TASK 3 - Development History

Regarding to divide our workload, each group member will incharge each task in this assignment, and if there any challenge happens, one will support another. Here is the information about student main incharge:

1. Task 1: Anh Huy Phung
2. Task 2: Wei Yu Su

Task 2

Date: 05/04/2023

Today, we went through the assignment specification, and started exploring task 2 data.

Contribution:

Group member 1:

- Process concatenating all the data in the Excel sheets, and remove duplicated records and null columns. Extract the final DataFrame with 2 columns **'ChannelID'** and **'textOriginal'**.

Group member 2:

- Remove all the special characters and filter all English comments only, create a summary for each ChannelID that has at least 15 English comments.

proof

```
def regexp_tokenizer(text):
    """
    Tokenize text using regular expressions.

    Parameters:
        text (str): The input text.

    Returns:
        list: List of tokens.
    """
    tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
    unigram_tokens = tokenizer.tokenize(text)
    return unigram_tokens

def tokens_in_channelID_df(df):
    """
    Aggregate words by ChannelID.

    Parameters:
        df (DataFrame): Input DataFrame containing text data.

    Returns:
        DataFrame: DataFrame with word lists aggregated by ChannelID.
    """
    df = df.copy()
    df['word_list'] = df['textOriginal'].apply(lambda x: regexp_tokenizer(x))
    tokenlist_by_channelID_df = df.groupby('ChannelID')['word_list'].agg(lambda x: sum(x, []))
    return tokenlist_by_channelID_df.reset_index()
```

```
def channel_summary(df):
    """
    Generates a summary DataFrame containing aggregated information about channels.

    Args:
    - df: DataFrame containing channel data.

    Returns:
    - DataFrame with channel summary information including total comments and English comments count.
    """
    # Create a copy of the DataFrame to avoid modifying the original
    df_copy = df.copy()

    # Replace null values in 'textOriginal' with 1
    df_copy['textOriginal'].fillna(1, inplace=True)

    # Perform aggregation
    summary_df = df_copy.groupby('ChannelID').agg(
        total_comments=('textOriginal', 'count'),  # Count non-null values
        en_comments=('is_english', lambda x: (x == True).sum())
    ).reset_index()

    # Rename columns
    summary_df.columns = ['channel_id', 'all_comment_count', 'eng_comment_count']

    return summary_df


def english_df(summary_df, df):
    """
    Filters DataFrame to include only English comments from channels with at least 15 English comments.

    Args:
    - summary_df: Summary DataFrame containing channel information.
    - df: Original DataFrame containing comments.

    Returns:
    - Filtered DataFrame containing English comments from selected channels.
    """
    # Filter channels with at least 15 English comments
    filter_id = summary_df['channel_id'][summary_df['eng_comment_count'] >= 15]

    # Filter DataFrame to include only selected channels and English comments
    filtered_channels = df[df['ChannelID'].isin(filter_id)]
    filtered_channels = filtered_channels[filtered_channels['is_english'] == True]

    return filtered_channels[['ChannelID', 'textOriginal']]
```

Step 2: Text extraction and cleaning

```
#str to json and catch target message
def json_txt_catch(dataframe, idx):
    log=[]
    try:
        snippet_txt = total_df.iloc[idx]['snippet']
        final_dictionary = eval(snippet_txt)
        textOriginal = [final_dictionary['channelId'], final_dictionary['topLevelComment'][
    except x:
        log.append(str(x))
    finally:
        return textOriginal
```

```
def load_emoji_txt(filename):
    emojiwords = []
    try:
        with open(filename, encoding="utf-8") as f:
            emojiwords = f.read().splitlines()
    except:
        print('The file is not found')
    finally:
        return emojiwords
```

```
#remove the emojis
def remove_emoji(string, emojiwords):
    tem_txt = ''
    try:
        emoji_pattern = re.compile('|'.join(re.escape(emoji) for emoji in emojiwords))
        tem_txt = emoji_pattern.sub(r'', string)
    except:
        print('Something went wrong when removing emoji')
    finally:
        return tem_txt
```

Date 13/04/2024

Today, we started deciding to allocate steps for processing tokens and came with this flow:

1. Tokenization
2. Bigram
3. Remove stopwords
4. Remove rare tokens
5. Remove less than 3 words tokens
6. Stem

Contribution:

Group member 1:

- Develop a function to tokenize all the English comments by comment level as well as the bigram function to extract bigram functions from these..

