString) {}

词根化。

private static String analyzeText(String text) throws IOException {}

分词，去停用词,长度大于 2 的词即过滤长度小于 3 的词 。

3）FileOperation 类：软件读取文件操作。

public class FileOperation {

public static List<Long> file_length_list;

public static Long all_files_length;

public static List<String> readAllFilePathConsRecursion(String

```
file_path,String          file_type, List<String> list);
        public static List<String> readAllFilePathCons(String file_path,String
file_type);
        public static String readFile(String fileName) ;
        public static void writeFile(String destFileName, String content) ;
        public static void writeFileNotAppdend(String destFileName, String
content);
        public static void writeFileNotAppdendMap(String destFileName,
        Map<String, String> contetn_map);
        public static void writeFileNotAppdendMapLDA(String destFileName,
        Map<String, String> contetn_map) ;
        public static ArrayList<String> getLineArrayList(String fileName) ;
        public static List<String> getLineLinkedList(String fileName) ;
        public static Map<String, String> getLineMap(String fileName) ;
        public static ArrayList<String> getLineListToLower(String fileName);

    }
```

主要的函数说明：

Public static List<String> readAllFilePathConsRecursion(String file_path,String file_type, List<String> list) {}

根据目的文件夹地址，递归一级一级地读取该文件夹目录下文档的信息。

public static String readFile(String fileName) {}

根据目的文档的地址，读取这个文档的信息。

public static void writeFile(String destFileName, String content) {}

根据要写的内容和要写的目的地址，进行写文档操作，追加着写。

public static void writeFileNotAppdend(String destFileName, String content) {}

根据要写的内容和要写的目的地址，进行写文档操作，若该地址已有内容，清空内容,重新写。

4）ContentParallelClean 类：并行进行过滤原 pdf 的信息。

```
public class ContentParallelClean implements Runnable {
    private int thread_num; 线程数量
    private int current_thread_id = 0;
    private Lock current_thread_lock;
    private int pdf_files_num;
    private PDFExtractor pdf_parallel_extr;
    private CountDownLatch count_down_latch;用于所用时间计数
    public ContentParallelClean(int thread_num_para,
            int current_thread_id_para, Lock current_thread_lock_para,
            PDFExtractor pdf_parallel_extr_para,
            CountDownLatch count_down_latch_para);
    public void run();
    public Lock getCurrent_thread_lock();
    public void setCurrent_thread_lock(Lock current_thread_lock) ;
}
```

主要的函数说明：

public Lock getCurrent_thread_lock() {}

返回当前的线程锁。

public void setCurrent_thread_lock(Lock current_thread_lock) {}

设定当前的线程锁。

public void run() {}

多线程运行程序。

5）PDFExtractor 类：从 pdf 中抽取特定的信息。

```
public class PDFExtractor {
    public String file_dir;
    public Map<String, String> id_content_map;
    public Map<String, String> id_title_map;
```

```
    public Map<String, String> id_clean_content_map;
    public Map<String, String> id_abstract_map;
    public Map<String, String> id_keywords_map;
    public Map<String, String> id_refs_map;
    public Map<String, String> id_pdffile_name_map;
    public Map<String, Long> id_file_length_map;
    public long all_length;
    public List<String> pdf_files_path_list;
    public static Set<String> wordSet;
    public static Set<String> oneWordSet;
    private static boolean pdf_read_judge = true;
    public static List<String> ext_stopwords_list;
    public static boolean ext_stopwords_read_judge = true;
    public static boolean stopwords_read_judge = true;
    public PDFExtractor(String file_dir_para);
    public PDFExtractor() ;
    public static void getStopWords() ;
    public void getPDFPath() ;
    public void getPDFPath(String dir_path) ;
    public static boolean isPdf_read_judge() ;
    public static void setPdf_read_judge(boolean pdf_read_judge) ;


}
```

主要的函数说明：

```
public static void getStopWords() {}
```

根据 stopword 的地址，获得 stopword。

```
public void getPDFPath() {}
```

获得 pdf 的路径。

```
public void getPDFPath(String dir_path) { }
```

根据给定的文件，获得 pdf 的路径。

public static boolean isPdf_read_judge() {}

判断当前的 pdf 文件是否被读。

public static void setPdf_read_judge(boolean pdf_read_judge) {}

设定当前的 pdf 文件是否被读。

3.2.2 数据建模模块

该模块是将预处理后的文档信息进行 LDA 主题建模，形成 LDA 训练后的文件，同时根据 tf-idf 算出每篇文档的关键词，将每篇文档的关键词利用 Lucene 建立倒排索引。

