上海对外经贸大学

机器学习与数据挖掘实战课程项目

基于词频统计法和 TextRank 算法的抽取式单文本自动摘要

学院：统计与信息学院

班级：应用统计 1701

姓名：朱强强

学号：17064001

2019 年 12 月

一、 项目简介

文本摘要任务旨在从一篇或多篇相同主题的文本中抽取能够迅速反映主题的精简压缩版本，可以帮助用户快速形成对特定主题文本内容的全面了解，提高浏览信息和获取知识的效率。文本摘要根据抽取方式的不同可分为抽取式摘要和生成式摘要。抽取式摘要直接从原文本中不加修改地抽取文本片段（通常是句子）组成摘要；生成式摘要是重新组织句子形成比抽取式摘要更加精简的形式。

目前中文生成式自动摘要的公开数据集只有 LCSTS 和 NLPCC 数据集，由于数据集申请并未收到回复，没有数据集训练中文生成式自动摘要模型，所以本项目为基于词频统计算法和 TextRank 算法的中文抽取式自动摘要。

二、 算法介绍

1. TF-IDF

判断句子重要性一个很常见的方法就是统计关键词在句子中出现的频率，而查找关键词最常用的方法之一就是 TF-IDF 算法。

$$词频(TF) = \frac{某个词在文章中的出现次数}{文章的总词数}$$

但是，仅仅根据词频抽出的关键词很可能是一些像“的”，“是”，“在”这一类最常用的词，这对结果毫无帮助，需要被过滤掉。

所以我们还需要一个重要性系数来进一步衡量一个词是不是常见词。如果某个词比较少见，但是它在这篇文章中多次出现，那么它很可能就反映了这篇文章的特性，这才是我们需要的关键词。这个重要性系数被称为逆文档概率（IDF）。

$$逆文档概率(IDF) = \log\left(\frac{语料库中的文档总数}{包含该词的文档数 + 1}\right)$$

$$TF - IDF = TF \times IDF$$

TF-IDF 与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比。计算出文档中每个词的 TF-IDF 值，按降序排列取排在最前面的几个词就是关键词。

2. 余弦相似性

余弦相似性通过测量两个向量的夹角的余弦值来度量它们之间的相似性。我们将句子通过词向量的方法转化为向量形式，然后计算不同向量之间的余弦值。余弦值越接近 1，就表明两个向量所代表的句子相似性越高。

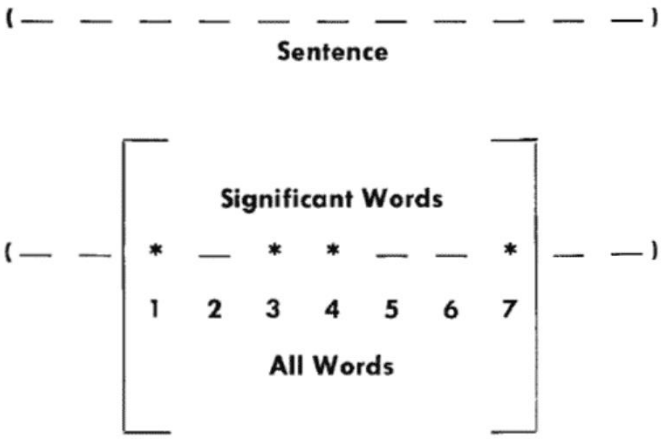
假设 A 和 B 是两个 n 维向量 A 是$[A_1, A_2, \cdots, A_n]$，B 是$[B_1, B_2, \cdots, B_n]$，则 A 与 B 的夹角θ的余弦等于：

$$cos\theta = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} = \frac{A \cdot B}{|A| \times |B|}$$

3. 词频统计方法

词频统计方法最早出自 1958 年的 IBM 公司科学家 H.P. Luhn 的论文《The Automatic Creation of Literature Abstracts》。

作者认为，文章的信息都包含在句子中，有些句子包含的信息越多，就说明这个句子越重要。Luhn 提出用“簇”来表示关键词的聚集。



Portion of sentence bracketed by and including significant words not more than four non-significant words apart. If eligible, the whole sentence is cited.

上图中被框起来的部分就是一个“簇”，只有关键词之间的距离小于一个阈值，它们就被认为处于同一个簇之中。Luhn 建议的阈值是 4 或 5，即如果两个关键词之间有 4 个或 5 个以上的关键词，就可以把这两个关键词分在两个簇中。

