XI'AN JIAOTONG-LIVERPOOL UNIVERSITY

INT401: Fundamentals of Machine Learning

Assignment 1: Feature Generation

Yihan Fan 2359022

November 5, 2023

1 EXPERIMENTAL OBJECTIVES

The aim of this experiment is to use **TFIDF** representation maps a text into a numeric representatio in python environment, to get the TFIDF matrix of the text content, so as to get the text keywords for text classification and analysis.

1.1 Datasets

The dataset consists of 2726 texts which are pre-divided into 5 subdirectories, the name of each subdirectory is their class label.

1.2 Experimental environment

This experiment runs on the Windows version of python (3.11) with the interface shown in Figure 1.1.

- Python: 3.11
- Python IDE: Pycharm
- LaTex IDE: Texstudio

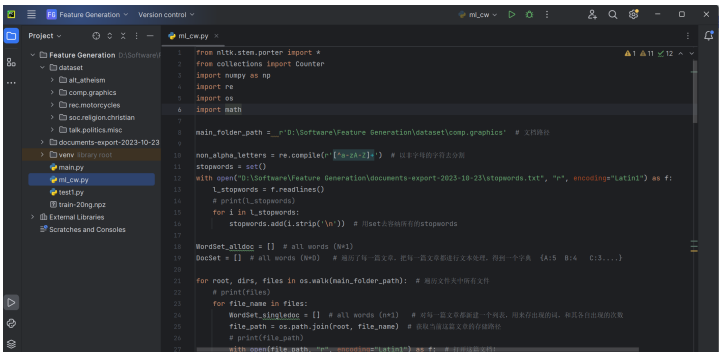

Figure 1.1: Schematic of the python runtime environment.

2 PRINCIPLE AND ROLE OF THE ALGORITHM

2.1 TFIDF principle

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical method for evaluating the importance of a word in a collection of documents. Its calculation is based on two main concepts: Term Frequency (TF) and Inverse Document Frequency (IDF).
TF measures how often a word appears in a document. In general, TF is defined as:

$$TF(t,d) = \frac{cnt\_td}{cnt\_d}$$

- cnt_td: Number of occurrences of word t in document a
- cnt_d: Total number of words in document d

IDF measures the importance of a word to the entire collection of documents. In general, IDF is defined as:

$$IDF(t,D) = \log(\frac{N}{nt})$$

- N: The total number of documents in the document collection D
- nt: Number of documents containing word t

The product of TF and IDF gives the importance of a word in the document. This is obtained by multiplying the word frequency and the inverse document frequency:

$$TFIDF(t,d,D) = TF(t,d) \times IDF(t,D))$$

For each word in a document, its corresponding TF-IDF value can be calculated. The TF-IDF vector of the document will contain the TF-IDF weights for all words.

2.2 Role of TFIDF

a. *Filtering Common Words*: TF-IDF is able to reduce the weight of common words, which may occur frequently in a document but are not important for distinguishing the document and providing useful information.

b. *Highlighting Keywords*: TF-IDF can highlight words that appear frequently in a document, but less frequently in the overall document set, and which are often important for understanding the content of the document.

c. *Document similarity*: TF-IDF can be used to compare the similarity between documents. By calculating the TF-IDF weights of the words in a document, the similarity between them can be measured, which is useful in tasks such as information retrieval and text clustering.

This experiment mainly fulfils the first and second roles.

3 Experimental Procedures (with codes)

The process of this experiment mainly includes data reading, document text processing, TFIDF representation, and result analysis.