主要包含的类：

1） IDF 类：主要是为了抽取文档的关键词

```
public class IDF implements Runnable {
    public CountDownLatch count_down_latch;
    public Map<String, String> id_only_content_map;
   public IDF(Map<String, String> id_clean_content_map_para,
        CountDownLatch count_down_latch_para) ;
    public void idfCalculate() ;
    public void run() ;
}
```

主要的函数说明：

public void idfCalculate() {}：

计算 idf 的函数。

public void run() {}

Runnable 接口的实现。

1) LDATraining 类：LDA 主题模型建模的实现。

```
public class LDATraining implements Runnable {
public CountDownLatch count_down_latch;
public LDATraining(CountDownLatch count_down_latch_para);
```

public void trainTopics(String contentFileName, String inferenceName,
String distr_file, String stoplists_path, int numTopics,
int numIterations) ;
public void run() ;
}

主要的函数说明：

public void trainTopics(String contentFileName, String inferenceName,
String distr_file, String stoplists_path, int numTopics,int numIterations)
根据原有文本信息的输入，计算迭代的次数，建立 LDA 模型的结果，并保存， 便于新来文档时的主题推测。

public void run() {}
Runnable 接口的实现。

3）LuceneIndex 类：建立 lucene 的倒排索引。

public class LuceneIndex implements Runnable {
public CountDownLatch count_down_latch;
public Map<String, String> id_clean_content_map;
public LuceneIndex(Map<String, String> id_clean_content_map_para,
CountDownLatch count_down_latch_para) ;
public void createLucenceIndexNew() ;
public void run() ;
}

主要的函数说明：

createLucenceIndex()：
用于建立 lucene 的索引。
public void run() {}
Runnable 接口的实现。

3.2.3 文档搜索模块

该模块是接受用户查询的文档，进行查询处理，呈现用户查询结果。

主要包含的类：

1)LuceneSearch 类：返回利用 Lucene 倒排索引获得的相似结果。

```
public class LuceneSearch implements Runnable {
  public String contents;
  public CountDownLatch count_down_latch;
  public SearchThreadInfo search_info;
  public LuceneSearch(String contents_para,
        CountDownLatch count_down_latch_para,
        SearchThreadInfo search_info_para) ;
  public Map<String, Float> similaryByLucene(String importantWord)
        throws IOException, ParseException ;
  public String getImportantWord(String contents) ;
  public void run() ;
}
```

主要的函数说明：

public Map<String, Float> similaryByLucene(String importantWord)

输入关键的字符串，获得相似的结果。

public String getImportantWord(String contents)

根据 tf-idf 原理，返回输入文档中的关键字符串组合。

2）MergeLDASimilarity 类：返回利用 LDA 主题模型获得的相似结果。

```
public class MergeLDASimilarity {
  public Map<String, Float> id_score_map;
  public List<KeyValueSort> final_simpapers_list;
  public double[] query_probabilities;
  public MergeLDASimilarity(Map<String, Float> id_score_map_Para,
```

```
                double[] query_probabilities_para) ;
    public void cosine() ;
}
```

主要的函数说明：

public void cosine() {}

根据每篇文档的不同主题的分布，计算其 cosine 相似度。

3)QueryPDFExtractor 类： 抽取输入文档的有效信息。

```
public class QueryPDFExtractor {
    public String query_pdf_path;
    public String query_contents;
    public QueryPDFExtractor(String query_pdf_path_para) ;
   public void queryPDFExtClean() ;
}
```

主要的函数说明：

public void queryPDFExtClean() {}

将 pdf 转化为 txt 同时抽取出文档的内容，概要等不同信息。

4）SearchMain 类：根据输入文档的路径，将上述类进行组合，获得最终相似文档的结果。

```
public class SearchMain implements Runnable {
    String file_path;
    SearchList sl;
    private CountDownLatch count_down_latch;
    public Lock topic_lock;
   public SearchMain(String file_path,SearchList sl,
            CountDownLatch count_down_latch_para, Lock topic_lock) ;
   public List<KeyValueSort> searchParallel(String file_path) ;
   public void run() ;
}
```