之后对于每一个簇，计算它们的重要性得分。

$$簇的重要性 = \frac{(包含的关键词数量)^2}{簇的长度}$$

找出包含分值最高的簇的句子，把它们结合在一起就构成了该文章的自动摘要。

4. TextRank 算法

TextRank 算法是一种基于图的用于文本自动摘要的排序算法，通过把文本分割成若干个组成单元（句子），构建节点连接图，用句子之间的相似度作为边的权重，通过循环迭代计算句子的 TextRank 值，最后抽取排名高的句子合成文本摘要。

$$TR(v_i) = \frac{1-d}{n} + d(\sum_{v_j \in In(v_i)} \frac{w_{ji}}{\sum_{v_k \in Out(v_j)} w_{jk}} TR(v_j)), i = 1,2,\cdots,n$$

d是阻尼系数，代表从某一句子指向其他任意句子的概率，一般设置为 0.85。

w_{ji}为边的权重，也就是句子的之间的相似度。

$In(v_j)$是指向句子v_j的句子集合。

$Out(v_j)$是句子v_j指出去的句子集合。

三、 算法思路

词频统计法

1. 进行文本预处理，将文本切割成单个句子，并进行分词。
2. 去除停用词，使用的是哈工大公开的停用词表。
3. 利用 TF-IDF 算法找出关键词。本文由于缺少语料库，无法计算逆文档概率，故仅使用 TF 寻找关键词。
4. 计算簇的重要性得分，并为句子的重要性打分并降序排序，由此得出的前 n 个句子组成该文档的摘要。

TextRank 算法

1. 文本预处理步骤同词频统计法。
2. 加载经过训练的 word2vec 词向量，并由此得到句子的向量表示。
3. 计算句子之间的余弦相似度，构成相似度矩阵。
4. 利用句子相似度矩阵构建图结构，句子为节点，句子相似度为转移概率。
5. 得到所有句子的 TextRank 值，并由此对句子进行排序，由此得出的前 n 个句子组成该文档的摘要。

四、 算法结果

----------top_n_summary----------

中共中央政治局常委、国务院总理李克强日前就在自由贸易试验区开展“证照分离”改革全覆盖试点作出重要批示。

批示指出：推进“证照分离”改革，是深化“放管服”改革、优化营商环境、更大激发市场活力的重大举措。

必须持续推进市场化、法治化、国际化营商环境建设，更大力度推进“放管服”改革，扎实做好“证照分离”改革试点扩面工作，着力推进照后减证和简化审批，

坚决克服“准入不准营”现象，促进更多新企业开办和发展壮大。

韩正强调，要高质量开展自贸试验区“证照分离”改革全覆盖试点，提升改革工作成效。

要把握好“进四扇门”的办法，按照直接取消审批、审批改为备案、实行告知承诺、优化审批服务等方式，从实际出发，分类实施、稳步推进改革。

----------mean_scored_summary----------

批示指出：推进“证照分离”改革，是深化“放管服”改革、优化营商环境、更大激发市场活力的重大举措。

韩正强调，要高质量开展自贸试验区“证照分离”改革全覆盖试点，提升改革工作成效。

----------TextRank----------

韩正表示，推进“证照分离”改革，是深入贯彻落实党的十九届四中全会精

神，推进国家治理体系和治理能力现代化的一项重要基础性工作。
必须持续推进市场化、法治化、国际化营商环境建设，更大力度推进“放管服”改革，扎实做好“证照分离”改革试点扩面工作，着力推进照后减证和简化审批，
坚决克服“准入不准营”现象，促进更多新企业开办和发展壮大。
韩正强调，要高质量开展自贸试验区“证照分离”改革全覆盖试点，提升改革工作成效。
批示指出：推进“证照分离”改革，是深化“放管服”改革、优化营商环境、更大激发市场活力的重大举措。
要把握好全覆盖这个基本要求，建立清单管理制度，将涉企经营许可事项全部纳入清单管理，明确改革目标，增加改革透明度，倒逼改革深化。

