文本分析模型设计文档

[一、 业务规则库设计 2](#_Toc477264662)

[1. 业务规则库概述 2](#_Toc477264663)

[2. 业务规则库说明 3](#_Toc477264664)

[二、 文本分词模型设计 4](#_Toc477264665)

[1. 模型业务目标 4](#_Toc477264666)

[2. 模型逻辑设计 4](#_Toc477264667)

[2.1. 文本预处理逻辑 4](#_Toc477264668)

[2.2. 分词算法设计 5](#_Toc477264669)

[2.3. 词性标注算法设计 8](#_Toc477264670)

[3. 模型功能模块设计 9](#_Toc477264671)

[3.1. 文本预处理模块功能设计 9](#_Toc477264672)

[3.2. 分词模块功能设计 11](#_Toc477264673)

[4. 模型整体流程设计 20](#_Toc477264674)

[三、 热词提取模型设计 21](#_Toc477264675)

[1. 模型业务目标 21](#_Toc477264676)

[2. 模型逻辑设计 22](#_Toc477264677)

[2.1. 文本预处理逻辑 22](#_Toc477264678)

[2.2. 热词提取算法设计 22](#_Toc477264679)

[3. 模型功能模块设计 23](#_Toc477264680)

[3.1. 文本预处理模块功能设计 23](#_Toc477264681)

[3.2. 热词提取模块功能设计 24](#_Toc477264682)

[4. 模型整体流程设计 26](#_Toc477264683)

[四、 词云展示接口设计 27](#_Toc477264684)

[1. 接口业务目标 27](#_Toc477264685)

[2. 接口逻辑设计 28](#_Toc477264686)

[2.1. 设计原则 28](#_Toc477264687)

[2.2. 认证设计 28](#_Toc477264688)

[3. 接口功能规范设计 28](#_Toc477264689)

[3.1. 接口规范 28](#_Toc477264690)

[3.2. 协议 29](#_Toc477264691)

[3.3. 域名 29](#_Toc477264692)

[3.4. 版本 29](#_Toc477264693)

[3.5. 路径 29](#_Toc477264694)

[3.6. HTTP动词 29](#_Toc477264695)

[3.7. 返回结果 30](#_Toc477264696)

[3.8. 其他 30](#_Toc477264697)

[4. 接口整体流程设计 30](#_Toc477264698)

[五、 关联分析模型设计 31](#_Toc477264699)

[1. 模型业务目标 31](#_Toc477264700)

[2. 模型逻辑设计 31](#_Toc477264701)

[3. 模型功能模块设计 31](#_Toc477264702)

[4. 模型整体流程设计 31](#_Toc477264703)

[六、 情感分析模型设计 31](#_Toc477264704)

[1. 模型业务目标 31](#_Toc477264705)

[2. 模型逻辑设计 31](#_Toc477264706)

[3. 模型功能模块设计 31](#_Toc477264707)

[4. 模型整体流程设计 31](#_Toc477264708)

[七、 附录 32](#_Toc477264709)

