INT304: LAB1 REPORT

Seungyeop Kang
Student ID: 1825478

ABSTRACT

Text classification is a process of categorizing the given documents into a specific class, using Natural Language Processing (NLP). Through NLP, texts can be automatically analysed and assigned to a set of predefined categories based on the contexts. However, the large number of documents can cause the high dimensionality of the word space that is problematic sometimes. Thus, this report will use the feature selection to choose the words that have the high Information Gain (IG) values and test it by calculating the TF-IDF values in the training and test dataset.

1 INTRODUCTION

Thanks to the development of Artificial Intelligence (AI), the AI-based programs can handle various tasks without human's manual labour. Natural Language Processing (NLP) is one of the benefits of AI that allows the machines to read, understand and deliver the meaning of texts automatically. Since the ability and accuracy of text processing have been improved significantly, NLP is widely used in countless fields, including healthcare, finance, human resource and media (Neupane, 2020).

Text classification is one of the most widely used method based on NLP. By using NLP, text classification can assign a label or class to the given texts. However, unlike human, computers have difficulty to read and understand the different formats of each text and grammar. Thus, the texts should be pre-processed before encoding. This procedure is called Data Preprocessing. From this step, every word in the given texts will be converted into its root and useless words will be removed as well. Thus, computers can handle the texts more easily based on this process. After this step, feature selection process can be applied if the number of texts is large because high dimensionality of the word space can cause problems and increase running time (Neupane, 2020).

1.1 FEATURE SELECTION

Generally, text classification requires a long time and computing power to get the features of every text. Therefore, the efficiency will be decreased rapidly if the dataset is large. In this case, feature selection will be helpful to compute the tasks efficiently. Feature selection is a method to select only informative words from the texts to improve efficiency and decrease the computation complexity. Normally, feature selection can be classified into 4 categories which are Filter, Wrapper, Embedded, and Hybrid methods. Filter method is based on the statistical analysis over the feature space to choose a discriminative subset of the features. On the other hand, Wrapper method will choose various subset of features to identify and evaluate using classifier. The feature selection method that is embedded into training phase of the classification is called the embedded approach. Lastly, the Hybrid approach is based on the advantages of both Filter and Wrapper. In this report, feature selection process will be based on the Filter method, using Information Gain (Chandra, 2018)

1.2 TF-IDF

After removing non-informative words from the dataset, the TF-IDF values of each word can be computed. Term Frequency (TF) - Inverse Document Frequency (IDF) is a method to evaluate how relevant the word is to the text in a category based on a statistical measure. Here, TF simply means the proportion of a word in a document. Then, the relevant words to the document can be easily found by selecting high TF words. However, there is a case that a word can occur in almost every document. This means that the word does not have the specific information about a document. Thus, the IDF value of a word can be computed by using a document frequency. Then,

those non-informative words will have low IDF values. By multiplying TF and IDF of a word, the most informative words can be selected by choosing words that have a high TF-IDF value. TF-IDF is often used in text classification for spam filtering, identifying fake news and real-time language translation

2 METHODOLOGY

2.1 DATA PREPROCESSING

Before starting the process, the required dataset **20news-bydate.tar.gz** should be downloaded. Then, the first step is to remove stopwords in this dataset. Stopwords are a set of commonly used words in a language that have no useful information. Generally, these words are used due to the grammar or repeated information. However, there are some cases when we should not remove stopwords, such as machine translation, language modelling, text summarization and question-answering problems. In this report, the stopwords in the given text file are used. After that, the non-alphabet characters in the dataset can be removed as well to increase the accuracy. However, although the words in the dataset are purely alphabetic, every word will have different formats due to the grammar in linguistics. Therefore, it is required to do text normalization to transform a word into a single canonical form. Generally, two methods called stemming and lemmatization are used in this task. Stemming will simply remove the suffixes from a word. On the other hand, lemmatization will find the root of a word through morphological analysis. Thus, the second method takes a longer time but provide higher accuracy. (Shubham, 2019).

