**数据分析**

**频繁模式挖掘**

[频繁模式简介(大白话)](https://blog.csdn.net/orthocenterchocolate/article/details/39297365)

[频繁模式详解(包括Apriore和FP-growth算法)](https://blog.csdn.net/weixin_56516468/article/details/121479149)

[PFP图示](https://blog.csdn.net/weixin_40532625/article/details/116047701)

[PFP算法简介](https://blog.csdn.net/yuankeLi/article/details/104599903)

* 什么叫频繁模式？就是经常一起出现的模式
* 比如在超市的销售记录里，常常会发现牛奶和面包是经常被一起购买的，那么牛奶和面包这两个item经常一起现在销售记录中，所以在这里牛奶和面包是一个可以看成是一个频繁模式，当然单独看牛奶，单独看面包，也是频繁模式
* 频繁模式有什么用？研究频繁模式的目的是得到关联规则和其他的联系，并在实际中应用这些规则和联系。比如，频繁地同时出现在交易数据集中的商品（比如牛奶和面包）的集合是频繁项集；频繁的出现的一个购买顺序（先买笔记本，再买杀毒软件）是频繁子序列。
* Apriori算法
* 原理

K+1-项频繁集的任意子集必定也是频繁的

比如，如果牛奶和面包频繁一起出现，那么牛奶也频繁出现、面包也频繁出现

实现过程分为两步，一步是连接，一步是剪枝

用K-项频繁集来生成K+1-项候选集，进一步得到频繁项集，其过程中也能够得到关联规则

* 评价

优点：原理简单，易于理解；能够得到频繁项集和关联规则

缺点：需要多次扫描数据集；产生大量频繁项集

* FP-growth算法（Frequency Pattern）
* 原理

FP-growth算法基于Apriori原理，通过将数据集存储在FP树上发现频繁项集

* 评价

优点：巧妙的利用了树结构改进了Apriori算法的IO瓶颈，比Apriori算法快一个数量级。其原因为：简洁的数据结构；内存数据结构以空间换时间；没有候选测试；没有重复的数据库扫描

