关键词分类——代码说明文档

参赛id：stdcoutzyx

源码分为三个主要部分，其路径与功能如表1所示：

表 1 源码各部分路径及其作用

|  |  |
| --- | --- |
| 包名 | 作用 |
| com.zyx.conf  com.zyx.exfeature  com.zyx.svm | 自己实现的代码，完成分类算法的总流程。算法流程见《算法说明文档》。 |
| de.bwaldvogel.liblinear | liblinear的Java程序包，是svm模型的训练与预测的具体实现，是外部引用的程序包 |
| org.wltea.analyzer.cfg  org.wltea.analyzer.core  org.wltea.analyzer.dic | IKAnalyzer分词程序包，是中文分词算法的具体实现，是外部引用的程序包。 |

下面首先对外部引用的程序包的使用进行简要的介绍，然后再对自己实现的代码进行详细阐述。

# liblinear程序包

在de.bwaldvogel.liblinear包下有19个类，但是调用svm进行训练预测的流程却只需要几个主要实现类即可。表2介绍了使用到的类的名称及它们各自的作用。

表 2 使用到的liblinear类的功能的简要介绍

|  |  |
| --- | --- |
| 类名 | 作用 |
| Model.java | 该类定义了svm模型的各项参数，用于保存算法训练得到的模型。 |
| Parameter.java | 该类定义了训练svm模型时的各项参数，在训练模型时作为输入。 |
| Problem.java | 该类定义了训练集数据在内存中的存储方式，在训练模型时作为输入。 |
| Train.java | 该类通过调用Linear.java中的Train方法实现了训练模型的流程，但在本方案中，主要重载了其中的readProblem方法，用于将hdfs上的训练集读取到内存中，原来的readProblem方法只能读取本地文件。 |
| Linear.java | 该类完成了训练模型和预测结果的主要流程，通过调用Linear.Train方法与Linear.Predict方法完成。 |

# IKAnalyzer程序包

关于中文分词程序包的调用，比较简单，只调用了其中一个类，但需要注意其配置文件与外部词表的存放位置。IKAnalyzer包下共分为三个子包，在使用中主要用到org.wltea.analyzer.core下的部分程序文件。但是由于中文分词算法的词表非常重要，所以此处除了对用到的程序文件叙述以外，还对各个词典文件所在的位置与其功能以及其路径在何处制定等做了介绍。如表3、表4、表5所示。

表 3 org.wltea.analyzer.core包下使用到的程序文件的功能说明

|  |  |
| --- | --- |
| 文件名称 | 作用 |
| IKSegmenter.java | 中文分词的接口，调用该类完成中文分词。 |
| Lexeme.java | 中文分词的返回结果，用于存储分词后的各个词语。 |

表 4 org.wltea.analyzer.cfg包下使用到的程序文件与词典文件的说明

|  |  |
| --- | --- |
| 文件名称 | 作用 |
| DefaultConfig.java | 用于获取配置文件中的词典位置，并进行读取。 |
| IKAnalyzer.cfg.xml | 配置文件，指明了外部词表与停用词表的位置。 |
| stopword.dic | 停用词表，改变位置时需改动IKAnalyzer.cfg.xml中的指向。 |
| ext.dic | 外部词表，改变位置时需改动IKAnalyzer.cfg.xml中的指向。 |

表 5 org.wltea.analyzer.dic包下的词典文件的说明

|  |  |
| --- | --- |
| 文件名称 | 作用 |
| main2012.dic | 分词算法的主词表，路径在org.wltea.analyzer.cfg.DefaultConfig.java中直接定义，如果变动位置需进行改动。 |
| quantifier.dic | 量词词表，路径在org.wltea.analyzer.cfg.DefaultConfig.java中直接定义，如果变动位置需进行改动。 |