Group member 2:

- Develop other functions (remove stopwords, rare tokens and less than 3 words tokens).

After this, we also plan to do other functions for extracting vocabulary and sparse presentation.

Proof

```
def regexp_tokenizer(text):
    """
    Tokenize text using regular expressions.

    Parameters:
        text (str): The input text.

    Returns:
        list: List of tokens.
    """
    tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
    unigram_tokens = tokenizer.tokenize(text)
    return unigram_tokens

def tokens_in_channelID_df(df):
    """
    Aggregate words by ChannelID.

    Parameters:
        df (DataFrame): Input DataFrame containing text data.

    Returns:
        DataFrame: DataFrame with word lists aggregated by ChannelID.
    """
    df = df.copy()
    df['word_list'] = df['textOriginal'].apply(lambda x: regexp_tokenizer(x))
    tokenlist_by_channelID_df = df.groupby('ChannelID')['word_list'].agg(lambda x: sum(x, []))
    return tokenlist_by_channelID_df.reset_index()
```

```
def load_stopwords_txt(filename):
    stopwords = []
    try:
        with open(filename) as f:
            stopwords = f.read().splitlines()
    except:
        print('The file is not found')
    finally:
        return stopwords
```

```
def remove_indep_stopwords(string, stopwords):
    tem_txt = ''
    try:
        #sw_pattern = re.compile('|'.join(re.escape(stp) for stp in stopwords))
        pattern = r"\b(?:" + "|".join(stopwords) + r")\b"
        regex = re.compile(pattern, re.IGNORECASE)
        tem_txt = regex.sub(r'', string)
        #tem_txt = sw_pattern.sub(r'', string)
    except e:
        print('Something went wrong when removing stopwords' + str(e))
    finally:
        return tem_txt
```

```
def remove_indi_stop_words(tokens, stopwords, is_uni):
    stopwords_set = set(stopwords)
    if is_uni:
        stopped_tokens = [w for w in tokens if w not in stopwords_set]
    else:
        stopped_tokens = [w for w in tokens if w.split('_')[0] not in stopwords_set and w.sp
    return stopped_tokens
```

```
def remove_dep_stopwords(df, threshold):
    # Step 1: Calculate the number of unique ChannelIDs each word appears in
    word_id_counts = df.explode('word_list').groupby('word_list')['channel_id'].nunique()

    # Step 2: Calculate the threshold count
    ch_id_set = set(df['channel_id'])
    threshold_count = threshold * len(ch_id_set)

    # Step 3: Set the threshold
    context_dependent_stopwords = word_id_counts[word_id_counts >= threshold_count].index.t

    return context_dependent_stopwords
```

```
def remove_dep_stopwords_list(df, list, threshold):
    # Step 1: Calculate the number of unique ChannelIDs each word appears in
    word_id_counts = df.explode('word_list').groupby('word_list')['channel_id'].nunique()

    # Step 2: Calculate the threshold count
    ch_id_set = set(df['channel_id'])
    threshold_count = threshold * len(ch_id_set)

    # Step 3: Set the threshold
    temp_list = word_id_counts[word_id_counts >= threshold_count].index.tolist()
    context_dependent_stopwords = [w for w in list if w not in temp_list]
    return context_dependent_stopwords
```

```
# Removing the Less Frequent Words
def remove_less_freq_words(df, threshold):
    # Step 1: Calculate the number of unique ChannelIDs each word appears in
    word_id_counts = df.explode('word_list').groupby('word_list')['channel_id'].nunique()

    # Step 2: Calculate the threshold count
    ch_id_set = set(df['channel_id'])
    threshold_count = threshold * len(ch_id_set)

    # Step 3: Set the threshold
    rare_tokens = word_id_counts[word_id_counts >= threshold_count].index.tolist()

    return rare_tokens
```

```
# Removing the Tokens with a length less than 3
def remove_less_len(word_list, length):
    tem_list = [token for token in word_list if len(token) >= length]
    return tem_list
```