# 业务规则库设计

## 业务规则库概述

业务规则库涉及分词库、新词词库、停用词库、噪声词库、实体名称库以及正负面情感标签库。

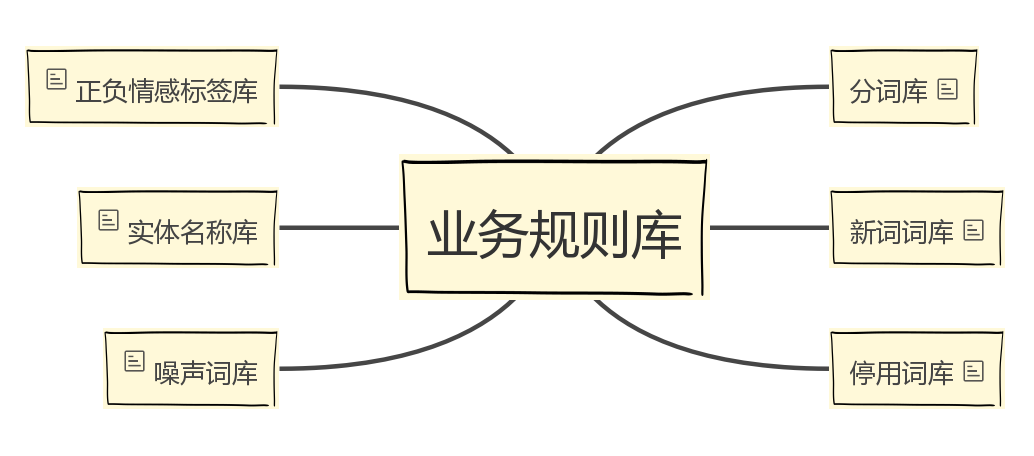


图1 业务规则概念图

## 业务规则库说明

**表1 业务规则库详情说明表**

|  |  |  |
| --- | --- | --- |
| **规则库** | **主要属性** | **规则库说明** |
| 分词库 | 词语、词频、词性 | 用于文本分词。每一个词语有固定的词频和词性。词性标注规范详见[附录1](#_附录)。 |
| 新词词库 | 词语、词频（可选）、词性（可选） | 用于及时补充新颖词汇，增强歧义纠错能力，提高文本分词的准确率和效果。由于新词出现快，消失也快，所以该词库需要频繁更新。词性标注规范详见[附录1](#_附录)。 |
| 停用词库 | 停用词语、停用符号 | 用于在处理文本之前或之后会自动过滤掉某些字或词。 |
| 噪声词库 | 噪声词语 | 用于在处理文本之前或之后，根据用户的需求，过滤掉某些字或词。由于噪声是用户在使用过程发现并提出的，并在处理文本前后及时清理噪声，该词库具备即时性、无本地保存性。 |
| 实体名称库 | 股票名称、股票所属公司全称、公司简称、公私募公司名称 | 用于关联分析。各个属性一一对应。 |
| 正负情感标签库 | 情感标签、情感分数 | 用于情感分析。对于情感分数，正数判断为积极，负数判断为消极。对于分数的绝对值作为积极或者消极程度。 |

# 文本分词模型设计

## 模型业务目标

基于python对资讯文章进行文本分词，根据来源于saas的数据，进行分词和词性标注。

## 模型逻辑设计

### 文本预处理逻辑

#### 文本解码

由于文本经过抓取、传输、存储这一系列过程，文本的编码可能发生一些变换。为了确保各类文本都可以被正确地切分，在文本分词之前，我们通常需要对即将分词的文本进行解码处理。常用的中文编码有以下两种：

1. UTF-8

Unicode TransformationFormat-8bit，允许含BOM，但通常不含BOM。是用以解决国际上字符的一种多字节编码，它对英文使用8位（即一个字节），中文使用24为（三个字节）来编码。UTF-8包含全世界所有国家需要用到的字符，是国际编码，通用性强。

1. GBK

国家标准GB2312基础上扩容后兼容GB2312的标准。GBK的文字编码是用双字节来表示的，即不论中、英文字符均使用双字节来表示，为了区分中文，将其最高位都设定成1。GBK包含全部中文字符，是国家编码，通用性比UTF8差，不过UTF8占用的数据库比GBD大。

因为UTF-8比GBK更具普遍性。所以我们优先使用UTF-8转码，假如转码失败，再用GBK转码。

#### 文本清洗

对于一个文本，数字，特殊符号以及英文单词是不需要再进行分词的，而且类似括号或者引号里面的小文本可能就是一个分词。为了分词速度更快，我们需要对文本进行清洗。

我们利用正则表达式匹配文本，找出所有的汉字序列并切分文本。剔除特殊符号、英文单词、数字。由此生成一组短语或短句子列表，便于进一步的分词行为。

### 分词算法设计

#### 基于统计信息的N-最短路径词语粗分模型

假设待分字串，其中，为单个的汉字，n为字串的长度，并满足。建立一个节点数为n+1的切分有向无环图G，各节点编号依次为。

通过以下两步建立G所有可能的词边：

1. 相邻节点之间建立有向边，边的长度值为，边对应的词默认为。
2. 如果是词表中的词，则节点之间建立有向边,边的长度值为，边对应的词为w。

假设表示词出现的概率。切分W的概率为

令，则就可以作为。粗分模型的目标就是求最小值的N钟切分结果，我们直接用DP算法求得最优解。

#### 基于字的生成式模型和区分式模型相结合的HMM

隐马尔科夫模型（HMM）是一个统计模型，我们不能直接观察模型的状态转换过程，而通过状态转移概率以及隐蔽随机函数去观察事件的随机过程。它的组成如下：

（1）状态值集合N

（2）观察值集合M

（3）转移概率矩阵

为从状态转向另一个状态的概率，其中：

（4）发射概率矩阵

为状态j观察到符号k的概率。那么

（5）初始状态分布

其中：

对于文本分词来说，状态值集合为（B，M，E，S）：{B：begin，M：middle， E：end， S：single}。每个状态代表的是该字在词语中的位置，B代表该字是词语中的起始字，M代表是词语中的中间字，E代表是词语中的结束字，S则代表是单字成词。观察值集合为就是待分词文本，借助初始状态，转移概率矩阵，发射概率矩阵这三个模型参数算出观察值集合最有可能的BMES状态序列。这五元的关系是通过一个Viterbi算法串接起来。

#### Viterbi算法

Viterbi是一个动态搜索最优状态序列。定义Viterbi变量是在时间t时，模型通过某路径到达并输出观察的最大概率：

1. ：表示在观察时刻t正处在状态j，且沿一条路径，产生出的最大概率。
2. ：表示一个状态值，每次计算的结果都是由上一个状态产生的。
3. ：表示在观察时刻t中所有的状态内最大的状态，所以它也是状态值。