主要的函数说明：

public List<KeyValueSort> searchParallel(String file_path) {}

根据输入的文档，并行地进行预处理，LDA 相似检索，Lucene 相似检索等综合搜索处理。

5）TopicDistr 类：处理主题分布

```
public class TopicDistr implements Runnable {
    public String contents;
    public CountDownLatch count_down_latch;
    public SearchThreadInfo search_info;
    public Lock topic_lock;
    public TopicDistr(String contents_para,
            CountDownLatch count_down_latch_para,
            SearchThreadInfo search_info_para,Lock topic_lock_para)；

   public void run() ；
}
```

主要的函数说明：

public void calcTopicDistr(){}

计算文档的主题分布。

6)BatchParallelSearch 类：批量并行搜索

```
public class BatchParallelSearch implements Runnable {
    public SearchList batch_sl;
    private CountDownLatch count_down_latch;
    private Lock current_thread_lock;
    private int pdf_files_num;
    private int thread_num;
    private int current_thread_id = 0;
   public BatchParallelSearch(SearchList batch_sl,int thread_num_para,
            int current_thread_id_para, Lock current_thread_lock_para,
            CountDownLatch count_down_latch_para) ;
```

```
    public void batchSearch();
    public void run() ;
}
```

主要的函数说明：

public void batchSearch(){}

进行批量搜索。

3.2.4 数据中心模块

该模块是数据建模模块，文档搜索模块的中间桥接模块，负责几个模块的数据交互工作。

主要的类：

1）ConfigSet 类：保存软件设置，等配置信息。

```
public class ConfigSet {

    public static boolean model_listen_judge = true;
    检测文档集是否经过训练
    public static boolean batch_listen_judge = true;
    检测是否进行批量搜索
    public static String error_log_path = "files/log/error_log.txt";
    设定故障日志地址
    public static String operation_log_path = "files/log/operation_log.txt";
    设定用户操作日志地址
    public static String pdf_dir = "files/paperdata";
    设定文档集的地址
    public static String index_dir_path = "files/index";
    设定存储文档标号信息的地址
    public static String final_content_path = "files/processedData/id_content_lda.txt";
    设定lda训练文档的地址
    public static String id_title_Path = "files/processedData/id_title.txt";
    设定文档标题的地址
    public static String id_filePath_Path = "files/processedData/id_filePath.txt";
    设定文档路径的地址
    public static String term_idf_path = "files/processedData/idf.txt";
    设定单词idf的地址
    public static String lda_dir_path = "files/index";
```

```
public static String temp_dir_path = "files/temp";
public static String stopwords_path = "files/stoplist/ext_stopword.dic";
设定停用词的地址
public static String lda_stopwords_path = "files/stoplist/en.txt";
public static String lda_distr_path = "files/lda/distr.txt";
设定lda主题分布的地址
public static String original_data_dir_path = "files/originalData";
设定初始文档信息的地址
public static String lda_inference_path = "files/lda/inference.txt";
设定LDA中间信息的地址
public static String lda_instances_path = "files/lda/instances.txt";
设定LDA中间信息的地址