两个算法得出的摘要结果存在一定差异，但目前对抽取式摘要精确度还并没有一个统一、精确的度量方法，只能靠人工判别。

Code

```
# -*- coding:utf-8 -*-

import nltk
import jieba
import codecs
import numpy as np
import networkx as nx
from sklearn.metrics.pairwise import cosine_similarity


# 文本预处理
def sent_tokenizer(texts):
    start = 0
    i = 0  # 每个字符的位置
    sentences = []
    punt_list = ".!?。！？；"
    for text in texts:
        if i < len(texts) - 1:
            if text in punt_list:
                sentences.append(texts[start:i + 1])  # 当前标点符号位置
                start = i + 1  # start标记到下一句的开头
                i += 1
            else:
                i += 1  # 若不是标点符号，则字符位置继续前移
        else:
            sentences.append(texts[start:])  # 处理文本末尾没有标点符号的情况
            break
    return sentences


# 停用词
def load_stop_words_list(path):
    stop_list = [
        line.strip()
        for line in codecs.open(path, 'r', encoding='utf8').readlines()
    ]
    stop_words = {}.fromkeys(stop_list)
    return stop_words


# 摘要
def summarize(text):
```

```
    stopwords = load_stop_words_list("stopwords.txt")
    sentences = sent_tokenizer(text)
    words = [
        w for sentence in sentences for w in jieba.cut(sentence)
        if w not in stopwords if len(w) > 1 and w != '\t'
    ]
    word_frequency = nltk.FreqDist(words)
    top_words = [
        w[0] for w in sorted(
            word_frequency.items(), key=lambda d: d[1],
reverse=True)
    ][:n]
    scored_sentences = _score_sentences(sentences, top_words)
    # 利用均值和标准差过滤非重要句子
    avg = np.mean([s[1] for s in scored_sentences])
    std = np.std([s[1] for s in scored_sentences])
    mean_scored = [(sent_idx, score) for (sent_idx, score) in
scored_sentences
                   if score > (avg + 1.5 * std)]
    top_n_scored = sorted(scored_sentences,
                          key=lambda s: s[1])[-
num_top_sentence:]
    top_n_scored = sorted(top_n_scored, key=lambda s: s[0])
    return dict(
        top_n_summary=[sentences[idx] for (idx, score) in
top_n_scored],
        mean_scored_summary=[sentences[idx] for (idx, score) in
mean_scored])


# 句子得分
def _score_sentences(sentences, top_words):
    scores = []
    sentence_idx = -1
    for s in [list(jieba.cut(s)) for s in sentences]:
        sentence_idx += 1
        word_idx = []
        for index in range(len(s)):
            if s[index] in top_words:
                word_idx.append(index)
            else:
                pass
        word_idx.sort()
        if len(word_idx) == 0:
            continue
        # 对于两个连续的单词，利用单词位置索引，通过距离阈值计算簇
        clusters = []
        cluster = [word_idx[0]]
        i = 1
```

```
        while i < len(word_idx):
            if word_idx[i] - word_idx[i - 1] < cluster_distance:
                cluster.append(word_idx[i])
            else:
                clusters.append(cluster[:])
                cluster = [word_idx[i]]
            i += 1
        clusters.append(cluster)
        # 对每个簇打分，每个簇类的最大分数是对句子的打分
        max_cluster_score = 0
        for c in clusters:
            key_words = len(c)
            total_words = c[-1] - c[0] + 1
            score = key_words * key_words / total_words
            if score > max_cluster_score:
                max_cluster_score = score
        scores.append((sentence_idx, max_cluster_score))
    return scores


# TextRank算法
def text_rank_summarize(text):
    stopwords = load_stop_words_list('E:/Python Files/stopwords/中文停用词表.txt')
    sentences = sent_tokenizer(text)
    words_list = []
    for sentence in sentences:
        word_list = [
            w for w in jieba.cut(sentence) if w not in stopwords
            if len(w) > 1 and w != '\t'
        ]
        words_list.append(word_list)

    # 加载word2vec词向量
    word_embeddings = {}
    with codecs.open("sgns.renmin.char", encoding='utf-8') as f:
        for line in f:
            # 把第一行的内容去掉
            if '355996 300\n' not in line:
                values = line.split()
                word = values[0]
                embedding = np.asarray(values[1:], dtype='float32')
                word_embeddings[word] = embedding
        f.close()

    # 得到句子的向量表示
    sentence_vectors = []
    for word_list in words_list:
```

```
        if len(word_list) != 0:
            # 如果句子中的词语不在字典中，那就把embedding设为300维元素为0的向量。
            # 得到句子中全部词的词向量后，求平均值，得到句子的向量表示
            v = sum(
                [word_embeddings.get(w, np.zeros([
                    300,
                ])) for w in word_list]) / (len(word_list))
        else:
            # 如果句子为[]，那么就向量表示为300维元素为0个向量。
            v = np.zeros([
                300,
            ])
        sentence_vectors.append(v)

    # 计算句子之间的余弦相似度，构成相似度矩阵
    sim_mat = np.ones([len(words_list), len(words_list)])
    for i in range(len(words_list)):
        for j in range(len(words_list)):
            if i != j:
                sim_mat[i][j] = cosine_similarity(
                    sentence_vectors[i].reshape(1, 300),
                    sentence_vectors[j].reshape(1, 300))[0, 0]

    # 利用句子相似度矩阵构建图结构，句子为节点，句子相似度为转移概率
    nx_graph = nx.from_numpy_array(sim_mat)

    # 得到所有句子的TextRank值
    scores = nx.pagerank(nx_graph)

    # 根据TextRank值对句子进行排序
    ranked_sentences = sorted(
        ((scores[i], s) for i, s in enumerate(sentences)), reverse=True)

    result = []
    for i in range(num_top_sentence):
        result.append(ranked_sentences[i][1])

    return result


# 读取文本文件
def read_text(path):
    with codecs.open(path, "r", encoding="utf-8") as f:
        text = f.read()
    return text
```

```
if __name__ == '__main__':
    n = 30  # 关键词数量
    cluster_distance = 5  # 单词间的距离
    num_top_sentence = 5  # 返回句子的数量
    text = read_text("text.txt")
    algorithm1 = summarize(text)
    algorithm2 = text_rank_summarize(text)
    print('--------------top_n_summary----------------')
    for sent in algorithm1['top_n_summary']:
        print(sent)
    print('-----------mean_scored_summary-------------')
    for sent in algorithm1['mean_scored_summary']:
        print(sent)
    print('----------------TextRank-------------------')
    for sent in algorithm2:
        print(sent)

```