由上面的解释可以得出这三个符号的数学表达式：

时间复杂度为

### 词性标注算法设计

我们采用基于HMM的词性标注方法，将词语划分成若干个等价类的策略，以类为单位进行参数估计。首先把所有具有相同的可能的词性的词语划分为一组，记作，（T为词性种类个数）。经过词语分组以后，对的处理方法如下：

其中，为词语l由词性标记j生成的概率，为中词语出现的次数，分母是在所有上的求和。并且，

其中，|L|是集合L中元素的个数。

一旦初始化完成后，HMM参数就可以利用前面介绍的Viterbi算法进行训练。

## 模型功能模块设计

### 文本预处理模块功能设计

文本预处理框架如下：



文本预处理模块，包括读取文本模块、文本转码模块和文本清洗模块等三个模块。各个模块功能设计如下：

1. 文本转换

功能描述：该模块主要用于对文本的编码格式进行转换。如果输入对象不为文本类型，那么优先转换为UTF-8编码，其次转换为GBK编码。

模块设计过程：通过Python的内建函数isinstance()实现对输入对象的类型判断。通过Python的内建函数decode()对输入对象进行解码。

文本转换str\_decode源代码如下：

|  |
| --- |
| def str\_decode(sentence):  if not isinstance(sentence, text\_type):  try:  sentence = sentence.decode('utf-8')  except UnicodeDecodeError:  sentence = sentence.decode('gbk', 'ignore')  return sentence |

1. 文本清洗

功能描述：该模块主要用于对文本的进行初步切分。并对无关字符（如数字、英文字符和特殊符号）进行剔除，将文本切割为词语或短句子列表。

模块设计过程：通过Python的re模块的正则表达式对文本进行切割和剔除。利用yield关键字,将函数变为一个词语生成器，逐个返回词语。

正则表达式源代码如下：

|  |
| --- |
| re\_han\_detail = re.compile("([\u4E00-\u9FD5]+)")  re\_skip\_detail = re.compile("([\.0-9]+|[a-zA-Z0-9]+)")  re\_han\_internal = re.compile("([\u4E00-\u9FD5a-zA-Z0-9+#&\.\_]+)")  re\_skip\_internal = re.compile("(\r\n|\s)")  re\_eng = re.compile("[a-zA-Z0-9]+")  re\_num = re.compile("[\.0-9]+")  re\_eng1 = re.compile('^[a-zA-Z0-9]$', re.U) |

文本清洗源代码如下：

|  |
| --- |
| re\_han = re\_han\_default  re\_skip = re\_skip\_default  cut\_block = self.\_\_cut\_DAG  # 先用正则对句子进行切分  blocks = re\_han.split(sentence)  for blk in blocks:  if not blk:  continue  if re\_han.match(blk): # re\_han匹配的串  for word in cut\_block(blk): # 进行具体的分词  yield word  else: # 按照re\_skip正则表对blk进行重新切分  tmp = re\_skip.split(blk) # 返回list  for x in tmp:  if re\_skip.match(x):  yield x  else: # 精准模式下逐个词语输出  for xx in x:  yield xx |

