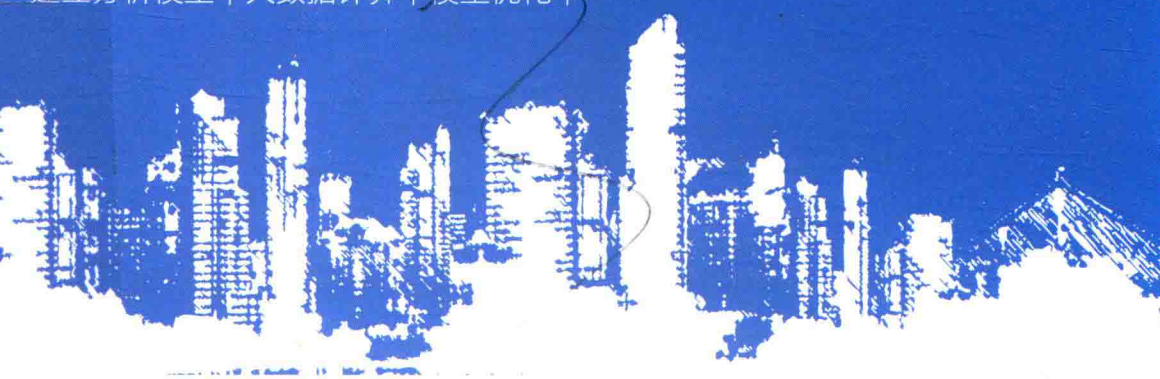
大数据分析方法

BIG DATA

陆　红◎著

大数据的处理过程

数据采集 | 大数据存储 | 大数据分析 |
建立分析模型 | 大数据计算 | 模型优化 |



中国财富出版社
CHINA FORTUNE PRESS

大数据分析方法

陆 红 著

中国财富出版社

图书在版编目（CIP）数据

大数据分析方法／陆红著．—北京：中国财富出版社，2017.3（2017.8 重印）

ISBN 978－7－5047－6416－4

Ⅰ.①大… Ⅱ.①陆… Ⅲ.①数据处理 Ⅳ.①TP274

中国版本图书馆 CIP 数据核字（2017）第 049009 号

策划编辑	寇俊玲	**责任编辑**	赵 翠		
责任印制	方朋远	**责任校对**	孙丽丽	**责任发行**	王新业

出版发行	中国财富出版社		
社　　址	北京市丰台区南四环西路 188 号 5 区 20 楼	**邮政编码**	100070
电　　话	010－52227588 转 2048/2028（发行部）	010－52227588 转 307（总编室）	
	010－68589540（读者服务部）	010－52227588 转 305（质检部）	
网　　址	http://www.cfpress.com.cn		
经　　销	新华书店		
印　　刷	北京九州迅驰传媒文化有限公司		
书　　号	ISBN 978－7－5047－6416－4/TP·0099		
开　　本	710mm×1000mm　1/16	**版　　次**	2017 年 6 月第 1 版
印　　张	10	**印　　次**	2017 年 8 月第 2 次印刷
字　　数	180 千字	**定　　价**	48.00 元

前　言

为什么要写本书

大数据时代，人们面对大量的数据，首先想到的是如何分析这些数据，但目前介绍大数据分析方法的书籍却不是很多。作者从事大数据分析研究工作多年，很想将做过的大数据研究课题积累的一些分析方法分享给各位读者。

本书组织结构

本书的结构是依据大数据的处理方法构建的，依次为大数据采集处理方法、大数据存储方法、大数据分布式计算方法、大数据分析模型构建方法、大数据分析模型检验方法、大数据分析模型优化方法。

本书的主要内容

第1章介绍了大数据采集处理方法，重点介绍了如何从互联网上采集数据，介绍了“网络爬虫程序”的设计和编写方法以及数据清洗方法，着重介绍如何清洗机器学习训练数据。

第2章介绍了大数据存储方法，重点介绍了分布式文件系统存储的原理、配置方法与使用方法。重点介绍了分布式数据库 Hbase 原理、配置方法与使用方法。

第3章介绍了大数据分布式计算的实现方法，详细介绍了如何搭建 Hadoop 大数据处理平台，列举了详细的搭建过程，提供了 Hadoop 搭建所需的各种配置文件源代码，提供了构建 Hadoop 所需的命令语句。着重介绍了 MapReduce 框架结构，运行机理，MapReduce 源代码分析，各种接口和类分析。本章给出了丰富的 MapReduce 示例，对示例进行详细的解读，读者可以模仿示