public static String batch_result_file = "batchResults.txt";
设定批量搜索结果的地址
public static String query_pdf_dir = "query_pdf";
public static String query_pdf_last_dir = "";
public static String batch_pdf_dir = "query_pdf";
public static String batch_pdf_last_dir = "";
public static String train_pdf_dir = "query_pdf";
public static String train_pdf_last_dir = "";
public static int num_topics = 20;
设定LDA训练的主题数
public static int num_iterations = 1200;
设定LDA训练的迭代数
public static int num_search_iterations = 12;
设定LDA搜索的迭代数
public static int title_multiple = 40;
public static int abstract_multiple = 4;
public static int keywords_multiple = 40;
public static int refs_multiple = 1;
public static int pdf_extractor_thread_num = 4;
public static int term_analyzer_thread_num = 4;
public static int lda_thread_num = 4;
设定LDA训练的线程数
public static int thread_num = 4;
设定线程数
public static int simpaper_num = 30;
public static int simpaper_start = 5;
public static int simpaper_end = 10;
public static int query_term_num = 100;
public static float lda_lambda = 0.47f;
设定LDA训练的初始数值
public static float lda_alpha = 0.05f;
设定LDA训练的初始数值
```

```
    public static int display_num = 10;
    设定显示的相似文档的个数
    public static int batch_search_num = 0;
    设定批量搜索文档个数
    public static boolean thread_num_judge = true;
    public static Map<String, String> rank_id_map;
    public final static String test_file_path = "files/test/test.pdf";
    public static boolean test_judge = true;
    设定测试文档的路径
    public static SimpleDateFormat matter = new SimpleDateFormat(
            "yyyy-MM-dd,HH:mm:ss");
    设定默认的日期格式
    public static String config_path = "files/config.txt";
    设定配置文件的地址
    public static boolean config_judge = true;
    检测是否进行配置文件设定
    public static int width = 510;
    public static int height = 415;
    public static int info_height = 30;
    public static int process_height = height - 40 - info_height - 60;
    public static List<String> stop_words_list = new LinkedList<String>();
    public static boolean read_judge = true;
    public static Map<String, Float> term_idf_map;
    读取search需要的文档
    public static boolean search_read_judge = true;
    public static boolean search_judge = true;
    public static QueryParser query_parser;
    public static IndexSearcher index_search;
    public static Directory dir;
    public static Map<String, String> id_title_map;
   public static void readConfigInfo() ;
   public static List<String> readStopWords() ;
   public static Map<String, Float> readIdfInfo() ;
   public static void readTitleInfo() ;
   public static void readFilePathInfo() ;
   public static void readTopicDistrInfo() ;
   public static void readLDAInfo() ;
   public static void readLuceneInfo() ;
   public static void readAllSearchInfo() ;

}
```