### 分词模块功能设计

分词模块框架如下：



分词模块，包括加载词库模块、粗切分词语模块、处理未登录词、词性标注等四个模块。各个模块功能设计如下：

1. 加载词库模块

功能描述：该模块主要用于初始化分词库。分词库不会立即初始化，我们采用延时加载的机制，并生成缓存。在生成缓存之后，如果有存在新词库，也对它进行初始化，合并到一起。

模块设计过程：通过前缀词典初始化词库，优化内存使用率。通过Python的marshal模块生成缓存文件和加载缓存文件。

加载分词库源代码如下：

|  |
| --- |
| def initialize(self, dictionary=None):  """  判断有无已经缓存的前缀词典cache\_file文件，  若有相应的cache文件则直接使用 marshal.load  方法加载前缀词典，若无则通过gen\_pfdict对指  定的词库dict.txt进行计算生成前缀词典  """  if dictionary:  abs\_path = \_get\_abs\_path(dictionary)  if self.dictionary == abs\_path and self.initialized:  return  else:  self.dictionary = abs\_path  self.initialized = False  else:  abs\_path = self.dictionary  with self.lock:  try:  with DICT\_WRITING[abs\_path]:  pass  except KeyError:  pass  if self.initialized:  return  default\_logger.debug("Building prefix dict from %s ..." % (abs\_path or 'the default dictionary'))  t1 = time.time()  if self.cache\_file:  cache\_file = self.cache\_file  # default dictionary  elif abs\_path == DEFAULT\_DICT:  self.cache = "zaber\_nlp.cache"  cache\_file = self.cache  # custom dictionary  else:  cache\_file = "zaber\_nlp.u%s.cache" % md5(  abs\_path.encode('utf-8', 'replace')).hexdigest()  cache\_file = os.path.join(  self.tmp\_dir or tempfile.gettempdir(), cache\_file)  # prevent absolute path in self.cache\_file  tmpdir = os.path.dirname(cache\_file)  load\_from\_cache\_fail = True  if os.path.isfile(cache\_file) and (abs\_path == DEFAULT\_DICT or  os.path.getmtime(cache\_file) > os.path.getmtime(abs\_path)):  default\_logger.debug(  "Loading model from cache %s" % cache\_file)  try:  with open(cache\_file, 'rb') as cf:  self.FREQ, self.total = marshal.load(cf)  load\_from\_cache\_fail = False  except Exception:  load\_from\_cache\_fail = True  if load\_from\_cache\_fail:  wlock = DICT\_WRITING.get(abs\_path, threading.RLock())  DICT\_WRITING[abs\_path] = wlock  with wlock:  self.FREQ, self.total = self.gen\_pfdict(self.get\_dict\_file())  default\_logger.debug(  "Dumping model to file cache %s" % cache\_file)  try:  # prevent moving across different filesystems  fd, fpath = tempfile.mkstemp(dir=tmpdir)  with os.fdopen(fd, 'wb') as temp\_cache\_file:  marshal.dump(  (self.FREQ, self.total), temp\_cache\_file)  \_replace\_file(fpath, cache\_file)  except Exception:  default\_logger.exception("Dump cache file failed.")  try:  del DICT\_WRITING[abs\_path]  except KeyError:  pass  self.initialized = True  default\_logger.debug(  "Loading model cost %.3f seconds." % (time.time() - t1))  default\_logger.debug("Prefix dict has been built successfully.") |

加载新词库源代码如下：

|  |
| --- |
| def load\_userdict(self, f):  """  加载新词词库，提高分词准确率  """  self.check\_initialized()  if isinstance(f, string\_types):  f\_name = f  f = open(f, 'rb')  else:  f\_name = resolve\_filename(f)  for lineno, ln in enumerate(f, 1):  line = ln.strip()  if not isinstance(line, text\_type):  try:  line = line.decode('utf-8').lstrip('\ufeff')  except UnicodeDecodeError:  raise ValueError('dictionary file %s must be utf-8' % f\_name)  if not line:  continue  # match won't be None because there's at least one character  word, freq, tag = re\_userdict.match(line).groups()  if freq is not None:  freq = freq.strip()  if tag is not None:  tag = tag.strip()  self.add\_word(word, freq, tag)  def add\_word(self, word, freq=None, tag=None):  self.check\_initialized()  word = str\_decode(word)  freq = int(freq) if freq is not None else self.suggest\_freq(word, False)  self.FREQ[word] = freq  self.total += freq  if tag:  self.user\_word\_tag\_tab[word] = tag  for ch in xrange(len(word)):  wfrag = word[:ch + 1]  if wfrag not in self.FREQ:  self.FREQ[wfrag] = 0 |

1. 粗切分词语

功能描述：该模块主要实现生成句子中汉字所有成词可能所构成的有向无环图，并找出最优切分路径。

模块设计过程：通过Python的字典表示DAG，在加载词库的时候已经将词语和词语的所有前缀加入了词典。利用典型的DP算法对句子生成的有向无环图进行最大概率路径的计算。DAG的每个节点都是带权重的，对于在词典里面的词语，其权重为其词频，即FREQ[word]。我们要求得，使得最大。

DAG源代码如下：

|  |
| --- |
| def get\_DAG(self, sentence):  # 获得待切分句子的DAG  self.check\_initialized()  DAG = {}  N = len(sentence)  for k in xrange(N):  tmplist = []  i = k  frag = sentence[k]  while i < N and frag in self.FREQ:  if self.FREQ[frag]:  tmplist.append(i)  i += 1  frag = sentence[k:i + 1]  if not tmplist:  tmplist.append(k)  DAG[k] = tmplist  return DAG |

最优切分组合求解源代码如下：

|  |
| --- |
| def calc(self, sentence, DAG, route):  N = len(sentence)  route[N] = (0, 0)  # 对概率值取对数，让概率相乘变成对数相加,防止相乘造成下溢)  logtotal = log(self.total)  # 从后往前遍历句子 反向计算最大概率  for idx in xrange(N - 1, -1, -1):  route[idx] = max((log(self.FREQ.get(sentence[idx:x + 1]) or 1) -  logtotal + route[x + 1][0], x) for x in DAG[idx]) |

1. 识别未登录词

功能描述：该模块主要对于没有在语料库出现的单个词语（即未登陆词）通过概率计算形成词语组合。

模块设计过程：通过HMM模型发现未登录词，通过用Viterbi算法来得到这个最佳的隐藏状态序列。利用预存好的汉字的状态分布概率、状态转换概率和状态发射概率计算出一串汉字最有可能的BMES组合形式，从而进行分词。

