一种用于企业数据的清洗方法

本发明涉及计算机技术、数据库、数据处理等理论与技术，主要应用于企业数据的清洗和标准化。

本系统解决了数据清洗只清洗重复数据而不清洗相似数据的问题，并且系统还将相关的相似数据进行聚合展示。同时在数据预处理阶段，系统内置了多种变换规则(拼音相同汉字匹配，学名与俗称匹配，大小写匹配，全角半角匹配，连接符号相似匹配，字符串相似匹配等)，通过这些规则可以将源数据根据需要进行多种变换，用于提高数据清洗的广度和深度。最后基于词典的分词也将相同的但是词序不同的数据进行了清洗。

与其他数据清洗工具相比，本系统提高了数据清洗的广度：不仅清洗重复数据，也同样清洗相似数据；本系统提高了数据清洗的深度：系统内置的多种数据变换规则扩大了数据清洗的适用性，同时对于清洗数据的相似度对比是基于词典分词的，并加以相似度计算的检测，能够有效提高数据准确率；本系统还提高了数据清洗的效率：对待清洗的数据先进行搜索服务的检索能有效的缩小相似度比对的范围，减小计算规模，缩短清洗时间。

数据清洗流程

如上图所示，为本系统的一个整体流程图。

系统可导入Excel等格式的数据文件，经过解析生成具体数据存入系统；在数据预处理阶段，系统根据用户所选规则(拼音相同汉字匹配，学名与俗称匹配，大小写匹配，全角半角匹配，连接符号相似匹配，字符串相似匹配等)进行数据的预处理生成待清洗的数据；清洗功能是本系统最核心的模块，它将待清洗数据根据特定算法进行相似度检测，并将高于用户设定的某一相似度阈值的相关数据进行聚类；系统根据用户的一系列操作(待定，确认，移除等)将结果缓存起来，之后可以进行同一批源数据的重复清洗；直到确认数据后，才会生成最终所需的数据关系文件并导出如Excel等格式的文件。



清洗数据模块

本系统最核心的功能就是清洗数据阶段。主要目的就是将杂乱无章的数据进行整理聚合，清除出相似或者相同的垃圾数据，提高数据质量。

清洗的流程实现如上图所示，首先将待清洗数据导入到搜索应用服务中，然后通过相似度服务模块计算相似度并进行后续的操作。具体的一次实现是，当将数据导入搜索应用服务时，搜索应用服务会对数据进行基于词典的分词并添加数据索引；然后通过搜索应用的搜索服务，找到某一数据相关的全部数据，再将相关的全部数据循环与这一数据进行基于词典分词的相似度计算，将相似度达到阈值的相关数据进行缓存，同时将搜索应用服务中的这些数据去除，防止重复数据的出现。如此往复循环整个待清洗的全部数据，直到所有数据都经过了这样的清洗流程表明本轮清洗结束，可以开始下轮清洗或者确认数据并导出。通过构造数据存储结构，并结合数据压缩算法，将磁盘里的数据放入内存，减少磁盘随机读取次数，提高数据清洗效率。

传统关系型数据库的索引储存方式采用B-Tree数据结构，能够有效减少磁盘的寻道读取次数和数据查找效率。与传统关系型数据库不同，本系统的搜索应用服务采用的是基于“倒排索引“的非关系型搜索服务。倒排索引为每个数据字段都建立基于字段的数据索引（形如key-values），如下例。

例如为以下数据建立索引：

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | AGE | SEX |
| 1 | Kate | 24 | Female |
| 2 | John | 24 | Male |
| 3 | Bill | 29 | Male |

以NAME字段建立索引：

|  |  |
| --- | --- |
| Kate | 1 |
| John | 2 |
| Bill | 3 |

以AGE字段建立索引：

|  |  |
| --- | --- |
| 24 | [1,2] |
| 29 | 3 |

以SEX字段建立索引：

|  |  |
| --- | --- |
| Female | 1 |
| Male | [2,3] |

为每个字段都建立一个这样的索引，用来储存针对某一字段的具有相同索引条件(term)的所有文档ID。

但当数据量提升的时候，一个最重要的问题是如何提高查询速度。这里我们采用了一种基于内存的树形结构来加快搜索速度，和一种基于bitmap的索引值存储方式来压缩数据所占用的内存空间。