主要的函数：

```
public static void readConfigInfo() {}
```

读取配置文件信息。
public static List<String> readStopWords() { }
读取配置文件信息。
public static Map<String, Float> readIdfInfo() {}
读取词的idf信息。
public static void readTitleInfo() {}
读取词的title信息。
public static void readFilePathInfo() {}
读取文件路径信息。
public static void readTopicDistrInfo() {}
读取主题分布信息。
public static void readLDAInfo() {}
读取LDA信息。
public static void readLuceneInfo() {}
读取lucene索引信息。

2）PreprocessParallelMain 类：并行进行预处理

```
public class PreprocessParallelMain {
    public static Map<String, String> id_clean_content_map;
    public static Map<String, String> id_only_content_map;
    public static void allPreprocess(PDFExtractor pdf_parallel_extr) ;
}
```

主要的函数说明：

public static void allPreprocess(PDFExtractor pdf_parallel_extr)

进行所有文档的预处理工作。

2）TrainingParallelMain 类：并行进行文档训练。

```
public class TrainingParallelMain {
public static void trainParallel() ;
}
```

主要的函数说明：

public static void trainParallel() {}

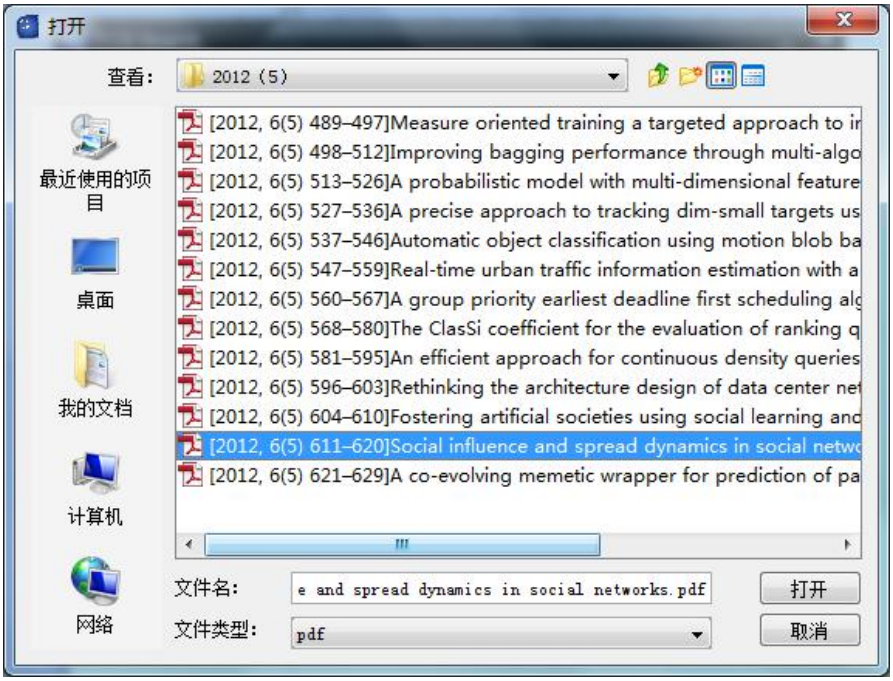
将所有文档集中的文档进行训练，产生相关的文件。

3）ParallelMain 类：读取配置信息，进行预处理，和训练综合工作。

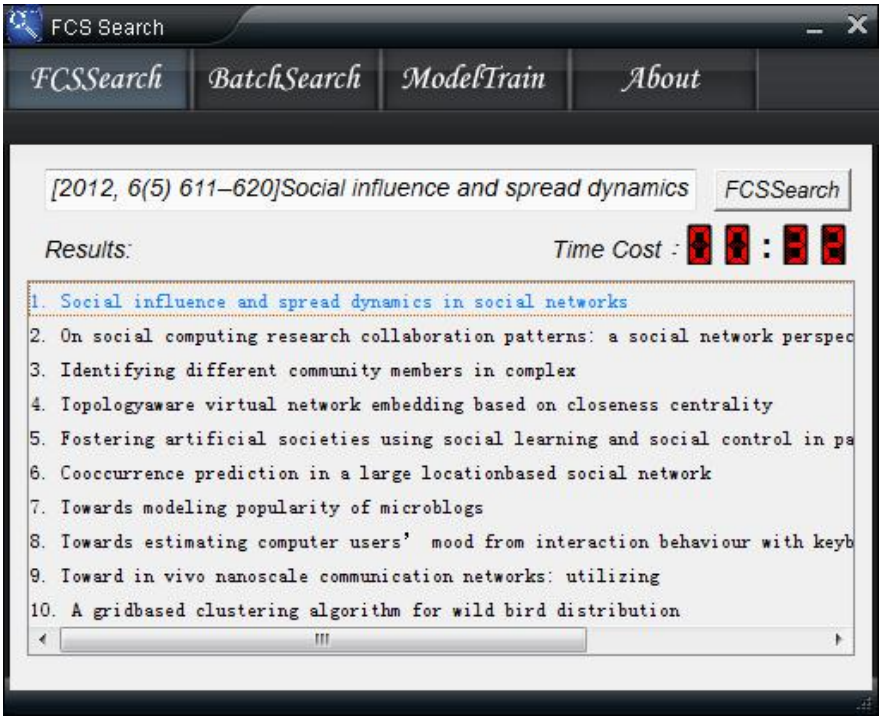
4. 系统查询结果的展现

4.1 单文档相似检索

用户首先挑选任意一篇文档，选择搜索，系统将会按照相似程度的高低返回一系列文档的集合，用户可以选择点开文档名，直接查看原 pdf 文件。图三显示用户选择一篇 pdf 文档，欲从文档集中选择跟这篇 pdf 文档相似的文档。图四显示了跟用户选择的 pdf 文档相似的文档。