3.1 Preprocessing

Firstly, read the dataset and the stopwords information and create a regular expression object named *non_alpha_letters* to match non-alphabetic characters in the string, create a collection to hold all the stopwords, and create a dictionary to hold the text information that will be processed later.

```
main_folder_path = r'D:\Software\Feature Generation\dataset\comp.graphics'  # 文档路径

non_alpha_letters = re.compile(r'[^a-zA-Z]+')
stopwords = set()
with open("D:\Software\Feature Generation\documents-export-2023-10-23\stopwords.txt", "r", encoding="Latin1") as f:
    l_stopwords = f.readlines()
    # print(l_stopwords)
    for i in l_stopwords:
        stopwords.add(i.strip('\n'))  # 用set去容纳所有的stopwords

WordSet_alldoc = []  # all words (N*1)
DocSet = []  # all words (N*D)   # 遍历每一篇文章，把每一篇文章都进行文本处理，得到一个字典  {A:5  B:4   C:3....}
```

Next, loop traversal is applied, in which conditional statements are used to exclude words from stopwords, split() splits the text information, lower() converts all words to lowercase form, strip() removes all non-alphabetic characters, and stemmer.stem(word) removes word suffixes. Finally the processed text information is stored in singledoc dictionary.

```
for root, dirs, files in os.walk(main_folder_path):  # 遍历文件夹中所有文件
    # print(files)
    for file_name in files:
        WordSet_singledoc = []  # all words (n*1)   # 对每一篇文章都新建一个列表，用来存出现的词，和其各自出现的次数
        file_path = os.path.join(root, file_name)  # 获取当前这篇文章的存储路径
        # print(file_path)
        with open(file_path, "r", encoding="Latin1") as f:  # 打开这篇文档：
            content = f.readlines()  # 读取文档里面的内容，内容存到content里
            for i in content:  # 这里 i 是原文里每一行的内容，不做任何处理，包含各种字符、空格
                for item in non_alpha_letters.split(i):  # item指的是划分完之后具体的单词，non_alpha_letters.split(i)为了划分词
                    if item.lower() not in stopwords and item != '':
                        item = re.sub( pattern: r'[^a-z]', repl: '', item.lower()).strip()  # 只保留字母、转成小写      "   cab
                        # dataset_all.append(re.sub(r'[^a-z]', '', item.lower()).strip())
                        WordSet_singledoc.append(item)  # 放进当前这篇文档对应的列表里    【A,B,B,A,C,GF,SDF,ASD,ASD】
                        WordSet_alldoc.append(item)  # 放进所有文章对应的词汇表里面
            stemmer = PorterStemmer()
            WordSet_singledoc = [stemmer.stem(word) for word in WordSet_singledoc]  # 去除词的后缀，找词根
```

3.2 TFIDF Representation

Counting the total number of words, the total number of documents, all the words appearing in all the articles are put into a list, generating a total dictionary of all the words appearing in all the articles and the number of times they each appear.

```
d_wordcnt_alldoc = {}
word_counts = Counter(WordSet_alldoc)
for word, count in word_counts.items():
    d_wordcnt_alldoc[word] = count  # 生成一个总的字典，把所有文章里出现的词和各自出现的次数都统计出来

N = len(DocSet)  # 数据集里总共有N篇文章
D = len(d_wordcnt_alldoc)  # 所有文章里出现的词的个数，一共有D个词

WordSet_final = list(d_wordcnt_alldoc)  # 把所有文章里出现的词都放进一个列表里
WordSet_final.sort()  # 根据首字母对词汇做排序

d_final = {}
for i in WordSet_final:
    d_final[i] = 0
```

Calculate the TF value: divide the number of occurrences of a word by the total number of words in the document.

```
d_wordcnt_singledoc.values()
total_words_num = sum(d_wordcnt_singledoc.values())  # 统计当前这篇文章里一共有多少词数，包含重复词
if k not in d_wordcnt_singledoc:
    return 0

TF = d_wordcnt_singledoc[k] / total_words_num  # f_ik
```

Calculation of IDF, a_ik value: first of all, through the loop traversal of all documents in the number of documents nk appeared in the vocabulary k, and then through the formulan $log(N/nk)$ to calculate the IDF, at this time, if nk == 0, according to the relevant literature on the nk to do *+1* processing, does not affect the results. Then according to $a\_ik = TF * IDF$ to get a_ik value.

```
nk = 0  # 所有文档里，出现过词汇k的文档数目
for i in DocSet:
    if k in i:
        nk += 1
if nk != 0:
    IDF = math.log((N / nk), base: 10)
elif nk == 0:
    IDF = math.log((N / (nk + 1)), base: 10)

a_ik = TF * IDF
return a_ik
```