2.2 FEATURE SELECTION

As mentioned before, this step is important to improve the efficiency of the task. In this report, the filter approach will be used, computing the IG values of each words and selecting the words have maximum IG values. Before computing the IG of each word, the low-frequency words in the document could be removed to reduce the computations. Here, words have the document frequency lower than 5 will be removed from the dataset. And then, we can compute IG values of the high-frequency words based on the equation below.

$$IG(w) = -\sum_{j=1}^{K} P(c_j)log(P(c_j))+P(w)\sum_{j=1}^{K} P(c_j|w)log(P(c_j|w))+P(\bar{w})\sum_{j=1}^{K} P(c_j|\bar{w})log(P(c_j|\bar{w})) \tag{1}$$

In equation 1, $P(c_j)$ is the number of documents in a category divided by the number of total documents in the traning dataset. $P(w)$ is the number of documents have the word w divided by the total document number. Also, $P(c_j|w)$ and $P(c_j|\bar{w})$ denote the number of documents that have the word w in a category divided by the document frequency of w and the number of documents that does not have w in the category divided by (the total number of documents in the dataset - the document frequency of a word).
After obtaining the IG of high frequency words, 1000 words with maximum IG values will be chosen as a output of feature selection. Detailed steps will be provided in the next section.

2.3 TF-IDF CALCULATION

Since the 1000 high-frequency words with maximum IG values are selected, the TF-IDF of these words can be computed lastly based on the equation below.

$$a_{ik} = f_{ik} * log(N/n_k) \tag{2}$$

In equation 2, term frequency, f_{ik} can be obtain by computing the number of words in a document divided by the total number of words in the document. Here, N and n_k denote the total number of documents in the dataset and the document frequency of a word, respectively.
After computing the TF-IDF values, these values should be normalized because the length of every document is different. Then, the final output can be obtained as a matrix of the normalized TF-IDF values from the dataset.

3 DESIGN AND RESULTS

In this section, the detailed steps and process of the self-designed algorithms will be explained step-by-step with the clipped running images. Since it is restricted to use any third-party library, the five functions below are defined first to call them when they are needed.

Algorithm 1 Function to remove non-alphabet characters: **non_alphabet()**

```
Input: List of the split texts
clear_word = []
for word in the input list do
    str_word = string type (word)
    low_word = lowercase(str_word)
    remov_na = only_alphabet(low_word)
    Append remov_na → clear_word[]
end for
Return clear_word
```

Algorithm 2 Function to convert dict() to []: **clr_docus_list()**

```
Input: Dictionary with nested lists
clr_docs = []
class_name = []
for key in the input dict() do
    for value in dict(key) do
        Append value → clr_docs[]
        Append key → class_name[]
    end for
end for
Return clr_docs, class_name
```

Algorithm 3 Function to find the low freq. words with document freq.: **find_low_freq()**

```
Input: List of words, nested list of the dataset
df = {}
low_words = []
count = 0
for word in the input list do
    for list of the input nested list do
        if word in list then
            count += 1
        else
            pass
        end if
    end for
    if count >= 5 then
        df{key: word, value: count}
    else
        Append word → low_words[]
    end if
    count = 0
end for
Return low_words, df
```

Algorithm 4 Function to delete low freq. words: **del_low_freq()**

```
Input: low_words, Dictionary with nested lists
for key in the input dict() do
    for word in low_words do
        for list in dict(key) do
            if word in list then
                for number of word in list do
                    remove word in list
                end for
            else
                pass
            end if
        end for
    end for
end for
Return The input dict()
```

Algorithm 5 Function to calculate the entropy: **get_entropy_sum()**

```
Input: List of the probability P(i)
entropy = 0
for probability in the input list do
    remov_na = only_alphabet(low_word)
    entropy += P(i)log(P(i))
end for
Return -entropy
```