缺点：不能发现数据之间的关联规则

* PFP算法（Parallel FP-Growth）
* 原理

PFP算法基于FP-growth算法原理，将挖掘数据集的任务分配在多个计算机上，每个计算机相互独立并行的计算与通信

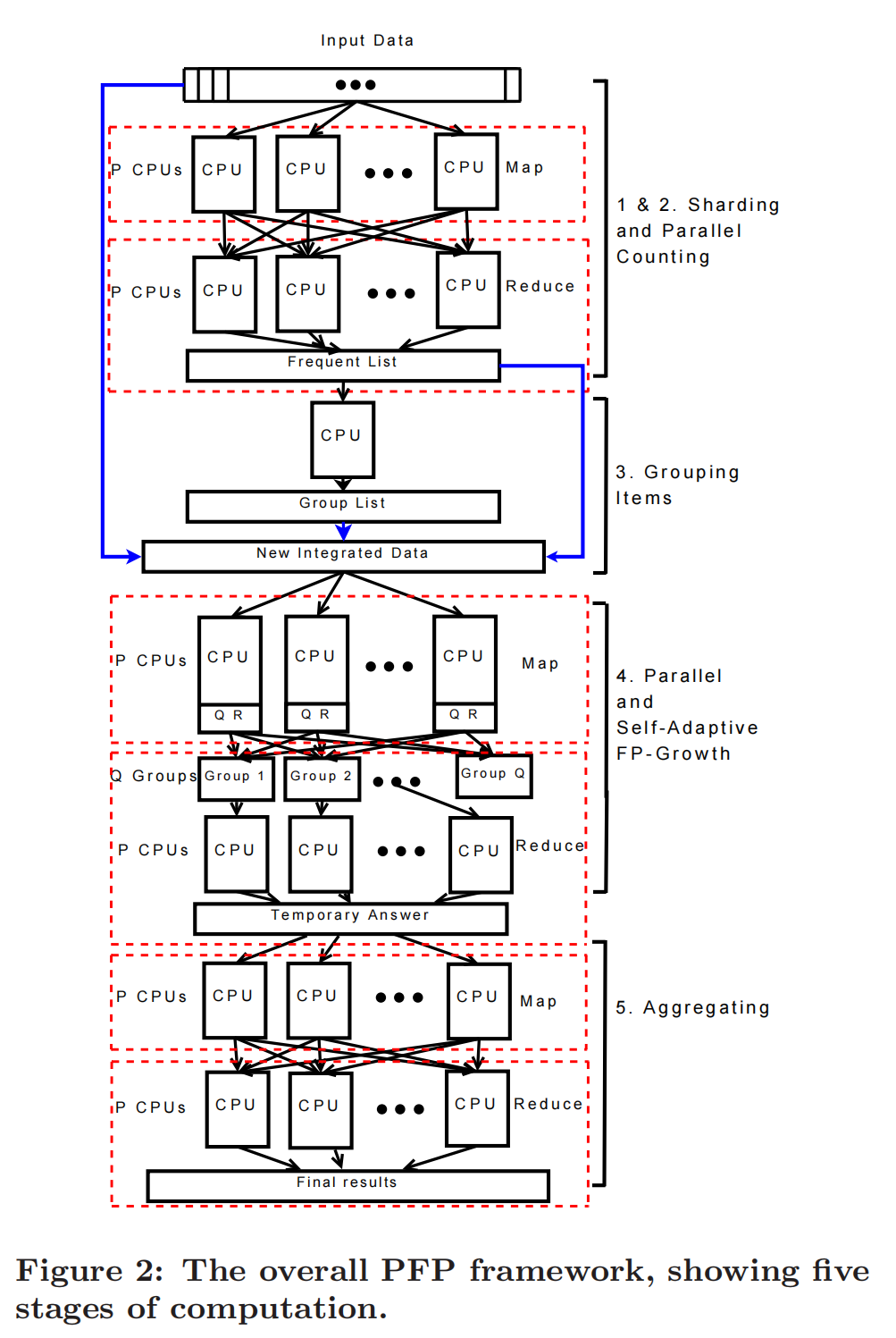
* 评价

直接使用FP-growth算法所占内存的使用和计算成本代价过大，瓶颈主要在与建立FP树和计算FP树上的频繁项集上，因此这里是PFP算法主要利用分布式优化的地方

PFP算法实现数据量和挖掘速度呈线性关系，适合用于数据量大的数据集，本lab采用该算法

**PFP**

**流程图**

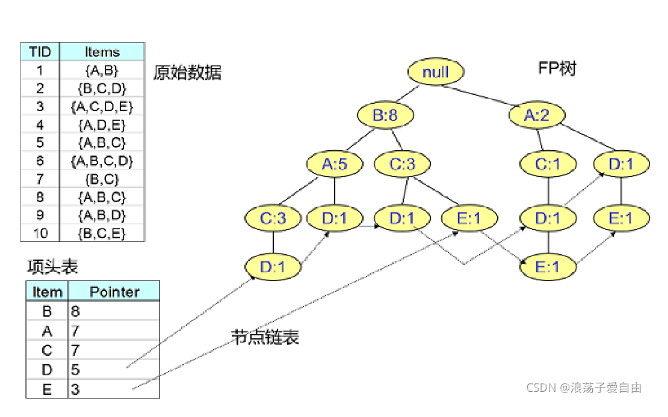


**流程概述**

1. 使用wordcouont的经典算法统计词频
2. 利用词频构建一个合理分配算力的哈希规则
3. 利用第2步所使用的的哈希规则选择合适的前缀进行数据的切分，在reduce中使用切分的数据分别构建FP树运算得到结果
4. 将第3步得到的结果按照自己的需求进行聚合修饰