Calculate Aik: Create an empty list. Calculate $\sqrt{\sum_{j=1}^{D}(a_{ij})^2}$ first, then store all the numerators in the list l_A_row_temp, and when all the words in this article have been traversed, divide each vector (numerator) by the denominator temp_val to get Aik.

```
l_A = []
cnt = 0
for l_singledoc in DocSet:
    l_A_row_temp = []
    temp_val = 0
    for word in WordSet_final:  # 总词库里的每个词，在当前这篇文档里出现的次数
        a_ij = cal_aik(word, len(DocSet), l_singledoc, DocSet)
        temp_val += a_ij ** 2  # 求平方
        vector = cal_aik(word, len(DocSet), l_singledoc, DocSet)
        l_A_row_temp.append(vector)
    temp_val = pow(temp_val, 0.5)  # 开根号，算出来就是分母 对一篇文章来说，不论是针对哪个词，分母都是一样的
    l_A_row = list(np.array(l_A_row_temp) / temp_val)  # 先把分子都存进列表l_A_row_temp，等这篇文章所有词都遍历完，给每一个向量
    l_A.append(l_A_row)
    t = 0
    for i in d_final:
        d_final[i] += l_A_row[t]
        t += 1
    cnt += 1
    print(cnt)
```

4 Experimental Results and Analysis

Run the code to generate a matrix of text documents for the entire dataset and read the .npz file to display the matrix A. The results are as follows:

```
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.03196617 0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

Because of the huge amount of information, which leads to the omission of the matrix data, the following screenshot shows a part of the matrix.

```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.03196617, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.06393234, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ]])
```

In order to facilitate the observation of the data results and to analyse the practical guidance, I add the code to count the data of the top 20 of Aik, the code is as follows:

```
res = sortedDictValues(d_final)  # 分析出现次数最高的20个词
print(res[:20])
```

Calculate the text data under the first subfolder alt_atheism, and run the code with the following result:

```
('keith', 12.765823125604044)('moral', 12.063093461099283)('god', 11.90245567783616)('islam', 10.355041647760874)('caltech', 10.178838591257476)
('sgi', 9.591434971610047)('livesey', 9.32857286772681)('peopl', 7.760700879988843)('object', 7.745390573944153)('atheist', 7.45676505520996)
('exist', 7.318717244081351)('religion', 7.015259211674687)('atheism', 6.827895174636962)('jon', 6.776248644697513)('solntz', 6.63316608186648)
('wpd', 6.63316608186648)('sandvik', 6.552367496223976)('muslim', 6.4614920135019975)('cwru', 6.460854174467262)('don', 6.419648147996624)
```

By running the results, it can be seen that the texts in the first subfolder containing the words "God", "Islam", "atheist", " Atheist" have high Aik values and seem to involve discussions related to religion and theology, suggesting that this collection of texts mostly focuses on religion and theology. In addition, this result also allows for further categorical predictions related to the texts under this subfolder. For example, with the terms "moral", "livesey", and "peopl" it is possible to predict texts that may involve discussions on moral and ethical dimensions.

Calculate the text data under the second subfolder comp.graphics, the result of running the code is shown below:

```
('imag', 10.157129741623699)('file', 9.39666640697109)('graphic', 9.327913269293491)('polygon', 8.02671350177839)('point', 7.762380601098098)
('program', 7.370970506456774)('color', 6.958728592303711)('ac', 6.837971606108484)('uk', 6.740688477168303)('window', 6.553318122854117)
('bit', 6.252963222576307)('format', 6.196892284348102)('ibm', 6.112399169055894)('packag', 6.053925264725813)('univers', 6.025668356957321)
('tiff', 5.796938875383963)('write', 5.709709916039597)('version', 5.669030748545133)('softwar', 5.5466889785460415)('articl', 5.505673435979175)
```

The results show that the texts in this collection are talking about topics related to "images" and that the topics in this collection are more uniform. From this result we can further refine the classification, e.g.: "image", "file", "graphic", "polygon", "point", which indicate that the text may be related to graphics processing and image formats; "programme", "ac", "window", "bit", "format", "ibm", "package", "software" These terms indicate that the text may be related to computer programs and software; "colour" indicates that the text may be related to image colours and attributes.

Calculate the text data under the third subfolder rec.motorcycles and run the code as shown below:

