# 文本推荐 - 新闻推荐系统示例

新闻推荐系统研究是当前新媒体时代的热点。在过去，电视、报纸、广播等传统媒体主要是通过人工给受众推荐信息；后来，互联网门户网站和搜索引擎兴起，更多的信息通过热门推荐的方式传递。现在，基于个性化的算法推荐和信息流越来越成为主流。

与电商平台的商品推荐类似，新闻推荐系统是提供个性化的新闻信心，而非商品。所以在算法层面，很多能够在电商平台使用的算法都可以用在新闻推荐中，比如基于用户行为的推荐算法。其中与商品推荐最大的不同在于，新闻推荐系统需要可以采用语义分析和语义理解的方法提供推荐，比如用户在浏览新闻A时推荐与这则新闻主题类似的其他新闻。

## 数据形态

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 唯一编号 | 类别 | 时间 | 浏览次数 | 跟帖次数 | 标题 | 正文 |
| 100000 | 国际要闻 | 2018-12-05 21:05 | 119 | 23 | 国际锐评｜中葡关系翻开新篇章 | 应葡萄牙总统德索萨邀请，中国国家主席习近平星期二（4日）开始对葡萄牙。。。 |

## 热点新闻

新闻的热度值计算，并写入数据库

观看次数\*0.4 + 评论次数 \* 0.5 - 天数时间差\*0.1

import org.apache.spark.sql.{DataFrame, Dataset, Row, SparkSession}  
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}

import spark.implicits.\_  
  
def readCsv(path: String) = {  
 val newSchema = StructType(*Array*(  
 StructField("id", StringType, true),  
 StructField("cate", StringType, true),  
 StructField("new\_time", StringType, true),  
 StructField("seenum", IntegerType, true),  
 StructField("disnum", IntegerType, true),  
 StructField("title", StringType, true)))  
  
 val data = spark.read.  
 option("header", false).  
 schema(newSchema).  
 csv(path)  
 data.as[New]  
}  
  
def calHotValue(data: Dataset[New],  
 seeValue: Float = 0.4f,  
 disValue: Float =0.5f,  
 decayValue: Float = 0.1f) = {  
 val now\_time = DateUtil.*getTime*()  
 data.map(line => {  
 val diff\_days = DateUtil.*dateTimeDiffInDay*(now\_time, line.new\_time)  
 val hot\_value = line.seenum \* seeValue + line.disnum \* disValue - diff\_days \* decayValue  
 (line.id, line.cate, hot\_value)  
 }).toDF("id","cate","hot\_value")  
}

输出新闻ID和热度值:

+------+--------+--------------------+-------------+

| id | new\_id | new\_hot | new\_cate\_id |

+------+--------+--------------------+-------------+

| 1083 | 100000 | 124.2 | 3 |

| 1084 | 100001 | 51.00000000000001 | 3 |

| 1085 | 100002 | 73.10000000000001 | 3 |

| 1086 | 100003 | 164.1 | 3 |

| 1087 | 100004 | 179.5 | 3 |

## 关键词提取

采用TFIDF算法进行关键词提取。TF-IDF是一种用于文本检索和文本转换的常用加权技术，其主要思想就是：如果某个词或者短语在一篇文中中出现的频率TF高，并且在其他文章中很少出现，就认为该词或者短语具有很好的主题区分功能。这个功能同样也适合与文本分类问题上。

假如一篇文件的总词语数是100个，而词语“手机”出现了8次，那么“手机”一词在该文件中的词频就是8/100=0.08。一个计算文件频率 (DF) 的方法是测定有多少份文件出现过“手机”一词，然后除以文件集里包含的文件总数。所以，如果“手机”一词在1,000份文件出现过，而文件总数是10,000,000份的话，其逆向文件频率就是 lg(10,000,000 / 1,000)=4。最后的TF-IDF的分数为0.08 \* 4=0.32[[1]](#footnote-1)。

import com.huaban.analysis.jieba.JiebaSegmenter  
import scala.io.Source  
def jieba\_analysis(input: String, path: String): List[String] = {  
 val stopwordPath = path + "stop\_words.txt"  
 val stopWords = Source.*fromFile*(stopwordPath).getLines().toArray  
 val segment = new JiebaSegmenter  
 val seg\_list = segment.sentenceProcess(input)  
 if (stopWords != null) {  
 for(sw <- stopWords){  
 if(seg\_list.contains(sw)) seg\_list.remove(sw)  
 }  
 }else *print*("Stopwords is null")  
 seg\_list.asInstanceOf[List[String]]  
}  
  
