关键词提取：TF-IDF和n-gram（附美女相亲帖）

原创 邓邓最棒 叫我NLPer 4月11日

一：今日相亲

搭档镇楼。

今天的头版给我漂亮的搭档，啥年芳二六、待字闺中之类的矫情话就不说了，希望看到文章的小伙子，如果对眼，请放下你手中的游戏，我可以牵线搭桥。

好好相爱，就是为民除害。

搭档是重庆妹纸，重庆妹纸长得是很水灵。

搭档给我的感觉是情商比较高，比较会捧哏，说话不会闷。

搭档身高160体重100，学历本科水瓶座，目前在重庆的银行工作。

以下为搭档的自我介绍：

性格慢热，绝不随便。
心里住着公主，也住着女王。
能吃苦下田种地，也能温书习字梳妆。
喜欢跳伞游泳潜水冲浪，就差一个男生一起去体验。
家里两房一车，父母事业单位就职，性格开明。

希望男生干净、上进，五官端正，性格开朗健谈，收入稳定能养家，家庭和谐，会做饭更佳。

二：内容预告

最近公司要做英语新闻推荐系统，需要对新闻内容做分析，主要包括网页去重、实体抽取和关键词提取。

提到关键词提取，大家很容易就能想到TF-IDF和TextRank。这两种方法可以分别通过调sklearn和jiabe的包来实现。

可如果要提取的是几个单词组成的短语呢？

这时候按标准的调包教程来做，是难以满足业务需要的。

所以本文探索，如何把TF-IDF和n-gram结合，用来提取短语。

本文关注以下三个问题：

- 关键词提取有哪些方法？
- 如何把TF-IDF和n-gram结合？
- 如何根据词性和停用词过滤？

三：关键词提取的方法

01 TF-IDF

TF-IDF在nlp领域无人不知无人不晓，思想简单却有效，荣获nlp界的诺贝尔奖：奥卡姆剃刀奖。

TF(term frequency)，即词频，用来衡量词在一篇文档中的重要性，词频越高，越重要。计算公式为：

$$TF = \frac{\text{某文档中某词出现的次数}}{\text{该文档的总词数}}$$

IDF((inverse document frequency)，叫做逆文档频率，衡量某个词在所有文档集合中的常见程度。

当包含某个词的文档的数量越多时，这个词也就烂大街了，重要性越低。计算公式为：

$$IDF = log\frac{\text{全部文档的数量}}{\text{包含某词的文档的数量} + 1}$$

于是TF-IDF = TF * IDF, 它表明字词的重要性与它在某篇文档中出现的次数成正比，与它在所有文档中出现的次数成反比。

使用TF-IDF的一个假设前提是：已经去掉了停用词。

TF-IDF的优点是计算速度快，结果稳健。

需要输入多篇文档，可以输出每篇文档的关键词。

02 TextRank

TextRank基于图计算来提取关键词，需要进行迭代，速度比TF-IDF慢，但不需要通过输入多篇文档来提取关键词。

TextRank是把一篇文档构建成无向图，图中的节点就是词语，图上的边就是共现词之间的连接。

共现词通过滑动窗口来确定，共现词之间用边相连，而边上的权重可以使用共现词的相似度。

jieba提供了用TextRank提取关键词的函数，但是边的权重是词共现的频率。

这样做其实比较粗糙，我们可以使用基于词向量计算的相似度得分作为权重，来进行迭代。

我这不知不觉又给自己安排了任务，文章名字都想好了：

《TextRank提取关键词：我也是改过jieba源码的人！》

TextRank也需要先去掉停用词。

03 文本聚类法

TF-IDF只从浅层的词频角度挖掘关键词，而通过Kmeans或Topic Model，可以从深层的隐含语义角度来提取关键词。

一种方法是通过Kmeans来提取，使用词向量作为特征。

比如对于一篇文档，我们要提取10个关键词。

那么可以把文档中的词聚成5类，然后取每个类中，与类中心最近的2个点，作为关键词。

也可以直接聚成10类，取和类中心最近的词。

另一种方法是通过Topic Model来提取，比如LSA和LDA，使用词频矩阵作为特征。

比如对于包含多篇文档的单领域语料，我们要挖掘关键词，整理词库。

那么可以用LDA进行聚类，得到每个主题的单词分布，再取出每个主题下排名靠前的topk个单词，或者权重高于某个阈值的单词，构成关键词库。

```
主题的单词分布为：
(0, '0.025*"基金" + 0.020*"分红" + 0.007*"中" + 0.006*"考试" + 0.006*"私募" + 0.005*"公司
(1, '0.007*"套装" + 0.007*"中" + 0.006*"设计" + 0.004*"元" + 0.004*"拍摄" + 0.003*"万" +
(2, '0.007*"英寸" + 0.005*"中" + 0.004*"中国" + 0.003*"拍摄" + 0.003*"比赛" + 0.002*"高清
(3, '0.082*"基金" + 0.015*"公司" + 0.014*"市场" + 0.014*"投资" + 0.009*"股票" + 0.007*"亿
...
```