例编写自己的 MapReduce 程序。

第 4 章介绍了大数据分析模型构建方法，主要介绍了如何通过机器学习方法构建大数据分析模型。

第 5 章介绍了大数据分析模型的检验方法，分析模型建立以后如何进行检验，这往往是一个难点；如何检验模型是否达到了最初设计标准，用什么手段来检验非常关键；检验方法是否科学也都很重要。此章还介绍了机器学习构建模型常用的检验方法，如回归诊断、交叉验证等方法。

第 6 章重点介绍了模型建立以后如何优化，主要介绍了几种优化方法，如逐步回归优化法、主成分分析优化方法等，特别介绍了通过神经网络进行优化。

本书的优势

本书的内容是作者从事科研项目的一些研究成果及开展大数据分析项目研究方法的总结和归纳，因此内容非常实用，可为从事大数据分析的研究人员提供研究步骤和方法。本书的模式是提出问题，然后给出解决方法，这样读者在现实中遇到类似问题就可以参考书中提供的方法加以解决。

本书的受众群体

本书的读者主要是从事大数据分析的研究人员、大数据分析系统开发人员，使用本书的人员应该在大数据分析方面有一定的基础，本书的内容比较深入，跳过了基础概念和基本理论，主要论述和探索大数据分析深层次的内容和方法。

致谢

本书是基于北京市教委科技计划一般课题：“基于机器学习方法的房价大数据分析模型构建研究”（课题编号：KM 201610857002）研究内容编写的。

感谢项目组成员冀钢、刘瑞新、范美英对本项目做出的贡献。感谢参与和支持本项目的所有人员。

陆　红

2016 年 12 月

目 录

1 大数据采集处理方法

如何获取大数据，如何从互联网上采集大数据？

大数据可以通过多种渠道获得，通过互联网采集大数据是常用的一种方法，采集互联网大数据通常采用爬虫技术抓取数据。

1.1 爬虫程序设计方案

1. **抓取** URL

要抓取网页的内容，首先要得到此网页的网址，URL（Uniform Resource Locator，统一资源定位器）是网页的网址统称。

URL 可以采取种子网页的形式获得，即以事先给定的 URL 作为种子，顺着这颗种子再获得相关的链接网页的 URL；也可将网页进行分类，按类别抓取网页 URL；还可以通过日志获得访问网页的特征，依据访问特征抓取网页 URL。

2. **存储** URL

将种子 URL 存放到待搜索队列中，不断将获得的相关链接的 URL 放入待搜索队列，将搜索过的 URL 打上标记放入已搜索队列，标记最好以时间戳的形式标注，为下次搜索顺序提供依据。分析和过滤 URL，将与主题词无关的 URL 过滤掉。

3. **搜索策略**

URL 搜索策略通常采用深度搜索或广度搜索。深度搜索是纵向搜索，从种子首页开始向下搜索相关的链接网页；广度搜索是横向搜索，先搜索所有首页，然后再搜索二级页面、三级页面等。

4. **提取内容规则**

可以设定按主题词提取，根据种子样本标记的主题词和框上的内容提取网页内容，抓取的网页不断地比较样本标记的主题词，将与此主题词相关的内容存储在数据库中，主题词作为数据库的字段。

5. **采用网页排名算法选取网页**

网页排名算法 PageRank（PR）用于筛选网页，决定是否将该网页 URL 放入待搜索队列。如果网页 T 存在一个指向网页 A 的链接，则表明 T 的所有者认为 A 比较重要，从而把 T 的一部分重要性得分赋予 A。这个重要性得分值为：$PR\ (T)\ /L\ (T)$。