*/\*\*  
 \* 获取关键词  
 \** ***@param data*** *\** ***@param path*** *\** ***@return*** *\*/*def getKeywords(data: Dataset[New], path: String) = {  
 val splitWords = data.map(file => {  
 (file.id, jieba\_analysis(file.title, path))  
 }).toDF("id","title")  
  
 import org.apache.spark.ml.feature.{CountVectorizer, CountVectorizerModel, IDF}  
 val cvModel: CountVectorizerModel = new CountVectorizer().  
 setInputCol("title").  
 setOutputCol("rawFeatures").  
 setVocabSize(1000000). //向量长度  
 setMinDF(2). //词汇出现次数必须大于等于2  
 fit(splitWords)  
 val cvDf = cvModel.transform(splitWords)  
  
 val idf = new IDF().  
 setInputCol("rawFeatures").  
 setOutputCol("features")  
  
 val idfModel = idf.fit(cvDf)  
 val idfData = idfModel.transform(cvDf)  
  
 val voc= cvModel.vocabulary  
  
 import scala.collection.mutable.ArrayBuffer  
 val keywordsDF = idfData.select("id","features").*rdd*.  
 map { x => {  
 val name = x.getAs[String](0)  
 //idf结果以稀疏矩阵保存  
 val v = x.getAs[org.apache.spark.ml.linalg.SparseVector](1)  
 var arrW = ArrayBuffer[String]()  
 var arrV = ArrayBuffer[Double]()  
 v.foreachActive((index:Int,value:Double)=>{  
 arrW += voc(index)  
 arrV += value  
 })  
 //根据idf值从大到小排序  
 (name, (arrW zip arrV).toList.sortBy(-\_.\_2).toArray.map(\_.\_1).mkString(","))  
 // (name, (arrW zip arrV).toList.sortBy(-\_.\_2))  
 }  
 }  
 keywordsDF  
}

输出新闻ID和对应的关键词：

100000 中葡,葡萄牙,合作,中欧,关系,中国,国家,共建,锐评,首访  
100001 葡萄牙,中国,合唱团,葡萄牙人,文化,教学,夫妇,夫妇俩,大学,传播  
100002 磋商,经贸,加征,会晤,关税,共识,团队,达成,责编,进晚餐  
100003 俄罗斯,刻赤,总统,海峡,俄方,部长,否认,亚速海,秘书,新闻

## 相似新闻

### 基于标签相似的新闻

通过上一步的关键字提取，我们得到了每一篇新闻标题的相关关键词，这些关键词就是新闻的标签。根据这些关键词列表就可以进行两两对比计算，找出标签相似的新闻。

def getSimWithTag(data: RDD[(String, String)], sp: String = ",") ={  
 val bc\_data = spark.sparkContext.broadcast(data.collect())  
  
 val sim = data.flatMap(rdd\_words => bc\_data.value.map(id\_words => {  
 val bc\_id = id\_words.\_1  
 val bc\_words = id\_words.\_2  
 (bc\_id, rdd\_words.\_1, rdd\_words.\_2, bc\_words)  
 })).filter(x=>x.\_1!=x.\_2).  
 map{case(id1, id2, words1, words2) =>  
 val wordList1 = words1.split(sp)  
 val wordList2 = words2.split(sp)  
 val corr = wordList1.intersect(wordList2).size.toDouble /  
 wordList1.union(wordList2).distinct.size  
 (id1, id2, corr)  
 }  
 sim  
}

输出新闻ID、相似度新闻ID和相似值：

+--------+-------------+------------+-----------------+

| id | new\_id\_base | new\_id\_sim | new\_correlation |

+--------+-------------+------------+-----------------+

| 194677 | 100000 | 100001 | 0.06 |

| 194678 | 100000 | 100022 | 0.18 |

| 194679 | 100000 | 100045 | 0.06 |

| 194680 | 100000 | 100046 | 0.17 |

### 基于用户行为的相似度新闻

基于用户行为的相似度新闻，最简单的就是协同过滤。显然该算法与电商平台和线上商超的商品推荐类似，所以不再赘述。

## 页面展示

主页登录： <http://172.16.20.26:8001/>

提供三个测试用户：

图片包含 屏幕截图

描述已自动生成

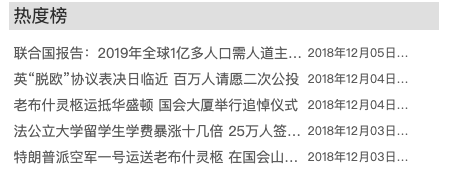
用户冷启动的主题选择（可以不选跳过）：

图片包含 屏幕截图

描述已自动生成

### 推荐场景

**热点新闻**



**主题新闻**

点击不同的主题，按照时间排序，最新的新闻排在前列。

**个性化推荐**

图片包含 屏幕截图

描述已自动生成

**个性化推荐计算逻辑：**

首先判断用户是否有浏览记录

如果有该用户的浏览记录 则从浏览的新闻获取相似的新闻返回

判断用户最近浏览的新闻是否够10个，如果够的话 取 top 10，每个取两个相似

如果不够10个 每个取 20/真实个数 +1 相似

遍历最近浏览的N篇新闻，每篇新闻取num篇相似新闻

如果该用户没有浏览记录，即该用户是第一次进入系统且没有选择任何标签 返回热度榜单数据的20-40个新闻。

1. <https://www.cnblogs.com/ywl925/p/3275878.html> [↑](#footnote-ref-1)