**FP树简介**

1. 首先统计词频，将每一个句子中的词汇去重，删去词频低于阈值的词最后按照词频排序
2. 建立一棵字典树，并且使用链表纪录每一个词汇出现在字典树上的位置



1. 最后按照词频最小到大的统计这个词出现的频繁模式，统计的方法就是重新建立之和这个词相关的字典树，按照阈值进行统计

|  |
| --- |
| Python # -\*- coding: utf-8 -\*- # reference from https://blog.csdn.net/weixin\_56516468/article/details/121479149 import operator class treeNode:  def \_\_init\_\_(self,nameValue,numOccur,parentNode):  self.name=nameValue #节点名  self.count=numOccur #节点元素出现次数  self.nodeLink=None #存放节点链表中，与该节点相连的下一个元素  self.parent=parentNode  self.children={} #用于存放节点的子节点,value为子节点名    def inc(self,numOccur):  self.count+=numOccur    def disp(self,ind=1):  print(" "\*ind,self.name,self.count) #输出一行节点名和节点元素数，缩进表示该行节点所处树的深度  for child in self.children.values():  child.disp(ind+1) #对于子节点，深度+1   # 构造FP树 # dataSet为字典类型，表示探索频繁项集的数据集,keys为各项集，values为各项集在数据集中出现的次数 # minSup为最小支持度，构造FP树的第一步是计算数据集各元素的支持度，选择满足最小支持度的元素进入下一步 def createTree(dataSet,minSup=1):  headerTable={}    #遍历各项集,统计数据集中各元素的出现次数  for key in dataSet.keys():  for item in key:  headerTable[item]=headerTable.get(item,0)+dataSet[key]     #遍历各元素，删除不满足最小支持度的元素  for key in list(headerTable.keys()):  if headerTable[key]<minSup:  del headerTable[key]  freqItemSet=set(headerTable.keys())    #若没有元素满足最小支持度要求，返回None，结束函数  if len(freqItemSet)==0:  return None,None  for key in headerTable.keys():  headerTable[key]=[headerTable[key],None] #[元素出现次数，\*\*指向每种项集第一个元素项的指针\*\*]  retTree=treeNode("Null Set",1,None) #初始化FP树的顶端节点    for tranSet,count in dataSet.items():  localD={} #存放每次循环中的频繁元素及其出现次数，便于利用全局出现次数对各项集元素进行项集内排序  for item in tranSet:  if item in freqItemSet:  localD[item]=headerTable[item][0]  if len(localD)>0:  orderedItems=[v[0] for v in sorted(localD.items(),key=operator.itemgetter(1),reverse=True)] #根据元素全局出现次数对每个项集(tranSet)中的元素进行排序  updateTree(orderedItems,retTree,headerTable,count) #使用排序后的项集对树进行填充  return retTree,headerTable     #树的更新函数 #items为按出现次数排序后的项集，是待更新到树中的项集；count为items项集在数据集中的出现次数 #inTree为待被更新的树；headTable为头指针表，存放满足最小支持度要求的所有元素 def updateTree(items,inTree,headerTable,count):  #若项集items当前最频繁的元素在已有树的子节点中，则直接增加树子节点的计数值，增加值为items[0]的出现次数  if items[0] in inTree.children:   inTree.children[items[0]].inc(count)  else:#若项集items当前最频繁的元素不在已有树的子节点中（即，树分支不存在），则通过treeNode类新增一个子节点  inTree.children[items[0]]=treeNode(items[0],count,inTree)  #若新增节点后表头表中没有此元素，则将该新增节点作为表头元素加入表头表  if headerTable[items[0]][1]==None:   headerTable[items[0]][1]=inTree.children[items[0]]  else:#若新增节点后表头表中有此元素，则更新该元素的链表，即，在该元素链表末尾增加该元素  updateHeader(headerTable[items[0]][1],inTree.children[items[0]])  #对于项集items元素个数多于1的情况，对剩下的元素迭代updateTree  if len(items)>1:  updateTree(items[1::],inTree.children[items[0]],headerTable,count)     #元素链表更新函数 #nodeToTest为待被更新的元素链表的头部 #targetNode为待加入到元素链表的元素节点 def updateHeader(nodeToTest,targetNode):  #若待被更新的元素链表当前元素的下一个元素不为空，则一直迭代寻找该元素链表的末位元素  while nodeToTest.nodeLink!=None:   nodeToTest=nodeToTest.nodeLink #类似撸绳子，从首位一个一个逐渐撸到末位  #找到该元素链表的末尾元素后，在此元素后追加targetNode为该元素链表的新末尾元素  nodeToTest.nodeLink=targetNode #加载简单数据集 def loadSimpDat():  simpDat = [['r', 'z', 'h', 'j', 'p'],  ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],  ['z'],  ['r', 'x', 'n', 'o', 's'],  ['y', 'r', 'x', 'z', 'q', 't', 'p'],  ['y', 'z', 'x', 'e', 'q', 's', 't', 'm']]  return simpDat   #将列表格式的数据集转化为字典格式 def createInitSet(dataSet):  retDict={}  for trans in dataSet:  retDict[frozenset(trans)]=1  return retDict   #由叶节点回溯该叶节点所在的整条路径 #leafNode为叶节点，treeNode格式；prefixPath为该叶节点的前缀路径集合，列表格式，在调用该函数前注意prefixPath的已有内容 def ascendTree(leafNode,prefixPath):  if leafNode.parent!=None:  prefixPath.append(leafNode.name)  ascendTree(leafNode.parent,prefixPath)   #获得指定元素的条件模式基 #basePat为指定元素；treeNode为指定元素链表的第一个元素节点，如指定"r"元素，则treeNode为r元素链表的第一个r节点 def findPrefixPath(basePat,treeNode):  condPats={} #存放指定元素的条件模式基  while treeNode!=None: #当元素链表指向的节点不为空时（即，尚未遍历完指定元素的链表时）  prefixPath=[]  ascendTree(treeNode,prefixPath) #回溯该元素当前节点的前缀路径  if len(prefixPath)>1:  condPats[frozenset(prefixPath[1:])]=treeNode.count #构造该元素当前节点的条件模式基  treeNode=treeNode.nodeLink #指向该元素链表的下一个元素  return condPats   #有FP树挖掘频繁项集 #inTree: 构建好的整个数据集的FP树 #headerTable: FP树的头指针表 #minSup: 最小支持度，用于构建条件FP树 #preFix: 新增频繁项集的缓存表，set([])格式 #freqItemList: 频繁项集集合，list格式   def mineTree(inTree,headerTable,minSup,preFix,freqItemList):  #按头指针表中元素出现次数升序排序，即，从头指针表底端开始寻找频繁项集  bigL=[v[0] for v in sorted(headerTable.items(),key=lambda p:p[1][0])]   for basePat in bigL:  #将当前深度的频繁项追加到已有频繁项集中，然后将此频繁项集追加到频繁项集列表中  newFreqSet=preFix.copy()  newFreqSet.add(basePat)  print("freqItemList add newFreqSet",newFreqSet)  freqItemList.append(newFreqSet)  #获取当前频繁项的条件模式基  condPatBases=findPrefixPath(basePat,headerTable[basePat][1])  #利用当前频繁项的条件模式基构建条件FP树  myCondTree,myHead=createTree(condPatBases,minSup)  #迭代，直到当前频繁项的条件FP树为空  if myHead!=None:  mineTree(myCondTree,myHead,minSup,newFreqSet,freqItemList)   simpDat=loadSimpDat() dataSet=createInitSet(simpDat) myFPtree1,myHeaderTab1=createTree(dataSet,minSup=3) myFPtree1.disp(),myHeaderTab1 freqItems=[] mineTree(myFPtree1,myHeaderTab1,3,set([]),freqItems) freqItems |