其中：$PR\ (T)$ 为 T 的 *PageRank* 值，$L\ (T)$ 为 T 的出链数；

则 A 的 *PageRank* 值为一系列类似于 T 的页面重要性得分值的累加。

假设一个只有 4 个页面组成的集合：A、B、C 和 D。如果所有页面都链向 A，那么 A 的 *PR* 值将是 B、C 及 D 的和。

即：$PR\ (A)\ = PR\ (B)\ + PR\ (C)\ + PR\ (D)$

继续假设 B 也有链接到 C，并且 D 也有链接到包括 A 的 3 个页面，一个页面不能投票 2 次。所以 B 给每个页面半票。以同样的逻辑，D 投出的票只有 1/3 算到了 A 的 *PageRank* 上。

$$PR\ (A)\ = \frac{PR\ (B)}{2} + \frac{PR\ (C)}{1} + \frac{PR\ (D)}{3}$$

换句话说，根据链出总数平分一个页面的 *PR* 值。

$$PR\ (A)\ = \frac{PR\ (B)}{L\ (B)} + \frac{PR\ (C)}{L\ (C)} + \frac{PR\ (D)}{L\ (D)}$$

$$PR(p_i)\ = \frac{1-q}{N} + q\sum_{p_j} \frac{PR(p_j)}{L(p_j)}$$

p_1，p_2，…，p_N 是被研究的页面，$M\ (p_i)$ 是 p_i 链入页面的数量，$L\ (p_j)$ 是 p_j 链出页面的数量，而 N 是所有页面的数量。

PR 值是一个特殊矩阵中的特征向量。这个特征向量为：

$$\boldsymbol{R} = \begin{pmatrix} PR\ (p_1) \\ PR\ (p_2) \\ \vdots \\ PR\ (p_N) \end{pmatrix}$$

$\boldsymbol{R}$ 是如下等式的一个解：

$$\boldsymbol{R}=\begin{bmatrix}(1-q)/N\\(1-q)/N\\\vdots\\(1-q)/N\end{bmatrix}+q\begin{bmatrix}e(p_1,p_1) & e(p_1,p_2) & \cdots & e(p_1,p_N)\\e(p_2,p_1) & \ddots & & \\\vdots & & e(p_j,p_j) & \vdots\\e(p_N,p_1) & \cdots & & e(p_N,p_N)\end{bmatrix}\mathrm{R}$$

如果网页 i 有指向网页 j 的一个链接，则

$$\sum_{i=1}^{N}e(p_i,p_j)=1,$$

否则 $e(p_i,p_j)=0$。

6. 依据主题词相似度分析算法抓取内容

假设 $\boldsymbol{X}$ 是词－文档矩阵，其元素 (i,j) 代表词语 i 在文档 j 中出现的次数，则 $\boldsymbol{X}$ 矩阵如下。

$$\begin{matrix}&&d_j\\&&\downarrow\\t_i^{\mathrm{T}}\rightarrow&\begin{pmatrix}x_{1,1}&\cdots&x_{1,n}\\\vdots&&\vdots\\x_{m,1}&\cdots&x_{m,n}\end{pmatrix}\end{matrix}$$

可以看到，每一行代表一个词向量，该向量描述了该词和所有文档的关系。

$$t_i^{\mathrm{T}}=\begin{bmatrix}x_{i,1}&\cdots&x_{i,n}\end{bmatrix}$$

相似的，一列代表一个文档向量，该向量描述了该文档与所有词的关系。

$$d_j=\begin{pmatrix}x_{1,j}\\\vdots\\x_{m,j}\end{pmatrix}$$

词向量 $\boldsymbol{t}_i^{\mathrm{T}}\boldsymbol{t}_p$ 的点乘可以表示这两个单词在文档集合中的相似性。矩阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 包含所有词向量点乘的结果，元素 (i,p) 和元素 (p,i) 具有相同的值，代表词 p 和词 i 的相似度。类似的，矩阵 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 包含所有文档向量点乘的结果，也就包含了所有文档之间的相似度。

现在假设存在矩阵 $\boldsymbol{X}$ 的一个分解，即矩阵 $\boldsymbol{X}$ 可分解成正交矩阵 $\boldsymbol{U}$ 和 $\boldsymbol{V}$，和对角矩阵 $\sum$ 的乘积。