# 总流程实现代码详述

上面简要介绍了liblinear程序包与IKAnalyzer程序包的主要接口的功能，下面将按照流程的顺序逐一介绍各个流程的具体实现。由于程序是在hadoop平台上实现，在描述过程中除了对类的功能进行描述外，还将对每个map-reduce过程的输入输出的数据格式进行充分的说明。

## com.zyx.exfeature

首先使用的是com.zyx.exfeature包下的程序，该包的主要功能是对数据进行预处理，使之形成符合liblinear程序包输入格式的数据。预处理过程包括中文分词、向量化两个步骤，中文分词步骤由KWSeg.java完成，向量化步骤由Count.java、IndexData.java完成。在该包下还有一个类ExFeature，它的作用是调用Count与IndexData中的Map-Reduce过程，从而将向量化步骤的两个部分合并。

对于KWSeg类，完成中文分词，其输入文件只有keyword\_class.txt文件，该文件由(关键词,类别)键值对组成，其中对于已经标记的关键词，其类别值为1-33，对于没有标记的，其类别值为-。KWSeg对所有关键词均进行了分词处理，对已分类的关键词，输出(类别,分词结果)键值对；对未分类的关键词，输出(-,关键词:分词结果)。区别对待的原因在于，一是区别训练集与测试集，二是预测时需要知道赋予某个分词结果确定的向量某个类别值时，其原词语是什么。

Count类用于统计训练集中各个词语的出现次数。其输入为KWSeg类的输出，但只会统计(类别,分词结果)键值对而忽略(-,关键词:分词结果)。输出为(词语,出现次数)键值对，其中词语与关键词不同，是关键词经过分词后形成的词语。

IndexData类用于将分词结果转化为向量格式。它的输入为KWSeg类的输出，并以Count类的输出为配置输入（**这里所说的配置输入是指在hadoop过程中并不直接参加map-reduce过程，而在setup中被读入，作为每个map的公有数据，下同）**。在读入Count类的输出文件时，将词语所在行的行号作为该词语的索引值。该类有两个job，对应两个输出，对于(类别,分词结果)，输出到训练数据文件夹下，格式为(类别,向量)；对于(-,关键词:分词结果)，输出到测试数据文件夹下，格式为(关键词,向量)。其中向量格式为多个index:value 键值对，键值对之间用空格分开，index是词语的索引值，该值由小到大排列，value值是该index的权重，该值可以直接简单设为1，也可以为该index所代表的的词语在对应关键词的分词结果中出现的次数（实验表明，value值为1可以提高正确率）。

## com.zyx.svm

完成预处理后，就可以使用svm算法对数据进行分类了。正如《算法说明文档》中所述，由于训练数据较大，不能在job的setup过程中将所有数据都读入训练一个模型，所以本解决方案采用分组训练多个分类器对所有数据分别进行分类，然后使用投票法得到最终的结果。在该包下，GroupSVMClassify类训练多个分类器分别对所有数据进行分类；VoteResult完成投票法确定最终预测类别；

GroupSVMClassify类首先将所有类别分成多个组，分组信息定义在源代码中main函数上方的一个二维数组中。对于每个组-组对，执行一个job，每个job的输入均为上面IndexData类生成的测试集，并以IndexData类生成的训练集作为配置输入，对于配置输入，只读取该组-组对所包含的类别的数据到内存中（该处代码实现在de.bwaldvogel.liblinear.Train中的“readProblem(LineReader line\_in, double bias, int labels1[], int labels2[])”方法下），训练一个分类器，对输入的测试集进行预测，对输入的每个(关键词,向量)键值对，得到(关键词,类别)输出。这样，对于测试集输入，得到分组数(分组数-1)/2个预测结果，对于现有的jar包，分组数=8。

VoteResult的输入即为GroupSVMClassify的输出，通过统计某个词语的所有预测值，得到最多次数的预测值，并以之为最终预测值。VoteResult的输出为(关键词,类别)，VoteResult的输出就可以使用evaluate命令进行测试。