When a list of words is inputted to Algorithm 1, the words in the list will be changed to lowercase and removed any non-alphabet characters in each item. Algorithm 2 will simply convert the dictionary with values in nested lists to a combined nested list and a list of its category names.
Algorithm 3 is an important function that takes a list of words without any repetition and nested list which includes all documents in the dataset. This function will finally output a list of low-frequency words which should be removed in the dataset and a dictionary of words with its document frequency. Based on these results, Algorithm 4 will remove the low-frequency words from the

dataset. Algorithm 5 is a function to calculate the entropy of probabilities. This function will be often used in the feature selection. The return value $-entropy$ is equal to the equation below.

$$-entropy = -\sum_{i=1}^{K} P(i)log(P(i)) \tag{3}$$

After declaring the self-defined functions, the training dataset of **20news-bydate.tar.gz** can be encoded in the program.

Figure 1: The loaded path of the training dataset and categories.

In the same way, the given stopwords can be also encoded in the program. To avoid any repeated words, the stopwords are stored in a set.

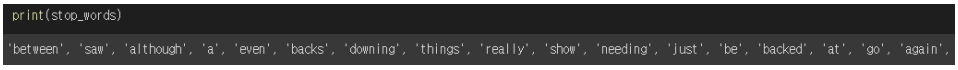


Figure 2: The loaded stopwords stored in a set.

Based on these loaded data, the data preprocessing can be started to remove stopwords, non-alphabet characters and the suffixes of each words for normalization. In this report, stemming method is used for text normalization, using the **nltk** library. Note that the output of this process will be all in lowercase letters.

Algorithm 6 Final step for Data Preprocessing

```
one_docu = []
all_docu = []
key_words = []
cate_list = []
cate_dict =
for category in the training dataset do
    Get the file path of category
    text_files = list of documents inside category
    for document in text_files do
        open and read the document as f
        for line in f do
            list_txt = list of split words in line
            one_docu += non_alphabet(list_txt)
            remov_one_docu = [word for word in one_docu if word not in stopwords]
            stem_words = [stemmed words in remov_one_docu]
        end for
        Append stem_words → all_docus
        key_words += stem_words (List of words allowing repetition)
        Append stem_words → cate_list
        Initialize one_docu[]
    end for
    cate_dict{key: category, value: cate_list}
    total_num = total number of documents in training dataset
    Initialize cate_list[]
end for
```

In this step, every line of documents will be split and stored into a list. This list of a line will be inputted to **non_alphabet()** to remove non-alphabet characters and convert them into lowercase

letters. And then, one_docu will store the output of this function to remove stopwords. Then, finally, the list without stopwords will be used for stemming. There are four different final outputs. *all_docus* is a nested list which includes every documents in the training dataset as $A = (a_{ik})$. *key_words* will be used later for feature selection. *cate_dict* is a dictionary that store the documents to their categories. *total_num* will be used for the calculation in feature selection.

Figure 3: A part of the training dataset after data preprocessing (all_docus).

From the above result, it cab be clearly checked that non-informative words are removed and the remained words are normalized through the previous step.
Based on these outputs, the dataset can be used for feature selection. The first required process in this process is removing the low-frequency words in the dataset.

Algorithm 7 To remove the low-frequency words and calculate the document frequencies

```
key_set = list(set(key_words))
low_freq = find_low_freq(key_set, all_docus)
low_words = low_freq[0]
df_dict = low_freq[1]
clr_cate_dict = del_low_freq(low_words, cate_dict)
```

As shown in Algorithm 7, this step simply calls the self-defined functions and store the returned data. Here, *df_dict* is a dictionary that store the words and its document frequency. *clr_cate_dict* is a dictionary of the whole documents in its category, but removed the low-frequency words. The following figure shows the printed result of *clr_cate_dict* in the terminal after executing this algorithm:

Figure 4: A part of the training dataset after removing low-frequency words (clr_cate_dict).