这种分解叫作奇异值分解（SVD），即：

$$X = U\sum V^{\mathrm{T}}$$

因此，词与文本的相关性矩阵可以表示为：

$$XX^{\mathrm{T}} = (U\sum V^{\mathrm{T}})(U\sum V^{\mathrm{T}})^{\mathrm{T}} = (U\sum V^{\mathrm{T}})(V^{\mathrm{T}\mathrm{T}}\sum{}^{\mathrm{T}}U^{\mathrm{T}}) = U\sum V^{\mathrm{T}}V\sum{}^{\mathrm{T}}U^{\mathrm{T}} = U\sum\sum{}^{\mathrm{T}}U^{\mathrm{T}}$$

$$X^{\mathrm{T}}X = (U\sum V^{\mathrm{T}})^{\mathrm{T}}(U\sum V^{\mathrm{T}}) = (V^{\mathrm{T}\mathrm{T}}\sum{}^{\mathrm{T}}U^{\mathrm{T}})(U\sum V^{\mathrm{T}}) = V\sum{}^{\mathrm{T}}U^{\mathrm{T}}U\sum V^{\mathrm{T}} = V\sum{}^{\mathrm{T}}\sum V^{\mathrm{T}}$$

因为 $\sum\sum^{\mathrm{T}}$ 与 $\sum^{\mathrm{T}}\sum$ 是对角矩阵，因此 U 肯定是由 XX^{T} 的特征向量组成的矩阵，同理 V 是由 $X^{\mathrm{T}}X$ 特征向量组成的矩阵。这些特征向量对应的特征值即为 $\sum\sum^{\mathrm{T}}$ 中的元素。综上所述，这个分解如下：

$$\begin{array}{cccc} X & & U & \sum & V^{\mathrm{T}} \\ (d_j) & & & & (d_j) \\ \downarrow & & & & \downarrow \end{array}$$

$$(t_i^{\mathrm{T}})\rightarrow\begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} = (\hat{t}_i^{\mathrm{T}})\rightarrow\begin{bmatrix}\begin{bmatrix} \\ u_1 \\ \\ \end{bmatrix}\cdots\begin{bmatrix} \\ u_t \\ \\ \end{bmatrix}\end{bmatrix}\cdot\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & \sigma_t \end{bmatrix}\cdot\begin{bmatrix}[v_1] \\ \vdots \\ [v_t]\end{bmatrix}$$

σ_1，…，σ_t 被称作奇异值，而 μ_1，…，μ_t 和 v_1，…，v_t 则叫作左奇异向量和右奇异向量。通过矩阵分解可以看出，原始矩阵中的 t_i 只与 U 矩阵的第 i 行有关，我们则称第 i 行为 $\hat{t}_i$。同理，原始矩阵中的 $\hat{d}_j$ 只与 V^{T} 中的第 j 列有关，我们称这一列为 $\hat{d}_j$。t_i 与 $\hat{d}_j$ 并非特征值，但是其由矩阵所有的特征值决定。

当我们选择 k 个最大的奇异值，和它们对应的 U 与 V 中的向量相乘，则能得到一个 X 矩阵的 k 阶近似，此时该矩阵和 X 矩阵相比有着最小误差（即残差矩阵的 Frobenius 范数）。

但更有意义的是这么做可以将词向量和文档向量映射到语义空间。向量 $\hat{t}_i$ 与含有 k 个奇异值的矩阵相乘，实质是从高维空间到低维空间的一个变换，可以理解为是一个高维空间到低维空间的近似。同理，向量 $\hat{d}_j$ 也存在这样一个从高维空间到低维空间的变化。这种变换用公式总结出来如下：

$$X_k = U_k\sum{}_k V_k^{\mathrm{T}}$$

有了这个变换，则可以做以下事情：

（1）判断文档 j 与 q 在低维空间的相似度。比较向量 $\sum_k \hat{d}_j$ 与向量 $\sum_k \hat{d}_q$（比如使用余弦夹角）即可得出。

（2）通过比较 $\sum_k \hat{t}_i^{\mathrm{T}}$ 与 $\sum_k \hat{t}_p^{\mathrm{T}}$ 可以判断词 i 和词 P 的相似度。