HMM模型的源代码如下：

|  |
| --- |
| def \_\_cut(sentence):  global emit\_P  prob, pos\_list = viterbi(sentence, 'BMES', start\_P, trans\_P, emit\_P)  begin, nexti = 0, 0  # print pos\_list, sentence  for i, char in enumerate(sentence):  pos = pos\_list[i]  if pos == 'B':  begin = i  elif pos == 'E':  yield sentence[begin:i + 1]  nexti = i + 1  elif pos == 'S':  yield char  nexti = i + 1  if nexti < len(sentence):  yield sentence[nexti:] |

Viterbi算法的源代码如下：

|  |
| --- |
| def viterbi(obs, states, start\_p, trans\_p, emit\_p):  V = [{}] # tabular  mem\_path = [{}]  all\_states = trans\_p.keys()  for y in states.get(obs[0], all\_states): # init  V[0][y] = start\_p[y] + emit\_p[y].get(obs[0], MIN\_FLOAT)  mem\_path[0][y] = ''  for t in xrange(1, len(obs)):  V.append({})  mem\_path.append({})  #prev\_states = get\_top\_states(V[t-1])  prev\_states = [  x for x in mem\_path[t - 1].keys() if len(trans\_p[x]) > 0]  prev\_states\_expect\_next = set(  (y for x in prev\_states for y in trans\_p[x].keys()))  obs\_states = set(  states.get(obs[t], all\_states)) & prev\_states\_expect\_next  if not obs\_states:  obs\_states = prev\_states\_expect\_next if prev\_states\_expect\_next else all\_states  for y in obs\_states:  prob, state = max((V[t - 1][y0] + trans\_p[y0].get(y, MIN\_INF) +  emit\_p[y].get(obs[t], MIN\_FLOAT), y0) for y0 in prev\_states)  V[t][y] = prob  mem\_path[t][y] = state  last = [(V[-1][y], y) for y in mem\_path[-1].keys()]  # if len(last)==0:  # print obs  prob, state = max(last)  route = [None] \* len(obs)  i = len(obs) - 1  while i >= 0:  route[i] = state  state = mem\_path[i][state]  i -= 1  return (prob, route) |

1. 词性标注

功能描述：该模块主要实现对分词结果进行词性标注。

模块设计过程：通过Python的re模块正则表达式匹配是否为数字英文，并标注对应词性。若为汉字词语，则通过词性词典标注词性。

词性标注源代码如下：

|  |
| --- |
| def \_\_cut\_internal(self, sentence):  self.makesure\_userdict\_loaded()  sentence = str\_decode(sentence)  blocks = re\_han\_internal.split(sentence)  cut\_blk = self.\_\_cut\_DAG  for blk in blocks:  if re\_han\_internal.match(blk):  for word in cut\_blk(blk):  yield word  else:  tmp = re\_skip\_internal.split(blk)  for x in tmp:  if re\_skip\_internal.match(x):  yield pair(x, 'x')  else:  for xx in x:  if re\_num.match(xx):  yield pair(xx, 'm')  elif re\_eng.match(x):  yield pair(xx, 'eng')  else:  yield pair(xx, 'x') |

## 模型整体流程设计

模型整体流程如下：



分词模型采用HMM模型和Viterbi算法，前缀词典生成词库，DAG构建词语所有可能组合，DP算法求解最优切分路径。

# 热词提取模型设计

## 模型业务目标

基于python对资讯文章提取热门词汇，根据来源于saas的数据，进行热词提取，词性剔除和除去停用词。

## 模型逻辑设计

### 文本预处理逻辑

#### 文本分词

由于词是最小的能够独立运用的语言单位，而很多孤立语和黏着语（如汉语）的文本不像西方的屈折语的文本，词与词之间没有任何空格之类的自然分界符，因此，文本分词就成了计算机处理孤立语和黏着语文本时面临的首要问题。

考虑到文本分词存在切分歧义消除和未登录词识别两个主要问题。所以我们将文本分词分为两个阶段：首先对文本进行粗切分，然后再进行歧义消除和未登录识别。针对这两个阶段，我们将采用基于统计信息的N-最短路径粗分模型和基于字的生成式和区分式模型相结合的HMM模型。[详见分词算法设计](#_分词算法设计)

#### 词性标注

词性是词汇基本的语法属性，通常也称为词类。词性标注就是在给定句子中判定每个词的语法范畴，确定其词性并加以标注的过程。词性标注是自然语言处理中一项非常重要的基础性工作。我们将采用基于HMM的词性标注方法。[详见词性标注算法设计](#_词性标注算法设计)