After removing the low-frequency words, the information gain of the words in the current dataset can be calculated. Firstly, the entropy of each class, $P(w)$ and $P(\bar{w})$ can be obtained as follows:

Algorithm 8 To compute the entropy of $P(c_j)$

```
Pcj_list = []
for category in clr_cate_dict do
    P(c_j) =
    len(clr_cate_dict[category]) / total_num
    Append P(c_j) → Pcj_list[]
end for
Ecj = get_entropy_sum(Pcj_list)
```

Algorithm 9 To compute $P(w)$ and $P(\bar{w})$

```
Pw_dict = {} and Pnotw_dict = {}
for word in df_dict do
    P(w) = df_dict[word] / total_num
    P(w̄) =1 - P(w)
    Pw_dict{key: word, value: P(w)}
    Pnotw_dict{key: word, value: P(w̄)}
end for
```

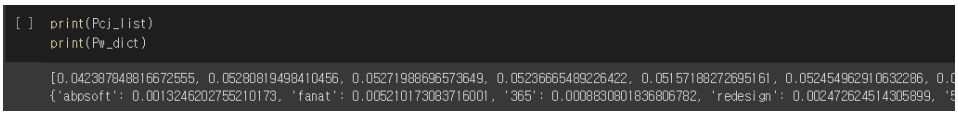


Figure 5: A part of the computed result of $P(c_j)$ and $P(w)$.

After that, the left values can be computed as follows:

Algorithm 10 To compute the entropy of $P(c_j|w)$ and $P(c_j|\bar{w})$

```
Pcjw_list = [], Pcjnotw_list = [], Ecjw_dict = {}, Ecjnotw_dict = {}, useful_word = 0, cn = 0
for word in df_dict do
    if (total_num - df_dict[word]) != 0 then
        Check every document in clr_cate_dict[category] if word is in document
        if word in document then
            cn += 1
        else
            pass
        end if
        if cn != 0 then
            P(c_j|w) = cn / df_dict[word]
            if cn != len(clr_cate_dict[category]) then
                P(c_j|w̄) = (len(clr_cate_dict[category]) - cn) / (total_num - df_dict[word])
                Append P(c_j|w) and P(c_j|w̄) to Pcjw_list[] and Pcjnotw_list[]
            else
                pass
            end if
            cn = 0
        else
            pass
        end if
        useful_word += 1
    else
        pass
    end if
    if useful_word == 1 then
        Using get_entropy_sum(), compute the entropy of P(c_j|w) and P(c_j|w̄)
        Ecjw_dict{key: word, value: Ecjw}, Ecjnotw_dict{key: word, value: Ecjnotw}
        useful_word = 0
    else
        pass
    end if
    Pcjw_list = [], Pcjnotw_list = []
end for
```

Algorithm 10 looks complex, but most *if statements* are used to avoid any possible error. Thus, it simply check if the word is in the documents in a category and count how many documents in a category include the word. Based on this counted number, the following equations can be computed:

$$P(c_j|w) = \frac{P(c_j \cap w)}{P(w)} \quad \& \quad P(c_j|\bar{w}) = \frac{P(c_j \cap \bar{w})}{P(\bar{w})}) \tag{4}$$

The Fig. 6 and Fig. 7 show the computed results of equation 4, respectively. From the row of these figures, it can be known that there are 17,984 words and the column shows the number of categories which include these words.

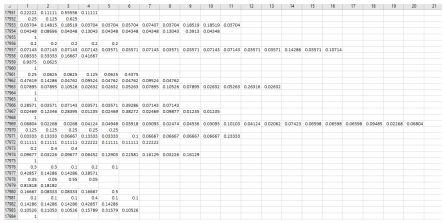


Figure 6: A part of the computed result of $P(c_j|w)$.

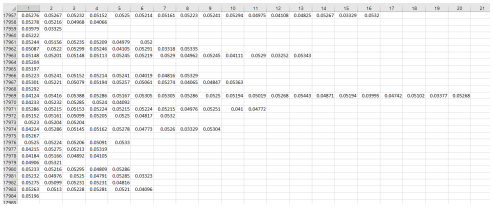


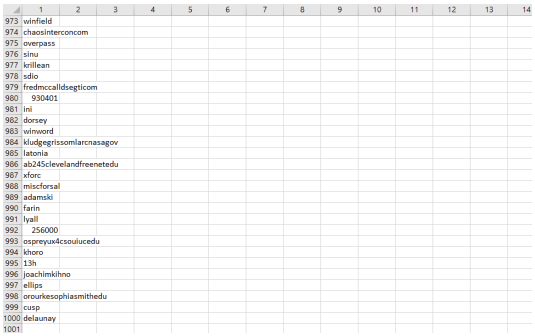
Figure 7: A part of the computed result of $P(c_j|\bar{w})$.