（3）有了相似度则可以对文本和文档进行聚类。

（4）给定一个查询字符串，算其在语义空间内和已有文档的相似性。

要比较查询字符串与已有文档的相似性，需要把文档和查询字符串都映射到语义空间，对于原始文档，由以下公式可以进行映射。

$$\hat{d}_j = \sum_k^{-1} U_k^{\mathrm{T}} d_j$$

其中对角矩阵 $\sum^k$ 的逆矩阵可以通过求其中非零元素的倒数来简单的得到。

同理，对于查询字符串，得到其对应词的向量后，根据公式 $\hat{q} = \sum_k^{-1} U_k^{\mathrm{T}} q$ 将其映射到语义空间，再与文档进行比较。

1.2 爬虫程序实现方法

1. 抓取 URL 程序

建立一个爬虫方法：

```
public void crawl () throws Throwable {
```

持续抓取 URL，调用 getNextUrl（）方法抓取下一个 URL：

```
while (continueCrawling ()) {
CrawlerUrl url = getNextUrl ();
```

如果待抓取的队列中 URL 不为空，调用 getContent（）方法获取该 URL 网页的文本信息：

```
if (url ! =null) {
printCrawlInfo ();
String content = getContent (url);
```

采用正则法则，判断网页内容是否与主题相关，如果网页内容与主题有关，将该网页存入数据库：

```
if (isContentRelevant (content, this.regexpSearchPattern)) {^sav-
eContent (url, content);
```

调用 extractUrls（）方法，提取网页链接的 URL：

```
Collection urlStrings = extractUrls (content, url);
```

将网页内容中的 URL 存储到待爬队列：

```
addUrlsToUrlQueue (url, urlStrings);
} else {
System.out.println (url + " is not relevant ignoring... ");
 }
```

调用线程休眠方法，延时 URL 抓取，防止屏蔽：

```
Thread.sleep (this.delayBetweenUrls);
 }
      }
```

关闭输出流：

```
closeOutputStream ();
}
```

得到下一个 URL 方法：

```
private CrawlerUrl getNextUrl () throws Throwable {
CrawlerUrl nextUrl = null;
```

判断下一个 URL 是否为空，如果为空从队列中移除：

```
while ( (nextUrl = = null) && (! urlQueue.isEmpty ())) {
CrawlerUrl crawlerUrl = this.urlQueue.remove ();
```

判断是否有权限访问该 URL：

```
if (doWeHavePermissionToVisit (crawlerUrl)
```

判断 URL 是否已经访问过：

```
&& (! isUrlAlreadyVisited (crawlerUrl))
```

设置深度搜索最大深度限制：

```
&& isDepthAcceptable (crawlerUrl)) {
nextUrl = crawlerUrl;
System.out.println ("Next url to be visited is" +
nextUrl);
 }
    }
return nextUrl;
```

提取 URL 方法，通过 HashMap 匹配链接 URL：

```
public List extractUrls (String text, CrawlerUrl crawlerUrl) {
```

```
Map < string, string > urlMap = < /string, string > new HashMap < string, string > (); < /string, string >
extractHttpUrls (urlMap, text);
extractRelativeUrls (urlMap, text, crawlerUrl);
return new ArrayList (urlMap.keySet ());
}
```

处理外部链接，将匹配的外部链接放入待爬队列中：

```
private void extractHttpUrls (Map < string, string > urlMap, String text)
{
< /string, string > Matcher m = httpRegexp.matcher (text);
while (m.find ()) {
String url = m.group ();
String [ ] terms = url.split ( " a href = \ "" ); for ( String term: terms) {  System.out.println (" Term = " + term);
if (term.startsWith (" http")) {
int index = term.indexOf (" \ " "); if (index > 0) {
term = term.substring (0, index);}
urlMap.put (term, term);
System.out.println ("Hyperlink:" + term);
 }
}
}
}
```

处理内部链接，将匹配的内部链接放入待爬队列中：

```
private void extractRelativeUrls ( Map < string, string > urlMap, String text, < /string, string > CrawlerUrl crawlerUrl) {Matcher m = relativeRegexp.matcher (text);
URL textURL = crawlerUrl.getURL ();
String host = textURL.getHost ();
while (m.find ()) {
String url = m.group ();
String [] terms = url.split ("a href = \ "");
for (String term: terms) {
if (term.startsWith (" /")) {
```

```
    int index = term.indexOf (" \" ");
    if (index > 0) {
    term = term.substring (0, index);
       }
    String s = " http: //" + host + term;
    urlMap.put (s, s);
    System.out.println ("Relative url:" + s);
            }
          }
        }
      }
    }
```