04 有监督的关键词提取

有监督的方法需要有标注的数据，我没有尝试过。

看一些文章说可以转化为统计机器翻译（SMT）的问题，转化为序列标注（NER）的问题，或者转化为词语排序（LTR）的问题。

我只理解了转化为序列标注问题的做法，这个和用深度学习做文本摘要类似。

文档中的词语，如果为关键词，则标注为1，否则标注为0，也就是对一个词进行二分类。

据说效果比上述无监督的方法好。

四：TF-IDF+n-gram提取关键词

提取单词作为关键词，比较容易实现，如果要提取短语呢？

我尝试了开源工具RAKE，它是根据停用词来划分句子，再提取短语的。

使用之后，我发现RAKE存在两个问题：

一是提取的短语有些长达4-5个单词，这显然不合适；
二是没有根据词性进行过滤。

当然，我们可以用RAKE得到一个粗糙的结果，然后再做细致的处理，比如根据包含单词的数量、根据词性模板等进行过滤。

不过RAKE的代码写得实在太乱了，我没有耐心看下去，对其原理也不太了解，也就没加工再利用。

我给出的方案是TF-IDF结合n-gram来提取关键短语，并根据单词长度、停用词和词性进行过滤。

01 英文新闻测试语料

我去某网站下载了4篇英语新闻，对于其中的一篇，经过增加一段和两段、删除一段和两段的操作，得到4篇内容高度重合的新闻，最终得到8篇英语新闻。

```
├── english_new_1.txt
├── english_new_2.txt
├── english_new_3.txt
├── english_new_add_1.txt       # 增加一段
├── english_new_add_2.txt       # 增加两段
├── english_new_base.txt        # 原始新闻
├── english_new_remove_1.txt    # 删除一段
└── english_new_remove_2.txt    # 删除两段
```

首先用SimHash去重，保留4篇英语新闻（为了测试SimHash）。

02 需要的库

TF-IDF的计算，用sklearn的包。

由于是英文文本的处理，所以需要用NLTK做英文单词拆分、词性标注和词形还原。

```
#coding:utf-8
import os,re
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
import numpy as np
from itertools import chain

from nltk import pos_tag, word_tokenize,sent_tokenize
from nltk.stem import WordNetLemmatizer

""" 一：初始化词形还原的类 """
lemmatizer = WordNetLemmatizer()
```

为什么需要做词形还原呢?

词形还原是指把英文的单词，从复数形态、第三人称形态等复杂形态，转换为最基础的形态，比如 salaries 还原为salary，makes还原为make。

如果我们提取的关键词是单个词，那么在使用TF-IDF进行提取之前，先要对每个单词做词形还原，不然salaries和salary会被认为是不同的两个单词。

词形还原是基于词典的，准确率比较高，所以使用NLTK做词形还原时，需要下载数据WordNet。

在公司很容易出现网络不通的问题，反正我是下载不了。

```
nltk.download('wordnet')

[nltk_data] Error loading wordnet: <urlopen error [Errno 104]
[nltk_data]     Connection reset by peer>
False
```

其实不止词形还原需要下载数据，做词性标注和实体识别，都需要下载，特别麻烦。

于是我干脆把NLTK的所有数据文件都下载了，文件大小为1.08G，解压后放到相应的路径，就一劳永逸了。

下载链接:
https://pan.baidu.com/s/1Ms4tfGlF3lA6F0Mg5Ljd8Q

提取码:
07qb

下载好后解压，在ubuntu环境下，给文件赋予相应的可执行权限（Goup和Others也需要可执行权限），然后把数据文件复制到以下路径:

```
  Searched in:

    - '/opt/anaconda3/nltk_data'
```

```
    - '/opt/anaconda3/share/nltk_data'
    - '/opt/anaconda3/lib/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
```

03 对新闻做单词拆分

最好不要对整篇文档做 word_tokenize，而是先 sent_tokenize（划分句子），再 word_tokenize。

因为实体识别、词性标注和句法分析，最好以句子为单位来做。

```
""" 一：对文档进行分词/拆字 """
def tokenize_doc(docs):
    """
    :params: docs—多篇文档
    """
    docs_tokenized = []
    for doc in docs:
        doc = re.sub('[\n\r\t]+',' ',doc)

        """ 1：分句 """
        sents = sent_tokenize(doc)

        """ 2：单词拆分 """
        sents_tokenized = [word_tokenize(sent.strip()) for sent in sents]
        docs_tokenized.append(sents_tokenized)

    return docs_tokenized
```