Then, the entropy of each word can be successfully computed and the IG value can be also calculated based on equation 1. Assume that the computed IG values are stored in *IG_dict* as {key: *word*, value: IG value}. Then, the 1000 words with maximum IG values can be selected as follows:

Algorithm 11 To choose 1000 words with maximum IG values

```
max_IG_dict = , def_max_IG_num = 1000, max_IG_words = []
for defined value of def_max_IG_num do
    fin_max = max(word of IG_dict)
    new_val = IG_dict[fin_max]
    Append fin_max → max_IG_words[]
    max_IG_dict{key: fin_max, value: new_val}
end for
```

Here, since the *def_max_IG_num* is defined to 1000, the *for loop* will operate 1000 times, and each loop will pick a word with the highest IG value. Finally, the 1000 words can be chosen and stored in max_IG_words and max_IG_dict. To use the list of these words for testing dataset, max_IG_words can be saved into a CSV.file to encode this data again. Then, the 1000 words will be stored as follows:



	1
973	winfield
974	chaosinterconcom
975	overpass
976	sinu
977	krillean
978	sdio
979	fredmccalldsegticom
980	930401
981	ini
982	dorsey
983	wlmword
984	kludgegrissomlarcnasagov
985	latonia
986	ab245clevelandfreenetedu
987	xforc
988	miscforsal
989	adamski
990	farin
991	lyall
992	256000
993	espreyux4csouiucedu
994	khoro
995	13h
996	joachimkihno
997	ellips
998	orourkesophiasmithedu
999	cusp
1000	delaunay
1001	

Figure 8: A part of the 1000 words with the maximum IG values of *max_IG_list*.

By using this list, the TF-IDF of the 1000 words can be computed.

In Algorithm 12, the self-defined function **clr_docus_list**() will be firstly used to convert the dictionary dataset into a nested list. Then, we can check the 1000 words in each document and compute a_{ik}. And then, every a_{ik} in each document will be normalized to A_{ik}. Then, these values will be stored in the final result, and the final result will show a $N \times 1000$ matrix of A_{ik} when N denotes the total number of documents in the training dataset. In the same way, the TF-IDF of the test dataset is also computed based on the 1000 selected words and described as a matrix, $B_{M\times 1000}$.

Algorithm 12 To compute TF-IDF of the 1000 words with maximum IG values

clr_data = **clr_docus_list**(clr_cate_dict)
clr_list = clr_data[0] → A nested list of total documents
aik_list= [], a_in_doc = [], norm_aik = 0, norm_aik_doc = []
for *document* in clr_list **do**
    for *word* in max_IG_dict **do**
        fik = *document*.count(word) / len(*document*)
        $a_{ik} = f_{ik} * log(len(clr\_list/df\_dict[word]))$
        norm_aik += ${a_{ik}}^2$
        Append a_{ik} → a_in_doc[]
    end for
    for a_{ik} in a_in_doc **do**
        $A_{ik} = \frac{a_{ik}}{\sqrt{\sum_{j=1}^{1000} a_{ij}^2}}$
        Append A_{ik} → norm_aik_doc[]
    end for
    Append norm_aik_doc → aik_list[]
    Initialize a_in_doc[] and norm_aik_doc[]
end for

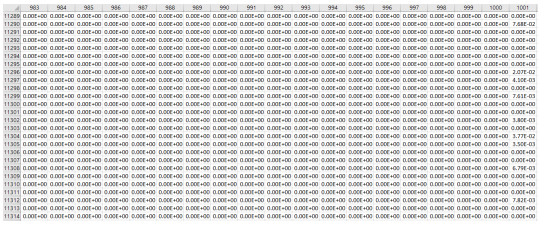


Figure 9: A part of the computed result of TF-IDF in training dataset.