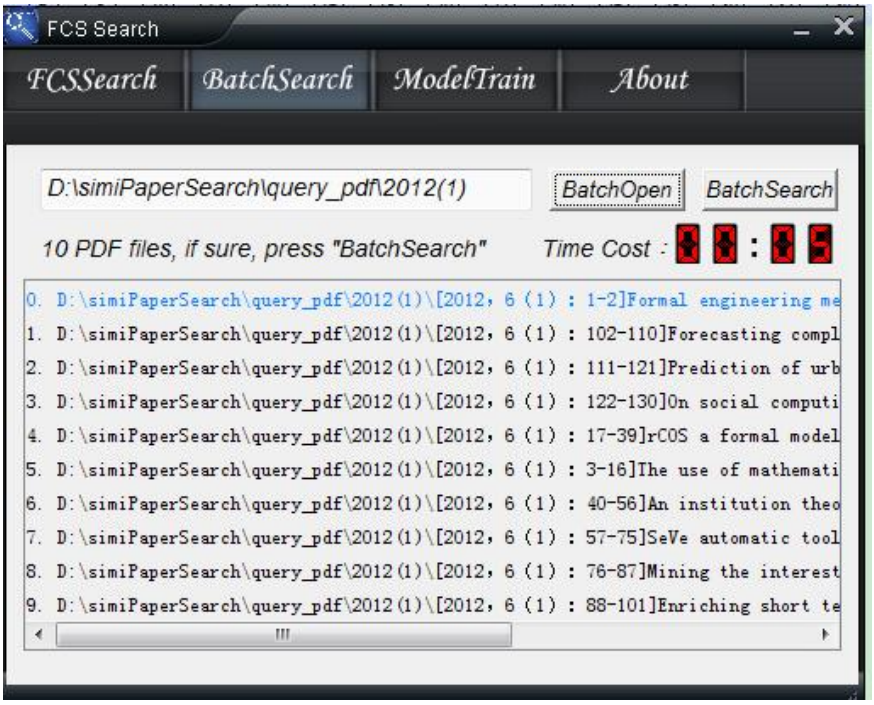
图三.用户选择一篇文档



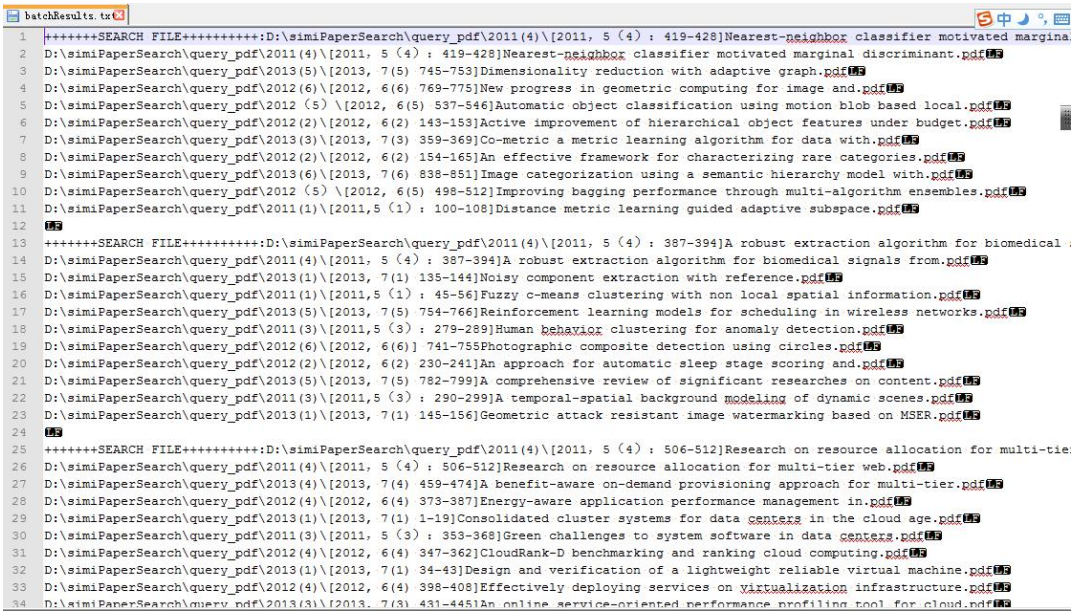
图四.相似文档检索结果

4.2 批量检索

用户可以选择文件夹下的一些列文章进行批量检索，在 batchResults.txt 文件中将显示批量检测的结果。图五显示用户选择批量搜索显示，图六显示批量搜索的结果。