#### 文本清洗

对于一个文本，将每一个句子作为一个窗口进行分词可得到该句子的各个特征项。由于这些特征项的维数非常高，存在许多对热词提取无用的特征，因此一方面利用停用词库去掉这些无用的词，并进行词性过滤。另外一方面，根据用户需求，除去噪声词。

### 热词提取算法设计

TextRank可以表示为一个有向有权图，由点集合 V和边集合 E 组成，E 是V ×V的子集。任两点之间边的权重为 ，对于一个给定的点，为指向该点的点集合，为点  指向的点集合。点  的得分定义WS如下：

其中，d为阻尼系数，取值范围为0到1，代表从图中某一特定点指向其他任意点的概率，一般取值为0.85。使用TextRank算法计算图中各点的得分时，需要给图中的点指定任意的初值，并递归计算直到收敛，即图中任意一点的误差率小于给定的极限值时就可以达到收敛，一般该极限值取 0.0001。

根据计算结果对特征词进行排序，并取前若干个特征词就可以得到文档对应的关键词列表。可根据文本的长度调整相应的特征词数量阈值。

## 模型功能模块设计

### 文本预处理模块功能设计

文本预处理框架如下：



文本预处理模块，包括文本分词模块、词性标注模块和文本清洗模块等三个模块。各个模块功能设计如下：

1. 文本分词、词性标注

功能描述：该模块主要用于将输入的文本或文本集的内容分割成句子，并对句子进行分词和词性标注。

模块设计过程：通过文本分词模型实现。

具体源代码见[分词模块功能设计](#_分词模块功能设计)

1. 文本清洗

功能描述：该模块主要用于清洗无用的词语。减少无关的特征，并保留候选的关键词。

模块设计过程：通过集合的关系运算（交集运算）实现噪声词除去和词性剔除。

文本清洗源代码如下：

|  |
| --- |
| def pairfilter(self, wp):  return (wp.flag in self.pos\_filt and len(wp.word.strip()) >= 2  and wp.word.lower() not in self.stop\_words) |

### 热词提取模块功能设计

热词提取模块框架如下：



热词提取模块，包括构建无权词语网络图模块、节点权重迭代计算模块等两个个模块。各个模块功能设计如下：

1. 构建无权词语网路图模块

功能描述：该模块借鉴Google 公司 PageRank算法的思路，将句子间的相似关系看成是一种推荐或投票关系，由此构建无权词语网络图。

模块设计过程：无向词语网络图，其中为边上的权重集合。为节点与间边的权重值，可通过距离相似度计算函数计算所得。

无权网络图源代码如下：

|  |
| --- |
| g = UndirectWeightedGraph()  cm = defaultdict(int)  words = tuple(self.tokenizer.cut(sentence))  for i, wp in enumerate(words):  if self.pairfilter(wp):  for j in xrange(i + 1, i + self.span):  if j >= len(words):  break  if not self.pairfilter(words[j]):  continue  if allowPOS and withFlag:  cm[(wp, words[j])] += 1  else:  cm[(wp.word, words[j].word)] += 1  for terms, w in cm.items():  g.addEdge(terms[0], terms[1], w) |

1. 节点权重迭代计算模块

功能描述：该模块主要用于根据公式，迭代传播权重计算各词语节点的得分，计算出每个词语节点的重要性。

模块设计过程：由图可得到节点间的一个的相似度矩阵，根据给定的G及相应的相似度矩阵，可迭代计算各个节点的权重，具体的权重计算如下：

迭代计算代码如下：

|  |
| --- |
| def rank(self):  ws = defaultdict(float)  outSum = defaultdict(float)  wsdef = 1.0 / (len(self.graph) or 1.0)  for n, out in self.graph.items():  ws[n] = wsdef  outSum[n] = sum((e[2] for e in out), 0.0)  # this line for build stable iteration  sorted\_keys = sorted(self.graph.keys())  for x in xrange(10): # 10 iters  for n in sorted\_keys:  s = 0  for e in self.graph[n]:  s += e[2] / outSum[e[1]] \* ws[e[1]]  ws[n] = (1 - self.d) + self.d \* s  (min\_rank, max\_rank) = (sys.float\_info[0], sys.float\_info[3])  for w in itervalues(ws):  if w < min\_rank:  min\_rank = w  if w > max\_rank:  max\_rank = w  for n, w in ws.items():  # to unify the weights, don't \*100.  ws[n] = (w - min\_rank / 10.0) / (max\_rank - min\_rank / 10.0)  return ws |

## 模型整体流程设计

模型整体流程如下：



利用文本分词模型进行文本分词和词性标注。利用Text Rank计算词语权重。

# 词云展示接口设计

## 接口业务目标

通过热词和热词权重生成词云的接口，可以灵活设置词云中词语的数目，如通过设置数目个数、权重阈值、噪声词过滤等。

## 接口逻辑设计

### 设计原则

RESTFUL设计原则，它提供了一种约束：需要用一个统一的接口来完成子系统之间以及服务与用户之间的交互。这使得各个子系统可以独自完成演化。对于这个约束，它的规范如下：