Date 15/04/2024

Today, we reviewed process, step 4 processing flow offline.

Contribution:

Group member 1:

- Program testing.

Group member 2:

- Check that the program results are consistent with the excel data.

Proof

```
[ ] import matplotlib.pyplot as plt
    def element_check(input1, input2):
        return [i for i in input1 if i not in input2]

    for element in test_list:
        combination, name, input1, input2 = element[-1], element[0], element[1], el
        check, len_check = element_check(input1, input2), len(element_check(input1, in
        combination = ' & '.join(a.__name__ for a in combination)
        print(f'Combination function: {combination} \n',\
                f'Bigram extracting place: {name}, \n', \
                f'final bigram: {len(input2)}, \n', \
                f'remove bigram: {len(element_check(input1, input2))} \n',\
                f'total vocab: {len(total_vocab_sorted)}')

    import pickle
```

```
Combination function: rare_tokens_set & less_three_tokens_remove & eliminate_or_choose_
 Bigram extracting place: tokens_in_channelID_df,
 final bigram: 194,
 remove bigram: 6
 total vocab: 2991
Combination function: rare_tokens_set & less_three_tokens_remove & eliminate_or_choose_
 Bigram extracting place: eliminate_or_choose_words,
 final bigram: 198,
```

	D	E
1	channel_id	comment
2760	UC-3sBKh8YYbG2KyVHnSyA1A	shame on you disney,, trying to brainwash children,,hope you go under.
2761	UC-3sBKh8YYbG2KyVHnSyA1A	doesn't sound like the original disney signers of the songs
2762	UC-3sBKh8YYbG2KyVHnSyA1A	https://youtube.com/shorts/widqnuci_1g?si=mtrghxbrdff-5iox
2763	UC-3sBKh8YYbG2KyVHnSyA1A	once upon a december
6204	UC-3sBKh8YYbG2KyVHnSyA1A	thank you,disney
9211	UC-3sBKh8YYbG2KyVHnSyA1A	loyal brave true
18391	UC-3sBKh8YYbG2KyVHnSyA1A	as melhores m??sicas e filmes de todos os tempos j?? n??o fazem filmes c
18392	UC-3sBKh8YYbG2KyVHnSyA1A	???
18393	UC-3sBKh8YYbG2KyVHnSyA1A	i am a transgender female and i get a lot of hate so disney is my stress rel
27305	UC-3sBKh8YYbG2KyVHnSyA1A	this brings back memories when i was little and i wouldn't want to let it g
32988	UC-3sBKh8YYbG2KyVHnSyA1A	i don t like it
35943	UC-3sBKh8YYbG2KyVHnSyA1A	when i was younger i went to disney a lot with my family but we stop goir
35944	UC-3sBKh8YYbG2KyVHnSyA1A	not only children needs lullaby, sometimes adult need it too for reminisci
41439	UC-3sBKh8YYbG2KyVHnSyA1A	disney was a nazi sympathizer now disney is a pedophile sympathetic com
43982	UC-3sBKh8YYbG2KyVHnSyA1A	aqu?? los que reviven su infancia escuchando esto
43983	UC-3sBKh8YYbG2KyVHnSyA1A	how far i'll go
46662	UC-3sBKh8YYbG2KyVHnSyA1A	i like the originals. all these knock offs wanna bes are not as good.
46663	UC-3sBKh8YYbG2KyVHnSyA1A	
46664	UC-3sBKh8YYbG2KyVHnSyA1A	eyy yo . disney medley is fire... im about to take this. no1 copyright me
49321	UC-3sBKh8YYbG2KyVHnSyA1A	my favorite is the little mermaid
49322	UC-3sBKh8YYbG2KyVHnSyA1A	i'm in love with this
51886	UC-3sBKh8YYbG2KyVHnSyA1A	disney my favorite songs about princess
54448	UC-3sBKh8YYbG2KyVHnSyA1A	hello disney it's robby orginse disney intelligence to over ride selena gome
61924	UC-3sBKh8YYbG2KyVHnSyA1A	alsd lo
64525	UC-3sBKh8YYbG2KyVHnSyA1A	i see the light
69418	UC-3sBKh8YYbG2KyVHnSyA1A	i love ???
71782	UC-3sBKh8YYbG2KyVHnSyA1A	i like the way that they different people singing it. and they all have beaut