图五.批量搜索显示



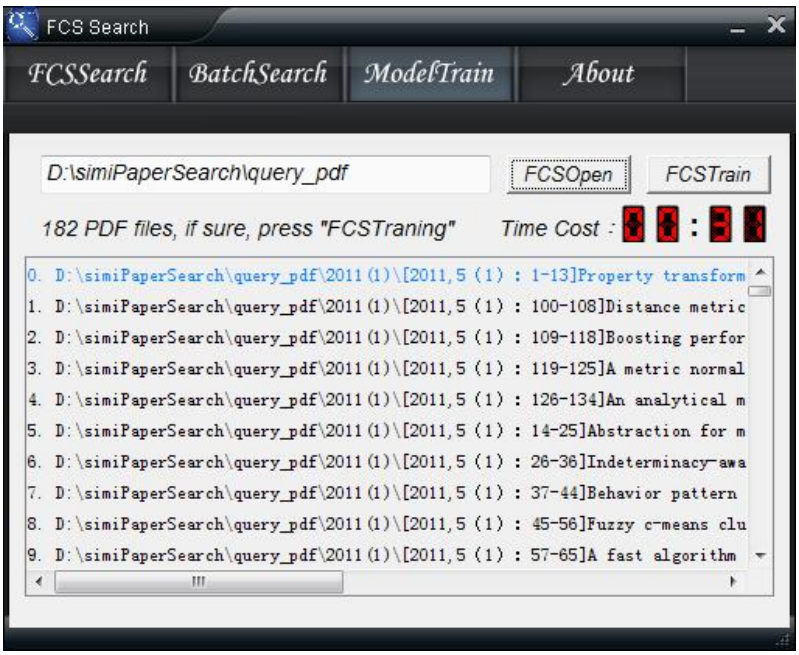
图六.批量搜索结果

4.3 模型训练

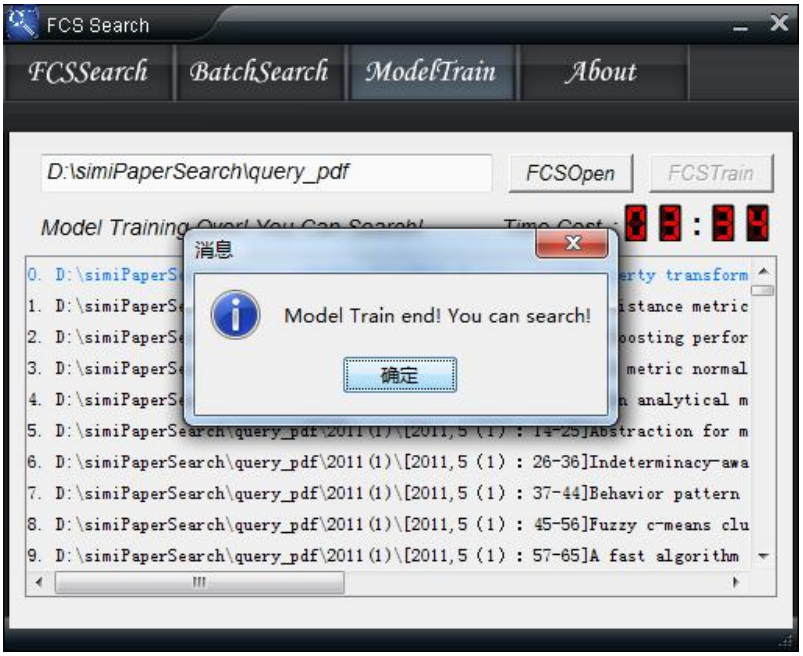
当我们一开始使用这个软件时，需要对我们已有的文档集进行训练，这样才能进行相似文档检测工作。

当我们又有新的文档加入到文档集的时候，我们也需要对文档集进行重新的训练。如图七，我们的文档集的大小为 182 篇 pdf 文档。

如图八，训练成功。



图七.文档集的训练


FCS Search
FCSSearch
BatchSearch
ModelTrain
About
D:\simiPaperSearch\query_pdf
FCSOpen
FCSTrain
消息
Model Train end! You can search!
确定
0. D:\simiPaperS
1. D:\simiPaperS
2. D:\simiPaperS
3. D:\simiPaperS
4. D:\simiPaperS
5. D:\simiPaperSearch\query_pdf\2011(1)\[2011,5(1) : 14-25]Abstraction for m
6. D:\simiPaperSearch\query_pdf\2011(1)\[2011,5(1) : 26-36]Indeterminacy-awa
7. D:\simiPaperSearch\query_pdf\2011(1)\[2011,5(1) : 37-44]Behavior pattern
8. D:\simiPaperSearch\query_pdf\2011(1)\[2011,5(1) : 45-56]Fuzzy c-means clu
9. D:\simiPaperSearch\query_pdf\2011(1)\[2011,5(1) : 57-65]A fast algorithm

图八.文档集的训练成功