2. 抓取网页程序

定义 http 应答方法，参数为 http、url、datum：

```
public HttpResponse (HttpBase http, URL url, CrawlDatum datum)
throws ProtocolException, IOException
```

判断协议是否为 http 协议：

```
if (!" http" .equals (url.getProtocol ()))
throw new HttpException (" Not an HTTP url:" + url);
```

获得路径：

```
String path = " " .equals (url.getFile ())?" /": url.getFile ();
```

得到主机名：

```
String host = url.getHost ();
```

根据 url 获取到主机名和端口名。如果端口不存在，则端口默认为 80：

```
int port;
String portString;
if (url.getPort () == -1) {
port = 80;
portString = " ";
   } else {
port = url.getPort ();
portString = " :" + port;
}
socket = new Socket ();
```

设置 socket 连接超时的时间：

```
socket socket.setSoTimeout (http.getTimeout ());
```

是否使用代理，获取 socket 主机和 socket 端口：

```
String sockHost = http.useProxy ( )? http.getProxyHost ( ): host;
int sockPort = http.useProxy ()? http.getProxyPort (): port;
```

创建 Socket 地址：

```
InetSocketAddress sockAddr = new InetSocketAddress ( sockHost,
sockPort);
```

建立 Socket 链接：

```
socket.connect (sockAddr, http.getTimeout ());
```

Socket 获取输出流：

```
OutputStream req = socket.getOutputStream ();
```

向服务器发出 Get 请求：

```
StringBuffer reqStr = new StringBuffer ("GET");
```

如果使用代理服务器，添加网址：

```
if (http.useProxy ()) {
reqStr.append (url.getProtocol ( ) + ": //" + host + portString +
path);
 } else {
reqStr.append (path);
}
reqStr.append ("HTTP/1.0 \r \n");
reqStr.append ("Host:");
reqStr.append (host);
reqStr.append (portString);
reqStr.append (" \r \n");
reqStr.append ("Accept - Encoding: x - gzip, gzip \r \n"); String
userAgent = http.getUserAgent ();
```

如果代理服务器为空：

```
if ( (userAgent = = null) | | (userAgent.length () = = 0)) {
if (Http.LOG.isFatalEnabled ()) {Http.LOG.fatal ("User - agent is
not set!");}
} else {
```

添加代理服务器：

```
reqStr.append ("User-Agent:");
reqStr.append (userAgent);
reqStr.append ("\r\n");} reqStr.append ("\r\n");
byte [] reqBytes = reqStr.toString () .getBytes ();
req.write (reqBytes);
req.flush ();
```

获得输入流，包括协议、缓存大小：

```
PushbackInputStream in = new PushbackInputStream (new BufferedInp-
utStream (socket.getInputStream (), Http.BUFFER_ SIZE), Http.BUFFER_
SIZE);
boolean haveSeenNonContinueStatus = false;
```

提取状态码和 HTML 中的头文件：

```
while (! haveSeenNonContinueStatus) {
this.code = parseStatusLine (in, line);
parseHeaders (in, line);
haveSeenNonContinueStatus = code ! =100;}
```

读取文件格式：

```
readPlainContent (in);
```

获取文件的格式，得到头内容编码，如果是压缩的文件则处理压缩：

```
String contentEncoding = getHeader (Response.CONTENT_ ENCODING); if ("
gzip" .equals (contentEncoding) | |"x-gzip" .equals (contentEncoding)) {
content = http.processGzipEncoded (content, url);} else {
if (Http.LOG.isTraceEnabled ())  {Http.LOG.trace ("fetched" +
content.length + "bytes from" +url);
    }
  }
```

提取状态行，压回输入流，从缓存中取出状态码：

```
private int parseStatusLine (PushbackInputStream in, StringBuffer
line) throws IOException, HttpException:
```

获得状态码开始位置和结束位置，状态码长度为结束位置减去开始位置：

```
int codeStart = line.indexOf (" ");
int codeEnd = line.indexOf (" ", codeStart +1);
if (codeEnd = = -1)
codeEnd = line.length ();
```