04 n-gram的生成

sklearn的TfidfVectorizer的类里边，也提供了n-gram的功能，那为什么我还要自己生成呢？

原因是sklearn内置的功能里，生成n-gram后，就直接计算TF-IDF了，没根据停用词和词性，过滤n-gram。

而根据停用词和词性过滤n-gram，必须在计算TF-IDF之前。

那在tokenize的时候做过滤不就得了吗？

不行！

过滤n-gram，是指如果n-gram中，有一个单词是停用词，或者有一个单词的词性是过滤的词性，那么整个n-gram都需要去掉，而不能只去掉该单词。

在tokenize时过滤单词，那么生成的n-gram短语块可能是错误的。

```
""" 二：产生n-gram，用于提取短语块 """
def gene_ngram(sentence,n=3,m=2):
    """
    ----------
    sentence: 分词后的句子
    n: 取3，则为3-gram
    m: 取1，则保留1-gram
    ----------
    """
    if len(sentence) < n:
        n = len(sentence)

    ngrams = [sentence[i-k:i] for k in range(m, n+1) for i in range(k, len(sentence)+1)]
    return ngrams
```

05 n-gram的过滤

生成的trigram如下：

```
[['What', 'are', 'the'], ['are', 'the', 'legal'], ['the', 'legal', 'implications'], ['l
```

我们要对n-gram进行过滤，根据停用词、词性、单词的长度以及是否包含数字，来过滤。

词性可以去网上找英文词性对照表。

如果n-gram中，包含长度为1的单词，那么过滤掉。

```
""" n-gram中是否有单词长度为1 """
def clean_by_len(gram):

    for word in gram:
        if len(word) < 2:
            return False

    return True


""" 三：按停用词表和词性，过滤单词 """
def clean_ngrams(ngrams):
    """
```

```
    :params: ngrams
    """
    stopwords = open("./百度英文停用词表.txt",encoding='utf-8').readlines()
    stopwords = [word.strip() for word in stopwords]
    pat = re.compile("[0-9]+")

    """ 如果n-gram中有停用词，则去掉 """
    ngrams = [gram for gram in ngrams if len(set(stopwords).intersection(set(gram)))==0

    """ 如果n-gram中有数字，则去掉 """
    ngrams = [gram for gram in ngrams if len(pat.findall(''.join(gram).strip()))==0]

    """ n-gram中有单词长度为1，则去掉 """
    ngrams = [gram for gram in ngrams if clean_by_len(gram)]

    """ 只保留名词、动词和形容词 """
    allow_pos_one = ["NN","NNS","NNP","NNPS"]
    allow_pos_two = ["NN","NNS","NNP","NNPS","JJ","JJR","JJS"]
    allow_pos_three = ["NN","NNS","NNP","NNPS","VB","VBD","VBG","VBN","VBP","VBZ","JJ",

    ngrams_filter = []
    for gram in ngrams:
        words,pos = zip(*pos_tag(gram))

        """ 如果提取单词作为关键词，则必须为名词 """
        if len(words) == 1:
            if not pos[0] in allow_pos_one:
                continue
            ngrams_filter.append(gram)

        else:
            """ 如果提取短语，那么开头必须为名词、动词、形容词，结尾为名词 """
            if not (pos[0] in allow_pos_three and pos[-1] in allow_pos_one):
                continue
            ngrams_filter.append(gram)

    return ngrams_filter
```

06 计算TF-IDF

用上面的函数，完成生成n-gram、过滤n-gram的步骤，然后把n-gram之间的单词用下划线连接："_"，n-gram之间用空格连接。

```
    """ 1：处理为n-gram，n_=2或3 """
    docs_ngrams = [gene_ngram(doc,n=n_,m=n_) for doc in docs_tokenized]

    """ 2：按停用词表和词性，过滤 """
    docs_ngrams = [clean_ngrams(doc) for doc in docs_ngrams]

    docs_ = []
    for doc in docs_ngrams:
        docs_.append(' '.join(['_'.join(ngram) for ngram in doc]))
```

得到的n-gram如下：

```
['face_tough_times Hiring_activity_declines reveals_Naukri_JobSpeak health_system_prepa
```

为什么n-gram之间的单词用下划线来连接呢？

我试了其他的符号：#、=，但是发现送入sklearn的包里计算时，n-gram会被重新拆分为单词，vocab不是n-gram短语，而是单词。

我估计是因为python中，下划线比较特殊，如正则表达式 \w，表示字母、数字和下划线，其他的符号则容易被视为文本噪音而去掉。

接着计算n-gram的TF-IDF，这里又有一个坑：如果不自己指定n-gram字典，那么sklearn自己构建的字典中，可能会有单词或单词加下划线这种奇怪的东西。