数据索引树

根据索引条件(term)形成一种基于内存的树形结构(如上图)，这棵树不会包含所有的完整数据词组，它包含的是数据的一些公共前缀。例如针对某一字段的索引条件Terms为（to,tea,ted,ten,A,inn,in），根据每个term来依次的构造整个树。首先存入单词to的路径，发现根节点下没有为t的前缀路径，则在根节点下构造转移条件为t的路径和存储内容为t的节点，然后在t节点下同样没有发现有转移条件为o的路径，则在节点t下构造转移条件为o的路径和节点to，形成单词to路径的存储；当存入单词tea时，发现根节点下已存在转移条件为t的路径，然后根据路径找到节点t并移动到节点t下，在节点t下未发现转移条件为e的路径，则在节点t下构造路径转移条件为e的路径和相应的节点并继续往下执行，数据tea接下来的流程以及之后数据的构造都以此类推，直到所有term都加入到这棵树中方为结束。当进行数据查询时，我们就可以根据待查询数据和这棵树形结构来高效的进行查询，对于待查询数据，我们可以把其前缀依次作为转移条件来进行节点间的搜索，例如查询条件term为单词inn时，我们就可以按照i->n->n这样的转移条件来作为数据的搜索的转移路径(如上图中的黑色连接线所示)，首先从根节点出发，根据条件i转移到相应的节点，然后从该节点出发，根据条件n转移到相应的节点，最后从此节点出发，再根据条件n进行转移，如果有该转移条件的路径就可以快速找到该数据对应的节点，反之如果不存在该转移条件的路径就表面该查询term没有对应的文档数据。通过Terms的公共前缀，并根据Terms的状态转移(例如从t->te就是根据状态e来进行转移的)来构造树形结构存储整个Terms，这样使得全部的索引数据都能够加载到内存中，再把数据按转移条件进行查询，比直接存储并逐条查询Terms能够极大的减少数据的查询量和查询时间，仅在最后获取具体的数据Document时才会去读取磁盘文件信息，从整体上提高数据的搜索速度。

同时，当数据量极大的情况下就会发生一个key对应非常多的values（例如以性别进行索引时，如果有上千万条数据，而世界上只有男/女这样两个性别，则每个term对应的全部数据文档ID都会有至少百万个文档ID）。为了有效的将这些数据都存入内存中，系统会将这些文档ID进行压缩处理，并采用了一种基于bitmap的索引值存储方式，但与一般的bitmap不同，一般Bitmap的存储空间随着文档个数成线性增长。本系统将索引值按65535将索引ID进行划分，将索引ID除以65535得到商和余数，通过构造多个能存储65535的bitmap结构来存储索引ID。这样按照65535为界限分块，比如第一块所包含的文档id范围在0~65535之间，第二块的id范围是65536~131071，以此类推。再用<商，余数>的组合表示每一组id，这样每组里的id范围都在0~65535内了，在某一区间中如果不存在要存储的数据，那就不用创建该范围区域的bitmap结构，可以有效的将传统bitmap的存储空间随着文档个数成指数增长进行缩减，能够极大地减少存储空间，并将全部索引ID加载到内存中。例如下表格所示是某个指定term的文档ID集合：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1000 | 62101 | 131385 | 132052 | 191173 | 196600 | …… |

如果采用传统Bitmap的存储方式，即每一位上存储0/1表示该位是否存在，则以上需要在第100,62101,131385,132052,191173,196600等位上存储1，因为目前可知最大ID为196658，以此采用这种方式最少需要196600bits空间。但是采用本系统的存储方式后，只需要建立有数据存储区间的bitmap结构，例如以上表数据需要建立如下数据区间：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0~65535 | | 131072~196607 | | | | …… |
| 1000 | 62101 | 131385 | 132052 | 191173 | 196600 | …… |

可以看到只需要建立两个包含65535bits的bitmap空间，即131072bits。远远小于传统的bitmap结构需要占用的内存空间。

本系统的计算相似度模块是基于词典分词的文本相似度计算。在比较相似度的时候，第一步先做基于词典的分词，以实际数据为例(压盖 IN50-32-250 离心泵 IN50-32-250)将分为(压盖，IN50，32，250，离心泵，IN50，32，250),其中“-”为停用词符号不计算在分词中，同样的(砂轮 250×32×32)也被分词为(砂轮,250,32,32)，其中“×”为停用词符号不计算在分词中；第二步，去除重复单词并列出识别出的所有单词，例如本例中的所有单词为(压盖，IN50，32，250，离心泵，砂轮)；第三步，计算词频，即计算第二步得到的所有单词在每条数据中出现的次数，在本例中结果如下：压盖 IN50-32-250 离心泵 IN50-32-250

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 单词 | 压盖 | IN50 | 32 | 250 | 离心泵 | 砂轮 |
| 出现次数 | 1 | 2 | 2 | 2 | 1 | 0 |

砂轮 250×32×32

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 单词 | 压盖 | IN50 | 32 | 250 | 离心泵 | 砂轮 |
| 出现次数 | 0 | 0 | 2 | 1 | 0 | 1 |

第四步，构造词频向量，本例是[1,2,2,2,1,0]和[0,0,2,1,0,1]，这里可以将原始的文本数据映射为多维空间中的向量，并且用这两个向量间的夹角大小来表示向量的相近程度，以此来计算两条文本数据的相似程度；

计算相似度公式

最后根据上一步得出的向量，运用文本相似度计算公式能算出这两个向量的夹角的COSINE值，

可以用这个COSINE值来表示这两个单词的相似度程度。即(压盖 IN50-32-250 离心泵 IN50-32-250)与(砂轮 250×32×32)的相似度为