Xi'an Jiaotong-Liverpool University

西交利物浦大学

Lab 6: Feature Generation

Author Pin QIAN

Module **INT104**

Teacher Xiaobo Jin

Date **20th**/March/2

Introduction

For NLP related problems, the essential step is using an appropriate method to map text data into a numeric representation. Besides machine learning methods, the TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a set of documents. In this work, it needs to read and preprocess data from dataset, and calculate the number of unique words, term frequency, and inverse document frequency of the word.

Procedure & Implementation

Procedure

1. Load Data
   1) Read documents and split the text into words
   2) Read stop words
2. Preprocessing
   1) Convert all words to lower case
   2) Remove all non-alphabet characters
   3) Remove all words that occur in stop words
   4) Make word stemming
3. Computing Section
   1) Count unique words
   2) Compute the frequency of each word in its document
   3) Compute the document frequency
   4) Compute TF-IDF and save the result

Implementation

Firstly, read the dataset and stop words. To make sure stop words are not duplicate, set is used to prevent the situation.

```
# read data -- len(dataset) = 2726
PATH = "dataset/"
file_dirs = os.listdir(PATH)
dataset = []
for f in file_dirs:
    file_path = PATH + f + '/' # e.g. /dataset/alt.atheism/
    txt_dirs = os.listdir(file_path)
    for t in txt_dirs:
        txt_path = file_path + t  # e.g. /dataset/alt.atheism/49960
        data = codecs.open(txt_path,'r',encoding='Latin1')
        dataset.append(data.read().split())

# read stopwords -- len(stop_words) = 851
f = open('stopwords.txt', 'r', encoding='Latin1')
stop_words = set(f.read().split()) # make sure words are not duplicate
```

Second step is to do data preprocessing.

doc_p is used as document pointer to traverse the whole dataset;

word_p is used as word pointer to traverse all words in a document.

For each word, it is first converted to its lower case;

Then, check if it is in stop words list and if it is alphabet character, if it is not in stop words and is alphabet character, it will be stemmed and the *word_p* will move to next position, else this word will be deleted.

```
# data preprocessing
stemmer = PorterStemmer()
for doc_p in range(len(dataset)):
    word_p = 0
    for i in range(len(dataset[doc_p])):
        word = re.sub(r'[^a-z]', '', dataset[doc_p][word_p].lower()).strip() # convert to lower case
        if word not in stop_words and word.isalpha(): # remove stopwords and non-alphabet characters
            dataset[doc_p][word_p] = stemmer.stem(word) # make word stemming
            word_p += 1
        else:
            del dataset[doc_p][word_p]
```

Thirdly, preprocessed data in dataset will be put together in the *words_pool,* the convert it to set to get *unique_words* set.

```
# unique words counting
words_pool = []
for data in dataset:
    words_pool += data # merge all words
unique_words = set(words_pool) # use set to get unique words
```

Then, term frequency is calculated. In this step, a NumPy matrix is initialized to store term frequency of each word in each document. Each row of the matrix represents a document, and each column represents a word that exists in *unique_words*. Then, make a deep copy of tf as df. Elements in df matrix which larger than 1 will be assigned to 1, then document frequency can be obtained by summing up column by column.

```
D = len(unique_words) # D = 27742
N = len(dataset) # N = 2726

# term frequency
tf = np.zeros(shape = (N, D)) # a numpy array storing term frequency
col = 0
for word in unique_words:      # for a word in unique_words
    row = 0                    # row starts from 0
    for doc in dataset:        # for each document in dataset
        tf[row, col] = doc.count(word)  # fill in with the count number of the word
        row += 1
    col += 1

# document frequency
df = tf.copy()  # copy tf as df to avoid pass reference
df[df > 0] = 1  # assign number which larger than 1 to 1
df = np.sum(df, axis = 0) # sum up colunm by colunm to get docement frequency
```

Finally, with the help of the broadcast mechanism of NumPy, the result, a matrix with shape (2726, 27742), can be obtained by multiplying the following two. In the end, each row is normalized, which represents numerical features of each document.

```
# result
A = tf * np.log(N / df)   # mutiply TF and IDF
A /= np.linalg.norm(A, axis=1, keepdims=True) # normalize the representation row by row

# save
np.savez('train-20ng.npz', X=A)
```

The result of the matrix is too large to render here, so the train-20ng.npz file will be submitted together with code and report.

```
In [8]: A = tf * np.log(N / df) # mutiply TF and IDF
        A /= np.linalg.norm(A, axis=1, keepdims=True) # normalize the representation row by row
        A

Out[8]: array([[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.]])
```

Conclusion

In conclusion, the TF-IDF method is applied in this lab assignment. For preprocessing stage, the text files from 5 subdirectories are read and split into words; all words are converted into lower case form and non-alphabet letters and stop words are removed; word stemming is performed on each legal word. For TF-IDF representation stage, the frequency of each word in its document and document frequency is calculated; and the normalization step is also completed.