所以需要自己传入vocab。

```
""" 四：获取 tf-idf 特征 """
def calcu_tf_idf(documents):
    """
    :param: data为列表格式的文档集合，计算 tf_idf 特征
    """

    """ 指定vocab，否则n-gram的计算会出错 """
    vocab = set(chain.from_iterable([doc.split() for doc in documents]))

    vec = TfidfVectorizer(vocabulary=vocab)
    D = vec.fit_transform(documents)
    voc = dict((i, w) for w, i in vec.vocabulary_.items())

    features = {}
    for i in range(D.shape[0]):
        Di = D.getrow(i)
        features[i] = list(zip([voc[j] for j in Di.indices], Di.data))

    return features
```

07 完成关键词提取

接着，就可以完成关键词的提取了。

考虑到关键词提取的全面性，分别提取unigram、bigram和trigram关键词。

如果提取unigram关键词，那么需要做词形还原。

```
def get_ngram_keywords(docs_tokenized,topk=5,n_=2):

    """ 1：处理为n-gram """
    docs_ngrams = [gene_ngram(doc,n=n_,m=n_) for doc in docs_tokenized]

    """ 2：按停用词表和词性，过滤 """
    docs_ngrams = [clean_ngrams(doc) for doc in docs_ngrams]

    docs_ = []
    for doc in docs_ngrams:
        docs_.append(' '.join(['_'.join(ngram) for ngram in doc]))

    """ 3：计算tf-idf，提取关键词 """
    features = calcu_tf_idf(docs_)

    docs_keys = []
    for i,pair in features.items():
        topk_idx = np.argsort([v for w,v in pair])[::-1][:topk]
        docs_keys.append([pair[idx][0] for idx in topk_idx])

    return [[' '.join(words.split('_')) for words in doc ]for doc in docs_keys]


""" 五：抽取n-gram关键词 """
def get_keywords(docs_tokenized,topk):

    """ 1：英文单词拆分 """
    docs_tokenized = [list(chain.from_iterable(doc)) for doc in docs_tokenized]

    """ 2：提取关键词，包括unigram，bigram和trigram """
    docs_keys = []
    for n in [1,2,3]:
        if n == 1:
            """ 3：如果是unigram，还需要做词形还原 """
            docs_tokenized = [[lemmatizer.lemmatize(word) for word in doc] for doc in d

        keys_ngram = get_ngram_keywords(docs_tokenized, topk,n_=n)
        docs_keys.append(keys_ngram)

    return [uni+bi+tri for uni,bi,tri in zip(*docs_keys)]
```

ok，来看关键词提取的结果。

选取其中一篇新闻，关于新冠肺炎禁闭防范期间，降薪和裁员的影响。

```
""" 标题：在冠状病毒禁闭期间裁员或减薪有什么法律影响 """
What are the legal implications of layoffs or salary cuts during coronavirus lockdown
```

以下是提取的unigram、bigram和trigram关键词。

从关键词中，大致可以看出是关于降薪、削减成本、劳动者保护、公司决策。

遗憾的是，冠状病毒（Coronavirus）这个词没有提取出来，但是通过实体识别抽取了出来。

```
'keywords': ['employee',                 # 雇员
             'order',                    # 订货
             'employer',                 # 雇主
             'cut',                      # 削减
             'lockdown',                 # 一级防范禁闭（期）
             'scenario',                 # 方案
             'time',                     # 时期
             'salary',                   # 薪酬
             'organisation',             # 组织
             'startup',                  # 创业公司
             'cost reduction',           # 成本削减
             'legal implications',       # 法律影响
             'salary cuts',              # 降薪
             'employer beware',          # 雇主品牌
             'discretionary spending',   # 可自由支配的个人开支
             'activity declines',        # 活动减少
             'recommended philosophy',   # 被推荐的方式
             'practice group',           # 业务部门
             'population density',       # 人口密度
             'pay scales',               # 工资标准
             'advising startup founders',  # 建议创业者
             'broader startup ecosystem',  # 更广泛的创业生态系统
             'employers make payment',     # 雇主支付
             'ensure legal protection',    # 确保法律保护
             'face tough times',           # 面临艰难时期
             'garner sufficient caution',  # 获得足够的关注
             'health system preparedness', # 卫生系统的准备
             'lower pay scales',           # 更低的工资标准
             'seek legal advice',          # 寻求法律咨询
             'top management level']       # 公司高层
```

时间紧，关键词提取做得还不够深入，没有尝试更多方法。

写这篇文章，是为了整理思路，希望有小伙伴可以一起交流有效的提取短语的方法。

还记得上面那位美女吗？