**流程详解**

1. 进行wordcount，Ti就是一个输入进来的句子，ai是句子中的词语，这里获得一个统计有词语对应出现次数的字典.（注意这里的ai+C的含义就是一个字典），将这一步生成的字典记为F\_list

|  |
| --- |
| Plain Text Procedure: Mapper(key, value=Ti) foreach item ai in Ti do  Call Output(<ai,′ 1′>); end  # 这里S(ai)指的是由1构成的一个列表 Procedure: Reducer(key=ai, value=S(ai)) C ← 0; foreach item ′1′ in S(ai) do  C ← C + 1; end Call Output(<null, ai + C>); |

1. 因为建立树和计算频繁项集是该问题内存和时间上的瓶颈，我们这一步利用F\_list规划一个哈希规则，用于将不同的任务分配到不同的主机上运行，我们希望可以每一个主机独立的完成任务，在Reduce之前不需要交换信息。我们将这个哈希表定义为G\_list，由于F\_list和G\_list一般不会太大，这里仅需要一个主机工作即可
2. 根据G\_list产生的哈希规则分布式的处理FP树和频繁集
3. Mapper部分

* 输入是一个一个的句子也就是原数据，输出的key是在G\_list中的组编号，值也是一个字符串，这个Map的目的是找到每一个在句子中的词汇最后出现的位置并输出对应的组编号和前缀。具体操作如下
* 这一个Mapper的首先加载G\_list因为我们需要对hash表做出内部修改操作（不影响全局的hash表）所以先加载出一个H
* 遍历句子的每一个词汇，得到hash值，如果不是空值那么将这个键从H这个hash表中删除（即以后在出现这个词汇的时候hash会给出NULL），并输出一个键是hash值（组编号）值是的字符串前缀。

|  |
| --- |
| Plain Text Procedure: Mapper(key, value=Ti) Load G-List; Generate Hash Table H from G-List; a[] ← Split(Ti); for j = |Ti| − 1 to 0 do  HashNum ← getHashNum(H, a[j]);  if HashNum != Null then  Delete all pairs which hash value is HashNum in H;  Call Output(<HashNum, a[0] + a[1] + ... + a[j]>);  end end |