tem_table

就緒 從 81824中找出 32筆記錄 100%

Date 16/04/2024

Today, we reviewed process, step 4 processing flow offline.

Contribution:

Group member 1:

- Process review, step 4 processing flow.

Group member 2:

- Process review, step 4 processing flow.

Proof

These are pictures to show our explaination to some of funtions and examples.

Testing 2

```
[ ] import pickle
    # Saving the test_list
    with open('test_list.pkl', 'wb') as f:
        pickle.dump(test_list, f)
```

```
[ ] from nltk.probability import *
    from nltk.tokenize import RegexpTokenizer
    from nltk.stem import PorterStemmer

    import multiprocessing as mp
    from itertools import permutations

    stop_work = pd.read_csv('/content/drive/Shareddrives/FIT5196_S1_2024/A1/stopwords_en.t
    stop_work_list = stop_work.values.flatten().tolist()

    functions = [rare_tokens_set, less_three_tokens_remove, eliminate_or_choose_words]
    combinations = list(permutations(functions, 3))
    combinations_as_lists = [list(combination) for combination in combinations]


    functions_test = [tokens_in_channelID_df]
```

```
# Step4-a
#token_df = tokenization_df(drop_new_df)
#token_df = tokenization_df(only_eg_df)
token_df = tokenization_df(eng_fifteen_df)
unigram_df = token_df.copy()
bigram_df = generate_ngrams_df(token_df, 2)
bigram_txt_df = generate_ngrams_txtpair_df(token_df, 2)
#token_df
#type(token_df)
#DataFrame
#channel_id, word_list
```

```
unigram_agg_df = token_agg_by_chid(unigram_df)
bigram_agg_df = token_agg_by_chid(bigram_txt_df)
```

```
# Step4-d
#Unigram
rare_unigram_df = unigram_agg_df.copy()
rare_unigram_list = remove_less_freq_words(rare_unigram_df, 0.01)
#Bigram
rare_bigram_df = bigram_agg_df.copy()
rare_bigram_list = remove_less_freq_words(rare_bigram_df, 0.01)
#list
```

```
n = 0
for i in range(len(rare_bigram_list)):
    if len(rare_bigram_list[i].split('_')) < 2:
        print(rare_bigram_list[i])
    else:
        n = n + 1
```

```
# Step4-e
less_uni_token_list = remove_less_len(rare_unigram_list, 3)
less_bi_token_list = remove_less_bi_len(rare_bigram_list, 3)
```

```
# Step4-b
# context-independent
#Unigram
less_uni_token_list_cp = less_uni_token_list.copy()
stopped_uni_list = remove_dep_stopwords_list(rare_unigram_df, less_uni_token_list_cp, 0.99)
#Bigram
less_bi_token_list_cp = less_bi_token_list.copy()
stopped_bi_list = remove_dep_stopwords_list(rare_bigram_df, less_bi_token_list_cp, 0.99)
#stopped_bi_df
# context-independent
#Unigram
stopped_uni_ind_list = remove_indi_stop_words(stopped_uni_list, stopwords, True)
#Bigram
stopped_bi_ind_list = remove_indi_stop_words(stopped_bi_list, stopwords, False)
```

```
# Step4-c
uni_stem = porter_token_stem(stopped_uni_ind_list)
bi_stem = porter_token_stem(stopped_bi_ind_list)
```

```
# Step4-f
#bi_stem
bi_stem_ele_list = []
for w in bi_stem:
    wd = w.split('_')
    bi_stem_ele_list.append(wd[0])
```

Date 18/04/2024

Today, we started review all of the functions and generated file submission file.