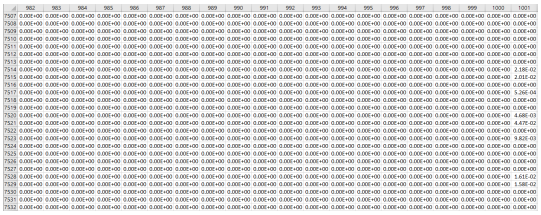


Figure 10: A part of the computed result of TF-IDF in test dataset.

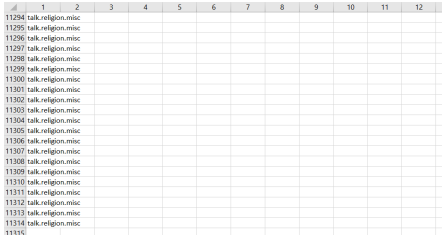


Figure 11: Categories of documents in a matrix $A_{N\times1000}$.

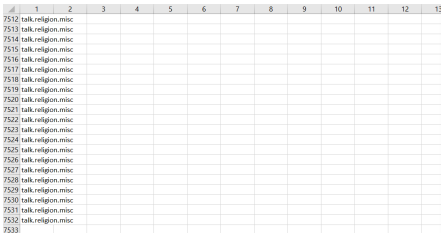


Figure 12: Categories of documents in a matrix $B_{M\times1000}$.

In Fig. 9 and Fig. 10, the column 1001 shows the sum of each row that is equal to the sum of the whole TF-IDF in a document. Fig. 11 and Fig. 12 shows the category of each document in the matrix $A_{N\times1000}$ and $B_{M\times1000}$, respectively. These two data can be useful when we use the text classification algorithm, such as Naive Bayes Classifier, based on these TF-IDF values.

4 ANALYSIS

From the designed algorithms in Section 3, all the required tasks in this experiment were completed successfully. However, there are several problems discovered while doing a specific analysis of the results. Firstly, during the TF-IDF computation of the test dataset, the result shows that there are some selected words from the training dataset are not existed in the test dataset. Fig. 13 is some

examples of the non-existed words in the test dataset.

```
Word does not exist: jodfishesilverucsindianaedu, Word does not exist: chaosinterconcom, Word does not exist: sdio,
Word does not exist: 930401, Word does not exist: ini, Word does not exist: winword, Word does not exist: miscforsal,
Word does not exist: adamski, Word does not exist: farin, Word does not exist: lyall, Word does not exist: 256000,
Word does not exist: ospreyux4csouiucedu, Word does not exist: 13h, Word does not exist: joachimkihno,
Word does not exist: orourkesophiasmithedu, Word does not exist: cusp, Word does not exist: delaunay

Process finished with exit code 0
```

Figure 13: Sample of the non-existed words from max_IG_words in the test dataset.

From this sample, it can be found that the 1000 words with maximum IG values selected from the training dataset include a large number of non-informative words, such as 'email address', 'numbers' and not accurately normalized words in the data preprocessing.

47	mwunixmitreorg
48	mithraist
49	basho
50	issckvmbyuedu
51	caldwellfacmanohsuedu
52	c5muiwaqcmailerccfsuedu
53	keithccocaltechedu
54	crameroptilinkcom
55	stevehthoriscbrcom
56	uunetpyramidoptilinkcram
57	mozumd
58	jaeger
59	jaegerbuphybuedu
60	clintongor
61	kaldisromulusrutgersedu
62	beauchain
63	bobbeviceicotekcom
64	lifeaimitedu
65	philnetcomcom
66	ronzon

Figure 14: Sample of the non-existed words from max_IG_words in the test dataset.