1. 唯一的资源标识
2. 简单的方法（抽象方法）
3. 一定的表达方式

REST 是以资源为中心，名词即资源的地址，动词即施加于名词上的一些有限操作，表达是对各种资源形态的抽象。

### 认证设计

Django Ajax CSRF 认证，CSRF 中间件和模板标签提供对跨站请求伪造简单易用的防护。防护CSRF 攻击的第一道防线是保证GET 请求（以及在9.1.1 Safe Methods，HTTP 1.1， [RFC 2616](http://tools.ietf.org/html/rfc2616.html#section-9.1.1) 中定义的其它安全的方法）不会产生副作用。对不安全的请求方法例如POST、PUT和DELETE进行防护。

## 接口功能规范设计

### 接口规范

接口设计规范如表2所示：

**表2 接口设计规范**

|  |  |
| --- | --- |
| **协议** | http(s) |
| **方法** | POST |
| **返回格式** | JSON |

### 协议

API与用户的通信协议，推荐使用HTTPs协议。

### 域名

应该尽量将API部署在专用域名之下。

比如：https://bdms.baifendian.com

### 版本

应该将API的版本号放入URL。

比如：https://bdms.baifendian.com/v1/

### 路径

路径又称"终点"（endpoint），表示API的具体网址。

在RESTful架构中，每个网址代表一种资源（resource），所以网址中不能有动词，只能有名词，而且所用的名词往往与数据库的表格名对应。一般来说，数据库中的表都是同种记录的"集合"（collection），所以API中的名词也应该使用复数。

举例来说，有一个API提供动物园（zoo）的信息，还包括各种动物和雇员的信息，则它的路径应该设计成下面这样。

https://api.example.com/v1/zoos

https://api.example.com/v1/animals

https://api.example.com/v1/employees

### HTTP动词

对于资源的具体操作类型，由HTTP动词表示。

常用的HTTP动词有下面五个：

GET：从服务器取出资源（一项或多项）。

POST：在服务器新建一个资源。

PUT：在服务器更新资源（客户端提供改变后的完整资源）。

PATCH：在服务器更新资源（客户端提供改变的属性）。

DELETE：从服务器删除资源。

还有两个不常用的HTTP动词。

HEAD：获取资源的元数据。

OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的。

### 返回结果

针对不同操作，服务器向用户返回的结果应该符合以下规范：

GET /collection：返回资源对象的列表（数组）

GET /collection/resource：返回单个资源对象

POST /collection：返回新生成的资源对象

PUT /collection/resource：返回完整的资源对象

PATCH /collection/resource：返回完整的资源对象

DELETE /collection/resource：返回一个空文档

### 其他

1. API的token认证使用Django Ajax CSRF 认证。
2. 服务器返回的数据格式使用JSON。

## 接口整体流程设计



# 关联分析模型设计

## 模型业务目标

通过资讯全文内容与实体名称库的关联，找到关联程度最大的具体的股票、公私募公司等。结合盯盘模型进行盘中异动股票的关联资讯推送。

## 模型逻辑设计

AC自动机是一类在字典树上进行kmp的多模式匹配算法。它可以被用来寻找在文本中找出出现的关键词，并且在效率上比其他常见的算法高。

建立AC自动机可以分为三步：

### 模型初始化

#### 构造前缀树

在实现前缀树的时候，会在节点结构中设置一个标志，用来标记该节点处是否构成一个单词（关键字）。可以看出，前缀树的关键字一般都是字符串，而且前缀树把每个关键字保存在一条路径上，而不是一个节点中。另外，两个有公共前缀的关键字，在前缀树中前缀部分的路径相同。

#### 构造失败指针

假设某个节点上的字母为C，沿着父节点的失败指针走，直到走到一个节点，它的子节点中也有字母为C的节点。然后把当前节点的失败指针指向那个字母也为C的节点。如果一直走到了root都没找到，那就把失败指针指向root。