Contribution:

Group member 1:

- Review all of the generated functions (handling tokens and deliver sparse representation), give example in each function for easier documentation.

Group member 2:

- Documentation.

Proof

These are pictures to show our explaination to some of funtions and examples.

```
    Returns:
        A dictionary of ChannelID and its sparse representation pairs.
    """
    # Step 1: Aggregate all tokens by ChannelID level
    tokens_channelID_df = tokens_in_channelID_df(eng_df)

    # Step 2: Tokenize each comment using mwe_tokenizer
    tokens_channelID_df['tokens_list'] = tokens_channelID_df['word_list'].apply(lambda x: mwe_tokenizer.tokenize(x))

    # Step 3: Handle collocations (bigrams) in the comments
    tokens_channelID_df['collocation'] = tokens_channelID_df['tokens_list'].apply(lambda x: bigram_in_comment(list(x)))
    tokens_channelID_df['collocation'] = tokens_channelID_df['collocation'].apply(lambda x: eliminate_or_choose_words(list(x), total_vocab_sorted, False))

    # Step 4: Stem all the unigrams and filter out ones in the unigram vocab only
    tokens_channelID_df['tokens_list'] = tokens_channelID_df['tokens_list'].apply(lambda x: parallel_word_stem(list(x)))
    tokens_channelID_df['tokens_list'] = tokens_channelID_df['tokens_list'].apply(lambda x: eliminate_or_choose_words(list(x), stemmed_unigram_set, False))

    # Step 5: Combine total vocabulary of each comment in a column 'freq_word_list'
    tokens_channelID_df['freq_word_list'] = tokens_channelID_df['tokens_list'] + tokens_channelID_df['collocation']

    # Step 6: Convert the 'freq_word_list' column to frequency distributions
    tokens_channelID_df['freq_word_list'] = tokens_channelID_df['freq_word_list'].apply(lambda x: FreqDist(list(x)))

    # Step 7: Create sparse representation using freq_index_dic and convert_dic functions
    tokens_channelID_df['freq_word_list'] = tokens_channelID_df['freq_word_list'].apply(lambda x: freq_index_dic(x, vocab_index_dic))
    tokens_channelID_df['freq_word_list'] = tokens_channelID_df['freq_word_list'].apply(lambda x: convert_dic(x, False))

    # Step 8: Create a dictionary containing each ChannelID as key and its sparse representation as value
    channel_dic = dict(zip(tokens_channelID_df['ChannelID'], tokens_channelID_df['freq_word_list']))

    return channel_dic
```

```
[ ] # A dictionary of ChannelID and its sparse representation pairs
    final_dic = countvec_output(eng_df, stemmed_unigram_set, vocab_index_dic, mwe_tokenizer, total_vocab_sorted)
```

Output files

```
# Write the formatted summary ChannelID to a text file
summary_df.to_csv('/content/drive/MyDrive/21_channel_list.csv', index=False)

# Write the formatted _countvec to a text file
with open('/content/drive/MyDrive/21_countvec.txt', 'w') as f:
    result_string_linebreak = convert_dic(final_dic, True)
    f.write(result_string_linebreak)

# Write the formatted _vocab to a text file
with open('/content/drive/MyDrive/21_vocab.txt', 'w') as f:
    for key, value in vocab_index_dic.items():
        f.write(f"{key}:{value}\n")
```


Google Colab Workbook Link

Task 2