After the inspection of the 1000 words from the training dataset, it is clear that almost half of the data is an email address. Also, many preprocessed texts are not similar to its original format. Therefore, these errors will influence the feature selection. Especially, there are many repeated email addresses in the dataset, and the preprocessed format of email addresses are always same, but the words will have different formats depends on its use in the context.
As a result, in Fig. 9 and Fig. 10, there are many documents which have no TF-IDF values. This problem can decrease the accuracy of text classification, especially when the dataset is significantly large. According to the preprocessed dataset, the possible solution of this issue could be using lemmatization instead of stemming. As already mentioned in the introduction, stemming is a simple method of text normalization by removing the suffixes of words.
Therefore, it could have inaccurate results of normalization. Hence, in the feature selection, it can keep ignoring the repeated informative words due to the wrong normalized format, instead choosing the uniform texts, such as email address and numbers. To check the performance of lemmatization, the data preprocessing was conducted again with exactly same methods, but changing stemming to lemmatization. Then, the preprocessed data can be obtained as follows:

```
'system', 'approximately', '130', 'price', '75', 'system', 'purchase', 'january', 'see', 'little', 'contact', 'via', 'method', 'signature', 'file', 'chris', 'freemesser', 'rit',
'cpet', 'visit', 'rochester', 'york', 'money', '1', 'priority', 'home', 'world', 'large', 'clf3678ultbiscritedu', 'cl', 'freemesser', 'subject', 'turbografx16', 'system', 'sale',
'distribution', 'na', 'organization', 'rochester', 'institute', 'technology', 'line', '24', 'nntppostinghost', 'ultbgwiscritedu', 'sale', 'nec', 'turbografx16', 'video', 'game',
'system', 'package', 'include', 'base', 'unit', 'power', 'supply', 'tv', 'hookup', 'controller', 'game', 'keith', 'courage', 'alpha', 'zone', 'ordyne', 'extra', 'controller',
'turbotap', 'hook', '5', 'controller', 'tg16', 'original', 'package', 'manual', 'etc', 'item', 'game', 'tv', 'sport', 'football', 'alien', 'crush', 'splatterhouse', 'takin',
'hoop', 'current', 'market', 'price', 'system', 'approximately', '130', 'price', '75', 'system', 'purchase', 'january', 'see', 'little', 'contact', 'via', 'method', 'signature',
'file', 'chris', 'freemesser', 'rit', 'cpet', 'visit', 'rochester', 'york', 'money', '1', 'priority', 'home', 'world', 'large', 'usenet', 'clf3678ultbritedu', 'collection',
'pothole', 'clf3678ultbiscritedu', 'cl', 'freemesser', 'subject', 'turbografx16', 'system', 'sale', 'distribution', 'na', 'organization', 'rochester', 'institute', 'technology',
'line', '24', 'nntppostinghost', 'ultbgwiscritedu', 'sale', 'nec', 'turbografx16', 'video', 'game', 'system', 'package', 'include', 'base', 'unit', 'power', 'supply', 'tv',
'hookup', 'controller', 'game', 'keith', 'courage', 'alpha', 'zone', 'ordyne', 'extra', 'controller', 'turbotap', 'hook', '5', 'controller', 'tg16', 'original', 'package',
'manual', 'etc', 'item', 'game', 'tv', 'sport', 'football', 'alien', 'crush', 'splatterhouse', 'takin', 'hoop', 'current', 'market', 'price', 'system', 'approximately', '130',
'price', '75', 'system', 'purchase', 'january', 'see', 'little', 'contact', 'via', 'method', 'signature', 'file', 'chris', 'freemesser', 'rit', 'cpet', 'visit', 'rochester',
'york', 'money', '1', 'priority', 'home', 'world', 'large', 'usenet', 'clf3678ultbritedu', 'collection', 'pothole', 'genie', 'cfreemesser', 'clf3678ultbiscritedu', 'cl',
'freemesser', 'subject', 'turbografx16', 'system', 'sale', 'distribution', 'na', 'organization', 'rochester', 'institute', 'technology', 'line', '24', 'nntppostinghost',
'ultbgwiscritedu', 'sale', 'nec', 'turbografx16', 'video', 'game', 'system', 'package', 'include', 'base', 'unit', 'power', 'supply', 'tv', 'hookup', 'controller', 'game', 'keith',
 'courage', 'alpha', 'zone', 'ordyne', 'extra', 'controller', 'turbotap', 'hook', '5', 'controller', 'tg16', 'original', 'package', 'manual', 'etc', 'item', 'game', 'tv', 'sport',
'football', 'alien', 'crush', 'splatterhouse', 'takin', 'hoop', 'current', 'market', 'price', 'system', 'approximately', '130', 'price', '75', 'system', 'purchase', 'january',
'see', 'little', 'contact', 'via', 'method', 'signature', 'file', 'chris', 'freemesser', 'rit', 'cpet', 'visit', 'rochester', 'york', 'money', '1', 'priority', 'home', 'world',
'large', 'usenet', 'clf3678ultbritedu', 'collection', 'pothole', 'genie', 'cfreemesser']]
```