1. Reduce部分

* 输入是组编号和字符串列表组成的元组，其中字符串列表就是在Map中得到的若干句子的前缀，输出的时候键是NULL，值是词汇列表（即挖掘出来的频繁模式）和这个模式的出现次数组成的元组
* 在这个分布式系统内部建立一棵FP树
* 对于每一个在gid这一组中的词汇做一次向上查询FP树，利用堆得到出现次数最多的，以这个词汇为底的频繁模式。
* 遍历这个堆输出每一个频繁模式

|  |
| --- |
| Plain Text Procedure: Reducer(key=gid,value=DB[gid]) Load G\_List; nowGroup ← G\_List[gid]; LocalFPtree ← clear; foreach Ti in DB(gid) do  Call insert\_build\_fp\_tree(LocalFPtree,Ti); end foreach ai in nowGroup do  Define and clear a size K max heap : HP;  Call TopKFPGrowth(LocalFPtree, ai,HP);  foreach vi in HP do  Call Output(<null, vi + supp(vi)>);  end end |

1. 得到若干频繁模式后的数据处理（这里较灵活，我介绍下论文的处理方式和我们的处理方式）

* 论文希望得到每一个词的对应的一些频繁模式所以论文的MapReduce代码如下，即一对多的关系分析。

|  |
| --- |
| Plain Text Procedure: Mapper(key, value=v + supp(v)) foreach item ai in v do  Call Output(<ai, v + supp(v)>); end Procedure: Reducer(key=ai, value=S(v + supp(v))) Define and clear a size K max heap : HP; foreach pattern v in v + supp(v) do  if |HP| < K then  insert v + supp(v) into HP;  else  if supp(HP[0].v) < supp(v) then  delete top element in HP;  insert v + supp(v) into HP;  end  end end Call Output(<null, ai + HP>); |

* 我们通过调用spark自带的库，得到了若干个词汇组成的模式对应于一个词汇的频繁模式分析，将多对一的关系。大致的伪代码如下

|  |
| --- |
| SQL Procedure: Mapper(key, value=v + supp(v)) foreach item ai in v do  Call Output(<v-ai, ai + supp(v)>); end Procedure: Reducer(key=v, value=S(ai + supp(v+ai))) Define and clear a size K max heap : HP; foreach pattern ai in S(ai + supp(v+ai))) do  if |HP| < K then  insert ai + supp(v+ai) into HP;  else  if supp(HP[0].a+v) < supp(ai+v) then  delete top element in HP;  insert ai + supp(v+ai) into HP;  end  end end Call Output(<null, v + HP>); |

**代码实现**

|  |
| --- |
| Python from pyspark.ml.fpm import FPGrowth from pyspark import SparkContext from pyspark.sql import SparkSession sc = SparkContext(appName="FPGrowth") spark = SparkSession(sc) data = sc.textFile("/mydata/data/data2") transactions = data.map(lambda line: (line.strip().split(' '),)) df = transactions.toDF(["items",]) fpGrowth = FPGrowth(itemsCol="items", minSupport=4e-5, minConfidence=0.4) model = fpGrowth.fit(df) answer=model.associationRules from pyspark.sql.types import \* from pyspark.sql.functions import udf def str\_to\_arr(my\_list):  my\_list.sort()  return '+'.join([str(elem) for elem in my\_list])  str\_to\_arr\_udf = udf(str\_to\_arr,StringType()) answer = answer.withColumn("antecedent",str\_to\_arr\_udf(answer["antecedent"])) answer = answer.withColumn("consequent",str\_to\_arr\_udf(answer["consequent"])) answer.write.option("header",True).csv("/mydata/output/data\_support\_4e-5\_confidence\_0.4") spark.stop() |