```
('bike', 9.519285986533847)('ride', 8.20402021596317)('dog', 8.099228513873294)('motorcycl', 7.735246918239O5)('helmet', 7.651278428582937)
('bnr', 7.397473422776942)('sun', 7.336182063124102)('uk', 6.74377708365558)('rider', 6.716415527386943)('org', 6.594835782966221)
('behanna', 6.159000111438772)('nec', 6.134650724621972)('don', 6.094950113490659)('bmw', 6.006764722035925)('apr', 5.952881714420067)
('ac', 5.8922111281831455)('univers', 5.851820348981126)('insur', 5.673925118761286)('cs', 5.66749004979361)('good', 5.6377786298850845)
```

It can be seen that the topic of discussion in these texts would be motorbike-related activities." bike", "ride", "motorcycle", "rider" indicate discussions that may be about riding and motorbike activities; "helmet", "bnr" suggest discussions that may be about riding safety and related equipment; "bmw" may point to discussions involving BMW motorbikes; "sun", "apr" imply discussions that may be related to the weather, an event or a discussions related to weather, events, or a particular point in time.

Calculate the text data under the fourth subfolder soc.religion.christian and run the code with the result shown below:

```
('god', 13.044741548624973)('christian', 11.274129112786564)('jesu', 10.942023902962O5)('hell', 10.88700349907974)('church', 10.133054203304004)
('truth', 8.7880019184944666)('sin', 8.644508048023598)('faith', 8.594468842971731)('exist', 8.485063544486474)('bibl', 8.277269884123726)
('peopl', 7.832819707347751)('christ', 7.368095446957219)('absolut', 7.259954875316981)('belief', 7.035504034456127)('question', 6.851339993978379)
('law', 6.819225291688756)('love', 6.627073871134177S)('don', 6.607294099736581)('atheist', 6.601502131541247)('live', 6.372204759487023)
```

As you can see, the keywords in this set of texts are related to "Christianity", "Christmas", and "religion". god", "christian", "Jesus", "church", "faith", "christian", "belief", "sin" suggest possible discussions about religion and Christian beliefs; "truth", "law", "love", "absolute" suggesting possible discussions about morality and ethics; "exist", "question", "atheist", "live" suggesting possible discussions about existence, questions of faith and atheism.

Calculate the text data under the fifth subfolder talk.politics.misc and run the code with the result as shown below:

```
('cramer', 9.690883169776175)('optilink', 8.452288789395942)('drug', 8.319218653287853)('tax', 7.583934321433169)('homosexu', 7.302151566907953)
('govern', 6.828976085170047)('gay', 6.648482037754256)('kaldi', 6.647226548539526)('clinton', 6.617227838372385)('men', 6.605590811402112)
('ohio', 6.210099431047058)('peopl', 6.169531535520251)('isc', 6.094773013768759)('virginia', 6.078498079514098)('br', 5.973109585647582)
('job', 5.954555725805596)('rutger', 5.821089973637834)('clayton', 5.799915231914601)('presid', 5.697812420980433S)('don', 5.307187727209639)
```

It can be seen that this group of texts is mainly talking about topics related to "politics". For example, the words "cramer", "tax", "govern", "clinton", and "president" suggest that there may be discussions about politics and specific political figures. In addition, if the scope of observation is expanded to include the top 30, other textual notes can be found, for example, "drug", "health", suggesting possible discussions of drugs, health or the medical field, and "law", "legal", suggesting possible discussions of law or legal issues.

REFERENCES

[1] Aas, K., & Eikvil, L. (1999). Text categorisation: A survey.

[2] Qaiser, S., & Ali, R. (2018). Text mining: use of TF-IDF to examine the relevance of words to documents.ăInternational Journal of Computer Applications,ă181(1), 25-29.

[3] Bafna, P., Pramod, D., & Vaidya, A. (2016, March). Document clustering: TF-IDF approach. Ină2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)ă(pp. 61-66). IEEE.