【NLP】【三】jieba源码分析之关键字提取（TF-IDF/TextRank）

nlp 机器AI学习 数据AI挖掘 5月16日

想了解更多好玩的人工智能应用，请关注公众号“机器AI学习 数据AI挖掘”，”智能应用"菜单中包括：颜值检测、植物花卉识别、文字识别、人脸美妆等有趣的智能应用。。

一】综述

利用jieba进行关键字提取时，有两种接口。一个基于TF-IDF算法，一个基于TextRank算法。TF-IDF算法，完全基于词频统计来计算词的权重，然后排序，在返回TopK个词作为关键字。TextRank相对于TF-IDF，基本思路一致，也是基于统计的思想，只不过其计算词的权重时，还考虑了词的上下文（通过窗口滑动来实现），而且计算词的权重时，也考虑了相关联系词的影响。可以说，TextRank实际上是依据位置与词频来计算词的权重的。下面，结合基于jieba源码，来分别解释两种算法的实现。

【二】TF-IDF

1. 原理解析

假设，共有N篇文档，分别用 d1,d2,d3,,,,,,dn来表示。

TF = 某个词在di篇文章中出现的次数/di篇文章的总词数 = count（W in di）/ count(di）。因此，TF计算的是单个词在单个文档中出现的词频。

IDF = 总的文档数 / 出现词W的文档数 。IDF其实反映了词W在文档之间的区别度。如果W在仅在一篇文档中出现，则说明可以使用W将该文档与其他文档区别开来。即IDF可以反映W的独特性 。

TF*IDF，可以得到词的重要性。比如：北京和西安在同一篇文档中的词频均为20%，那如何估计北京是该文的关键字，还是西安呢？如果同时有10篇文章均提到了北京，恰好只有这篇文章提到了西安，则西安作为这篇文章的关键字更为合理。

2. idf.txt

jieba有统计好的idf值，在 jieba/analyse/idf.txt中。

```
劳动防护 13.900677652
生化学 13.900677652
奥萨贝尔 13.900677652
考察队员 13.900677652
岗上 11.5027823792
倒车档 12.2912397395
```

3. idf.txt 加载

代码在 jieba/analyse/tfidf.py

```
class IDFLoader(object):

    def __init__(self, idf_path=None):
        self.path = ""
        self.idf_freq = {}
        # 初始化idf的中位数值
        self.median_idf = 0.0
        if idf_path:
            # 解析idf.txt
            self.set_new_path(idf_path)

    def set_new_path(self, new_idf_path):
        if self.path != new_idf_path:
            self.path = new_idf_path
            content = open(new_idf_path, 'rb').read().decode('utf-8')
            self.idf_freq = {}
            # 解析 idf.txt，拿到词与idf的对应值，key = word，value = idf
            for line in content.splitlines():
                word, freq = line.strip().split(' ')
                self.idf_freq[word] = float(freq)
            # 取idf的中位数
            self.median_idf = sorted(
                self.idf_freq.values())[len(self.idf_freq) // 2]
```

4. 利用tfidf算法提取关键字的接口：extract_tags

```
    def extract_tags(self, sentence, topK=20, withWeight=False, allowPOS=(), withFlag=F
        """
        Extract keywords from sentence using TF-IDF algorithm.
        Parameter:
            - topK: return how many top keywords. `None` for all possible words.
            - withWeight: if True, return a list of (word, weight);
                          if False, return a list of words.
            - allowPOS: the allowed POS list eg. ['ns', 'n', 'vn', 'v','nr'].
                        if the POS of w is not in this list,it will be filtered.
            - withFlag: only work with allowPOS is not empty.
                        if True, return a list of pair(word, weight) like posseg.cut
                        if False, return a list of words
        """
        # 判断提取出哪些词性的关键字
        if allowPOS:
            allowPOS = frozenset(allowPOS)
            # 如果需要提取指定词性的关键字，则先进行词性分割
            words = self.postokenizer.cut(sentence)
        else:
            # 如果提取所有词性的关键字，则使用精确分词
            words = self.tokenizer.cut(sentence)
        freq = {}
        # 按照分词结果，统计词频
        for w in words:
            if allowPOS:
                if w.flag not in allowPOS:
                    continue
                elif not withFlag:
                    w = w.word
            wc = w.word if allowPOS and withFlag else w
            # 该词不能是停用词
            if len(wc.strip()) < 2 or wc.lower() in self.stop_words:
                continue
            #统计该词出现的次数
            freq[w] = freq.get(w, 0.0) + 1.0
        # 计算总的词数目
        total = sum(freq.values())
        for k in freq:
            kw = k.word if allowPOS and withFlag else k
            # 依据tf-idf公式进行tf-idf值，作为词的权重。其中，idf是jieba通过语料库统计得到
            freq[k] *= self.idf_freq.get(kw, self.median_idf) / total

        # 对词频做个排序，获取TopK的词
        if withWeight:
            tags = sorted(freq.items(), key=itemgetter(1), reverse=True)
        else:
            tags = sorted(freq, key=freq.__getitem__, reverse=True)
        if topK:
            return tags[:topK]
        else:
            return tags
```

