Keywords Extraction Using TF-IDF Algorithm

Peng Li, 10175501102

School of Data Science and Engineering, East China Normal University

ABSTRACT

This project is aimed at extracting the keywords of the provided text data using the Term Frequency–Inverse Document Frequency (TF-IDF) algorithm. The data set contains 1357 documents saved as txt files while each document involves a Chinese paper in the field of computers. The data set is so dirty which contains not only Chinese terms but also English words, codes, mathematics formulas and some other symbols. The primary issue with this project is cleaning these data, and it is also important to maintain modularity while improving efficiency. In this project I cleaned the text data, extracted the top 10 keywords of each document using the TF-IDF algorithm and saved them into the result file.

Introduction to TF-IDF

In information retrieval, tf—idf or TF-IDF, short for term frequency—inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf—idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

In the case of the term frequency $\mathrm{tf}(t,d)$, the simplest choice is to use the raw count of a term in a document, i.e., the number of times that term t occurs in document d:

$$\mathrm{tf}(t,d) = \frac{n_{t,d}}{\sum_k n_{t,d}} \tag{1}$$

Where $n_{t,d}$ is the number of times the term t appears in the document d and the denominator represents the sum of the occurrences of all terms in the document.

The inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$\mathrm{idf}(t,D) = log\frac{N}{|\{d \in D : t \in d\}|} \tag{2}$$

with

- N: number of documents in the corpus $N = |D|$
- $|\{d \in D : t \in d\}|$: number of documents where the term t appears (i.e., $\mathrm{tf}(t,d) \neq 0$). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator to $1 + |\{d \in D : t \in d\}|$.

Then tf—idf is calculated as

$$\text{tf}-\text{idf}(t,d,D) = \text{tf}(t,d) \times \text{idf}(t,D) \tag{3}$$

A high weight in tf—idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms. Since the ratio inside the idf's log function is always greater than or equal to 1, the value of idf (and tf—idf) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the idf and tf—idf closer to 0.

Experiment

The experimental process and some matters needing attention are described as follows

Environment

- Python 3.6.4
- Numpy 1.14.2
- Pandas 0.22.0
- Jieba 0.39
- NLTK 3.2.5

Data Set Description

This project is provided with 1357 documents and each document involves a Chinese paper in the field of computers. The documents are so dirty that you have to clean them first. The documents contain not only Chinese terms but also English words, codes, mathematics formulas and some other symbols. Besides, some disturbing line breaks and carriage returns appears in some documents which means that you should be careful when you segmented the words.

Load Data

This section loads all the documents into a list. It should be noted that the encoding format of the file should be 'gb18030', otherwise some errors will occur. I spent a lot of time dealing with encoding format.

Text Cleaning

In this section I mainly deleted all the numbers in the document. Some special symbols will be processed when the sentence is segmented. It's important to note that you can't delete all English words because some documents contain a lot of English words, such as the file 'C19-Computer0877'.

Sentence Segmentation

In this part, I segmented the sentence using the 'Jieba' package. Note that you can't remove all the spaces before this because you have to extract English words. However, at the same time, you have to remove some spaces, because there are some spaces in the Chinese terms, which leads to that the tool for extracting words can not recognize these terms, i.e., ' 摘要' in ' 摘　要'. My approach is to extract Chinese terms and English words separately using the regular expression. Besides, I used the 'NLTK' package to extract the stem of the word when dealing with English words.

Besides, I assumed that the length of the keyword is greater than 1, so I deleted the words and symbols with a length equal to 1.

Calculate the Term Frequency (TF)

In this section I calculate the number of times each term t appears in the document d first, just as $n_{t,d}$ described in the formula (1). At the same term, for each word I figured out how many documents it appeared in, this will help the calculations that follow. After that, for the number of times each term appears in the document, I divided it by the total number of terms in each document and got the Term Frequency.

Calculate the Inverse Document Frequency (IDF)

Since I have previously calculated how many documents each word has appeared in, it's easy to get the result of the formula (2) here. Because my previous operation guarantees that the words here will appear in the corpus, so there is no need to add 1 here.

Get the Term Frequency-Inverse Document Frequency (TF-IDF)

After obtaining the result of the formula (1) and the result of the formula (2), I only have to do a multiplication here to obtain the result described in the formula (3).

Get the Top K Keywords

For each document, I got the sorted result using the 'sorted()' function and the lambda expression in python. It should be noted that there may not be K words in the document, so I have done some processing here.

Save the Result

Finally, I used the 'writelines()' function to save the final result to a txt document. Each line in the file contains the name and the first 10 keywords of each document.

Result

The keywords of the first 5 documents are listed in the table 1

Document	Top 10 Keywords
C19-Computer0003.txt	协同工作, 群体, 智能, 通信, 推理, 协同, 知识, 组织, 成员, 环境
C19-Computer0005.txt	教师, 线性表, 远程教学, 搜索, 编码, 社区, 分类, 教学内容, 结点
C19-Computer0007.txt	服务器, 接口技术, 数据库, 驱动程, 浏览器, 厂商, 接口, 文档, 编程, 连接
C19-Computer0009.txt	线程, 同步, 对象, 阻塞, 单写, 重置, 串行化, 释放, 单个, 申请
C19-Computer0011.txt	课件, 超文本, 使用者, 导航, 远程教育, 学习, 学习材料, 参与度, 材料, 链接

Table 1. Five samples of the result

All results are in the file 'result.txt', which also gives the TF-IDF value of each term.

I finally found that English words are not conducive to the calculation of TF-IDF, because the IDF value of English words is often very high. So I did not count the English words in the end.

In addition, it can be found from the results that there are no keywords in some documents, such as Document 'C19-Computer2411.txt'. After opening the original document you will find that this is an error when collecting data.

Conclusion

Although this experiment is simple, I have learned a lot from it. It took me half an hour to successfully solve the problem of the encoding format of the file. After observing some of the data, especially after discovering some disturbing spaces, I adjusted the way I processed the data. In the experiment, I used the knowledge of stem extraction and sentence segmentation, which are the contents taught in the class. I also used some tricks to improve the efficiency of the code while maintaining modularity, even though it was just a few simple improvements.

References

- TF-IDF, Wikipedia