Figure 15: A part of the result of data preprocessing with lemmatization.

Fig. 15 clearly shows the accurate performance of lemmatization by normalizing every word to its root. However, the problem of this method is the low efficiency. Compared to stemming, the running time of this method is significantly longer. Thus, the text normalization method can be decided depends on the classification accuracy of stemming (Shubham, 2019).
Additionally, the designed algorithms in this experiment has low efficiency due to the nested for loops and if statements. Therefore, in the next tasks, the use of nested for loop needs to be decreased and write simpler codes to increase the efficiency of the program.

5 CONCLUSION

In this experiment, the given dataset **20news-bydate.tar.gz** was used to apply the general process of Natural Language Processing for text classification. In NLP, data preprocessing is critically important to convert documents into texts which can be easily encoded in a machine.
Therefore, the defined algorithms removed the non-alphabet characters from the dataset and converted all data into lowercase. And then, every stopwords are removed because these texts are not informative. Based on this dataset, the stemming process was conducted for text normalization. The obtained results from this step show the preprocessed words as required. However, the text normalization was not accurate enough due to the property of stemming.
The next step of text categorization is the feature selection. In this experiment, the filter approach was used to select discriminative features based on the statistical analysis, using Information Gain. Since the dataset in this test is large, removing the low-frequency words could be helpful to improve efficiency and accuracy. Therefore, the words have document frequency lower than 5 are all removed by the self-defined functions. Then, only the information gain of the left words needs to be computed.
The computed result of each probability is saved into a CSV.file to check. Based on these probabilities, the designed algorithm calculated the entropy, and finally the information gain of each word.
The information gain will describe which words are highly informative for text classification. Thus, the designed algorithm selected 1000 words that have maximum IG values and stored these words into a CSV.file.
As a result, only the TF-IDF values of these 1000 words in the dataset should be computed to know the highly informative words for each document. Then, after normalizing the TF-IDF value of each word, the final output will be a matrix, $A_{N\times 1000}$ for the training dataset. By encoding the CSV.file that stores the 1000 selected words, the final output of the test dataset can be obtained as well as a matrix, $B_{M\times 1000}$.
However, the analysis in Section 4 explains that the inaccurate result of text normalization based on stemming has caused the low performance of the feature selection. Thus, the selected 1000 words include many non-informative data such as emails and numbers. As a result, the final result of test dataset shows that the total sum of TF-IDF values in many documents are zero.
As a solution of this issue, this report suggested to use lemmatization for text normalization. However, this method will highly increase the running time and memory usage. Thus, it requires a high performance machine to process.

(Note that the submission file will only contain the CSV.files of the outputs which have small size because some files are not able to submit due to its size.)

REFERENCES

Andreas Chandra. Feature selection in text classification, 2018. [Online: accessed 25-March-2022].

Parlad Neupane. Understanding text classification in nlp with movie review example, 2020. [Online: accessed 25-March-2022].

Singh Shubham. Nlp essentials: Removing stopwords and performing text normalization using nltk and spacy in python, 2019. [Online: accessed 25-March-2022].