5. jieba实现tf-idf总结

1）：idf的值时通过语料库统计得到的，所以，实际使用时，可能需要依据使用环境，替换为使用对应的语料库统计得到的idf值。

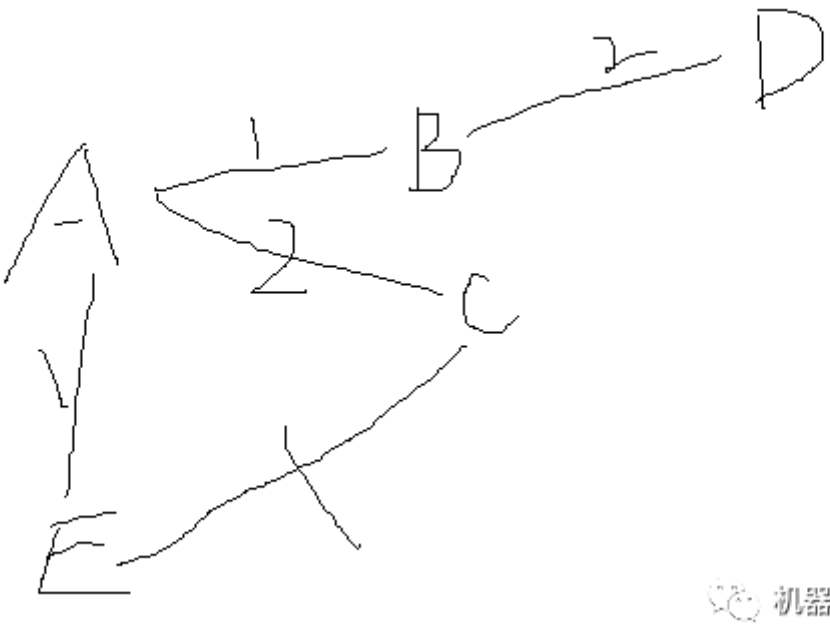
2）：需要从分词结果中去除停用词。

3）：如果指定了仅提取指定词性的关键词，则词性分割非常重要，词性分割中准确程度，影响关键字的提取。

【三】TextRank

1. 算法原理介绍

TextRank采用图的思想，将文档中的词表示成一张无向有权图，词为图的节点，词之间的联系紧密程度体现为图的边的权值。计算词的权重等价于计算图中节点的权重。提取关键字，等价于找出图中权重排名TopK的节点。



如上图所示：有A B C D E五个词，词之间的关系使用边连接起来，词之间连接的次数作为边的权值。比如：A和C一起出现了2次，则其边的权重为2,A与B/C/E都有联系，而D仅与B有联系。

所以说，TextRank背后体现的思想为：与其他词关联性强的词，越重要。通俗一点就是：围着谁转，谁就重要。就像大家基本都会围着领导转一样。

2. 图的构建

图的构建分为两部分：

1）：确认图的节点之间的联系

2）：确认边的权值

jieba是如何做的呢?

```
def textrank(self, sentence, topK=20, withWeight=False, allowPOS=('ns', 'n', 'vn',
    """
    Extract keywords from sentence using TextRank algorithm.
    Parameter:
        - topK: return how many top keywords. `None` for all possible words.
        - withWeight: if True, return a list of (word, weight);
                      if False, return a list of words.
        - allowPOS: the allowed POS list eg. ['ns', 'n', 'vn', 'v'].
                    if the POS of w is not in this list, it will be filtered.
        - withFlag: if True, return a list of pair(word, weight) like posseg.cut
                    if False, return a list of words
    """
    # 初始化关键字词性过滤条件
    self.pos_filt = frozenset(allowPOS)
    # 初始化一个无向权值图
    g = UndirectWeightedGraph()
    cm = defaultdict(int)
    # 使用精确模式进行分词
    words = tuple(self.tokenizer.cut(sentence))
    # 遍历分词结果
    for i, wp in enumerate(words):
        # 词wp如果满足关键词备选条件，则加入图中
        if self.pairfilter(wp):
            # span为滑动窗口，即词的上下文，借此来实现此的共现，完成词之间的连接。
            for j in xrange(i + 1, i + self.span):
                if j >= len(words):
                    break
                # 后向词也要满足备选词条件
                if not self.pairfilter(words[j]):
                    continue
                if allowPOS and withFlag:
                    # 共现词作为图一条边的两个节点，共现词出现的次数，作为边的权值
                    cm[(wp, words[j])] += 1
                else:
                    cm[(wp.word, words[j].word)] += 1
    # 将 备选词和与该词连接的词加入到graph中，即完成graph的构造
    for terms, w in cm.items():
        g.addEdge(terms[0], terms[1], w)
    # 调用graph的rank接口，完成TextRank算法的计算，即计算出各节点的权重
    nodes_rank = g.rank()
    if withWeight:
        # 对graph中的阶段的权重进行排序
        tags = sorted(nodes_rank.items(), key=itemgetter(1), reverse=True)
    else:
        tags = sorted(nodes_rank, key=nodes_rank.__getitem__, reverse=True)
```

```
        if topK:
            return tags[:topK]
        else:
            return tags
```

3. TextRank算法的实现

```
    def rank(self):
        ws = defaultdict(float)
        outSum = defaultdict(float)

        # 计算初始化节点的weight值
        wsdef = 1.0 / (len(self.graph) or 1.0)
        # 初始化各个节点的weight值，并计算各个节点的出度数目
        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())
        # 循环迭代10，迭代计算出各个节点的weight值
        for x in xrange(10):  # 10 iters
            for n in sorted_keys:
                s = 0
                # 依据TextRank公式计算weight
                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
```

【四】TF-IDF与TextRank算法的比较

1. 从算法原理上来看，基础都是词频统计，只是TD-IDF通过IDF来调整词频的权值，而TextRank通过上下文的连接数来调整词频的权值。TextRank通过滑动窗口的方式，来实现词的位置对词的权值的影响。

2. TD-IDF计算简单，运行性能更好。