使用广度优先搜索BFS，层次遍历节点来处理，每一个节点的失败路径。

特殊处理：第二层要特殊处理，将这层中的节点的失败路径直接指向父节点(也就是根节点)。

### 模式匹配过程

从root节点开始，每次根据读入的字符沿着自动机向下移动。

当读入的字符，在分支中不存在时，递归走失败路径。如果走失败路径走到了root节点，则跳过该字符，处理下一个字符。

因为AC自动机是沿着输入文本的最长后缀移动的，所以在读取完所有输入文本后，最后递归走失败路径，直到到达根节点，这样可以检测出所有的模式。

## 模型功能模块设计

关联分析模块框架如下：



## 模型整体流程设计



# 情感分析模型设计

## 模型业务目标

## 模型逻辑设计

## 模型功能模块设计

## 模型整体流程设计

# 附录

**附录1**

**词性标注表**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **一类** | **二类** | **符号** |
| 1 | 名词 | 名词 | n |
| 2 | 人名 | nr |
| 3 | 汉语姓氏 | nr1 |
| 4 | 汉语名字 | nr2 |
| 5 | 日语人名 | nrj |
| 6 | 音译人名 | nrf |
| 7 | 地名 | ns |
| 8 | 音译地名 | nsf |
| 9 | 机构团体名 | nt |
| 10 | 其它专名 | nz |
| 11 | 名词性惯用语 | nl |
| 12 | 名词性语素 | ng |
| 13 | 处所词 | 处所词 | s |
| 14 | 方位词 | 方位词 | f |
| 15 | 时间词 | 时间词 | t |
| 16 | 时间词性语素 | tg |
| 17 | 动词 | 动词 | v |
| 18 | 副动词 | vd |
| 19 | 名动词 | vn |
| 20 | 动词“是” | vshi |
| 21 | 动词“有” | vyou |
| 22 | 趋向动词 | vf |
| 23 | 形式动词 | vx |
| 24 | 不及物动词（内动词） | vi |
| 25 | 动词性惯用语 | vl |
| 26 | 动词性语素 | vg |
| 27 | 形容词 | 形容词 | a |
| 28 | 副形词 | ad |
| 29 | 名形词 | an |
| 30 | 形容词性语素 | ag |
| 31 | 形容词性惯用语 | al |
| 32 | 区别词 | 区别词 | b |
| 33 | 区别词性惯用语 | bl |
| 34 | 代词 | 代词 | r |
| 35 | 人称代词 | rr |
| 36 | 指示代词 | rz |
| 37 | 时间指示代词 | rzt |
| 38 | 处所指示代词 | rzs |
| 39 | 谓词性指示代词 | rzv |
| 40 | 疑问代词 | ry |
| 41 | 时间疑问代词 | ryt |
| 42 | 处所疑问代词 | rys |
| 43 | 谓词性疑问代词 | ryv |
| 44 | 代词性语素 | rg |
| 45 | 数词 | 数词 | m |
| 46 | 数量词 | mq |
| 47 | 量词 | 量词 | q |
| 48 | 时量词 | qt |
| 49 | 动量词 | qv |
| 50 | 副词 | 副词 | d |
| 51 | 介词 | 介词 | p |
| 52 | 介词“把” | pba |
| 53 | 介词“被” | pbei |
| 54 | 连词 | 连词 | c |
| 55 | 并列连词 | cc |
| 56 | 助词 | 助词 | u |
| 57 | 着 | uzhe |
| 58 | 了 喽 | ule |
| 59 | 过 | uguo |
| 60 | 的 底 | ude1 |
| 61 | 地 | ude2 |
| 62 | 得 | ude3 |
| 63 | 所 | usuo |
| 64 | 等 等等 云云 | udeng |
| 65 | 一样 一般 似的 般 | uyy |
| 66 | 的话 | udh |
| 67 | 来讲 来说 而言 说来 | uls |
| 68 | 之 | uzhi |
| 69 | 连 （“连小学生都会”） | ulian |
| 70 | 叹词 | 叹词 | e |
| 71 | 语气词 | 语气词(delete yg) | y |
| 72 | 拟声词 | 拟声词 | o |
| 73 | 前缀 | 前缀 | h |
| 74 | 后缀 | 后缀 | k |
| 75 | 字符串 | 字符串 | x |
| 76 | 非语素字 | xx |
| 77 | 网址URL | xu |
| 78 | 标点符号 | 标点符号 | w |
| 79 | 左括号，全角：（ 〔 ［ ｛ 《 【 〖 〈 半角：( [ { < | wkz |
| 80 | 右括号，全角：） 〕 ］ ｝ 》 】 〗 〉 半角： ) ] { > | wky |
| 81 | 左引号，全角：“ ‘ 『 | wyz |
| 82 | 右引号，全角：” ’ 』 | wyy |
| 83 | 句号，全角：。 | wj |
| 84 | 问号，全角：？ 半角：? | ww |
| 85 | 叹号，全角：！ 半角：! | wt |
| 86 | 逗号，全角：， 半角：, | wd |
| 87 | 分号，全角：； 半角： ; | wf |
| 88 | 顿号，全角：、 | wn |
| 89 | 冒号，全角：： 半角： : | wm |
| 90 | 省略号，全角：…… … | ws |
| 91 | 破折号，全角：—— －－ ——－ 半角：--- ---- | wp |
| 92 | 百分号千分号，全角：％ ‰ 半角：% | wb |
| 93 | 单位符号，全角：￥ ＄ ￡ ° ℃ 半角：$ | wh |
|  |  |  |  |