**运行**

（大致输出结果）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| antecedent | consequent | confidence | lift | support |
| 印尼+排华+是 | 怎么回事 | 1 | 2628.836409 | 1.13127E-4 |
| 影帝+派出所长 | 当 | 1 | 4068.981572 | 5.79555E-5 |
| 华人+印尼 | 女 | 0.487976128 | 72.08915233 | 1.0901E-4 |
| 华人+印尼 | 图片 | 0.620502019 | 35.8099881 | 1.38615E-4 |
| 华人+印尼 | 残害 | 0.909864841 | 3932.161388 | 2.03256E-4 |
| 山西+组织部 | 公示 | 0.988347252 | 2669.897742 | 2.99325E-4 |
| 山西+组织部 | 省委 | 0.991843076 | 3081.291429 | 3.00384E-4 |
| 减+腹部 | 脂肪 | 0.733296214 | 5528.683173 | 5.16423E-5 |
| gp+免费 | 下载 | 0.670460561 | 32.72617649 | 4.96621E-5 |
| 国际+董事长+集团 | 华能 | 0.99792498 | 3822.669791 | 1.22577E-4 |
| 儿女 | 有 | 0.749398143 | 112.4776076 | 1.28165E-4 |
| 客户端 | 下载 | 0.450314977 | 21.98054331 | 4.3447E-5 |
| 富豪+年轻+是+最 | 的 | 1 | 21.25221732 | 4.34078E-5 |
| com+pcpop+pop+最新 | site | 1 | 207.9082842 | 1.40771E-4 |
| com+pcpop+pop+最新 | www | 1 | 57.65684216 | 1.40771E-4 |
| com+pcpop+pop+最新 | 价格 | 1 | 137.0205969 | 1.40771E-4 |
| com+pcpop+pop+最新 | channel | 1 | 3258.668796 | 1.40771E-4 |
| com+pcpop+pop+最新 | product | 1 | 3235.106178 | 1.40771E-4 |
| 间谍案 | 沈国放 | 0.943162393 | 1938.807828 | 8.65411E-5 |
| 梁山伯 | 祝英台 | 0.897363083 | 18433.23419 | 4.33686E-5 |
| 图片+审+美军+越南 | 战俘 | 1 | 676.589295 | 7.47578E-5 |
| 图片+审+美军+越南 | 女 | 1 | 147.7308996 | 7.47578E-5 |
| 苹果+视频 | 范冰冰 | 0.616130454 | 514.523292 | 5.48185E-5 |
| 走私 | 粮食 | 0.572647653 | 2701.916059 | 4.99954E-5 |
| 富豪+是+最+的 | 年轻 | 1 | 3936.457822 | 4.34078E-5 |
| 祖坟+蒋介石 | 风水 | 0.952774156 | 713.3454401 | 1.28949E-4 |
| 农村+新 | 建设 | 0.638660287 | 536.9066956 | 6.54254E-5 |
| 不雅 | 照 | 0.923774207 | 663.8508138 | 4.39567E-5 |
| 幽默+经典 | 笑话 | 0.931314957 | 1888.349238 | 4.04081E-5 |
| 唐国强 | 死亡 | 0.821177025 | 342.2584244 | 5.46322E-4 |
| 唐国强 | 出车祸 | 0.816314502 | 1471.803995 | 5.43087E-4 |
| 女+时尚 | 夏装 | 0.8772127 | 2062.506642 | 6.12101E-5 |
| 女+时尚 | 连衣裙 | 0.80303456 | 947.0616904 | 5.60341E-5 |
| 答案+高 | 考试题 | 0.980250293 | 1086.828173 | 9.82851E-5 |
| 捐赠+未 | 到 | 0.999719993 | 501.5531629 | 2.1E-4 |
| com+pop+product+site+最新 | channel | 1 | 3258.668796 | 1.40771E-4 |
| com+pop+product+site+最新 | www | 1 | 57.65684216 | 1.40771E-4 |
| com+pop+product+site+最新 | 价格 | 1 | 137.0205969 | 1.40771E-4 |
| com+pop+product+site+最新 | pcpop | 1 | 2959.366638 | 1.40771E-4 |
| 前兆+吗+地震+有 | 前 | 0.572976418 | 685.3128493 | 5.28775E-5 |
| 前兆+吗+地震+有 | 汶川 | 0.981091991 | 290.4489451 | 9.05407E-5 |
| 年轻+最+的 | 富豪 | 0.941518042 | 7414.208774 | 4.45057E-5 |
| 年轻+最+的 | 是 | 0.920364994 | 68.69780218 | 4.35058E-5 |
| 世界+八+星级+的+酒店 | 唯一 | 1 | 1984.000467 | 4.73074E-4 |
| 中+大营 | 清军 | 0.913451512 | 10767.3459 | 5.15247E-5 |
| 中+大营 | 女犯 | 0.758081335 | 1832.671292 | 4.27608E-5 |
| 中+大营 | 的 | 0.997219326 | 21.19312182 | 5.62497E-5 |
| 佟 | 艾 | 0.524921599 | 2898.187948 | 1.54241E-4 |
| 佟 | 大为 | 0.452925869 | 3304.912851 | 1.33086E-4 |
| 卸甲+见 | 龙 | 1 | 2056.14303 | 7.13268E-5 |