TARUMT

LANGUAGE PATTERN MINING

WIKIPEDIA DATASET&
SOCIAL MEDIA DATASET

Table Content

1.0 Introduction

In this assignment, we were essentially tasked with discovering and extracting grammatical patterns from text corpus as well as the language pattern. Specifically, we were given two sources of text corpus: social media that consist of mixing languages and Wikipedia in Malay, English, and Mandarin that is already separated well.

1.1 Social Media Task

For social media corpus, we were tasked with discovering the language pattern and Part-Of-Speech structure for sentences. Hence n-gram of languages, tokens and part of speech tagging from bi-gram up to n=5 have to generate and filter those that contain the keywords. Mark for trigram and n=5 gram that have keywords in the middle. After that, showing the highest frequency of occurrence for both patterns and tags respectively. It is required to deal with three languages, Malay, English and Chinese.

1.2 Wikipedia Task

For the Wikipedia corpus, we were tasked with discovering the Part-Of-Speech structure for every sentence. After that, the N-gram has to be generated to form the word sequence and part of the speech sequence up to N=5. The N-gram then filters based on keywords and for trigram and 5 gram, marks those that have keywords in the middle. Lastly, shows the structures with the highest frequency of occurrence. It is required to deal with three language pos-tag, Malay, English and Chinese.

1.3 Contributions

For social media, the purpose of discovering such language patterns and POS structure is to help us understand and identify the most popular language patterns and POS structures used by people to form sentences for daily conversations. In a multilingual country like Malaysia, it is common for Malaysians to mix different languages into a sentence in daily interactions, such as on social media platforms. Hence, when it comes to dissecting and extracting information from such mixed sentences, it can be quite time-consuming to process each word into its corresponding language and POS tag. Some words that originate from dialect are even not able to be tagged. By using the discovered language patterns and POS structures in this assignment, it can be used as a template for processing such mixed sentences, which saves time and allows us to proceed to extracting information from the corpus.

1.3.1 N-gram from text corpus

The generated n-gram in the form of tokens can be helpful in terms of machine learning applications. Usually the token n-gram can be used to create a text autocomplete system by computing the highest probability words from a word. Such methodology includes Statistical (Probabilistic) Language models that aim to assign probabilities to a given sequence of words.

1.3.2 Language Detection

This is an interesting part of this task and it has quite an interesting application. Since the part of speech N-Gram is corresponding to the token N-gram, it can be used to predict what is the language of the next word given the current word's language. It can be used to know under what circumstances, what word they use together usually is multi languages and hence knowing and discovering the lexicon in this special Malaysian culture.

1.3.3 Part-Of-Speech Tagging

The part of speech tagging is useful in various fields including Name Entity Relationship (NER) that help classify the words to its belonging entity which focus on the words that belong to NOUN. The part of speech tagging also helps in shallow parsing or chunking because the basic part of speech is identified and the rest of work is grouping them to higher level combinations such as forming the phrases. The part of speech tagging can also enhance the lemmatization because it can give the lemmatizer about the context of the word in terms of its part of speech in a sentence.

2.0 Data Engineering Pipeline

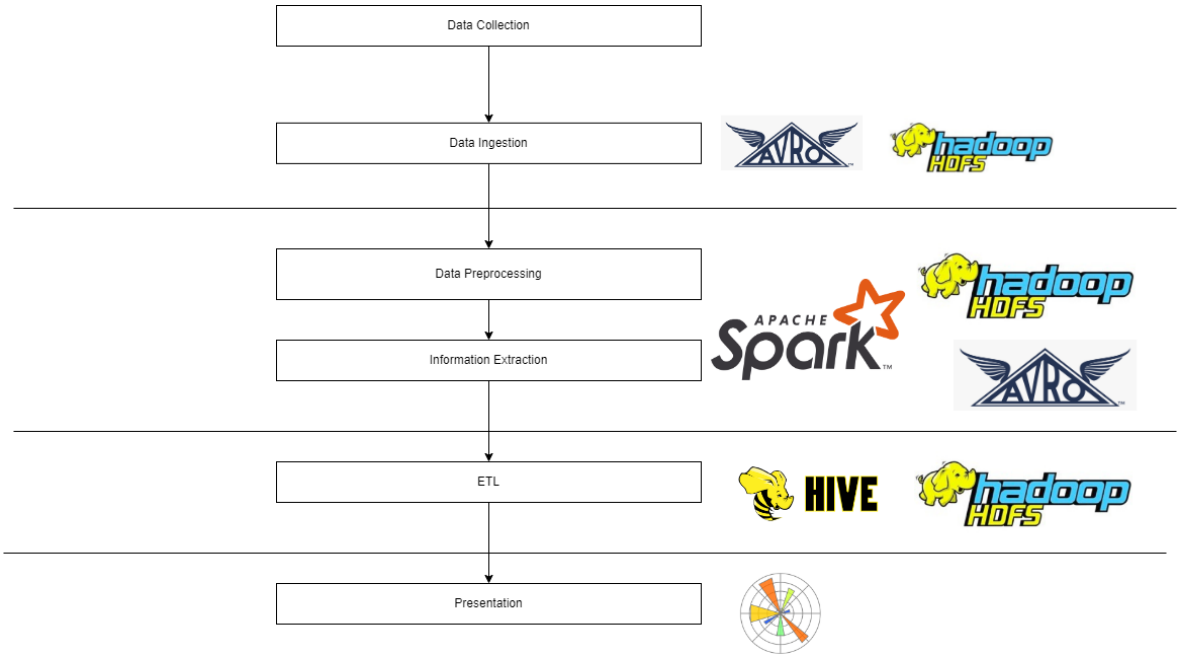


Figure 1 : Data pipeline

2.1 Data collection

The social media data source is from the result of webscarping, which is done by the professor assistant. The data collection of the wikipedia data is by downloading the compressed dump files from the wikipedia website and extracting it which is done by the professor assistant. The resultant file of data collection is in the format of csv for social media and text file for wikipedia.

2.2 Ingestion

The data ingestion process of wikipedia and social media are the same. Initially, in the Jupyter notebook, the csv and text file is converted into the avro file and saved into the local linux file system. This approach has been improved in the combined version of code by changing the storage into the hdfs file system rather than the local linux file system and was tested running fine. The benefit of avro file format is that it stores the file in a smaller size and compressed manner.

2.3 Data preprocessing

The data preprocessing for both social media data and wikipedia data is based on the apache Spark with python api, in short, PySpark. The most used data structure in data preprocessing is the pyspark dataframe and rdd is quite handy to handle some specific task like word counting. The file is also stored temporarily into hdfs in avro format after a specific preprocessing step and deleted after the next step is completed.

2.4 Information Extraction

The information we want to extract from the social media corpus is language pattern, part of speech tagging in the n-gram form from bagram to n=5 n gram which contain the keywords. We use various python libraries to achieve the goal. After that, we also extract the information of the language and tag n-gram frequency for analysis purposes later on. For wikipedia, the information we need is only the part of speech tagging part. The information is loaded into hdfs in avro file format before the etl process.

2.5 Extract, Transform, Load

The avro files with information extracted are then loaded into the hive table, saved using the Optimized Row Columnar (ORC) file for a highly efficient way to store Hive data.

2.6 Data Visualization

The data visualization is just using the python visualization library matplotlib.

2.7 Data Flow Diagram

The data pipeline can be visualized using the data flow diagram (DFD) invented by Larry Constantine.

2.7.1 Data Flow Diagram Social Media

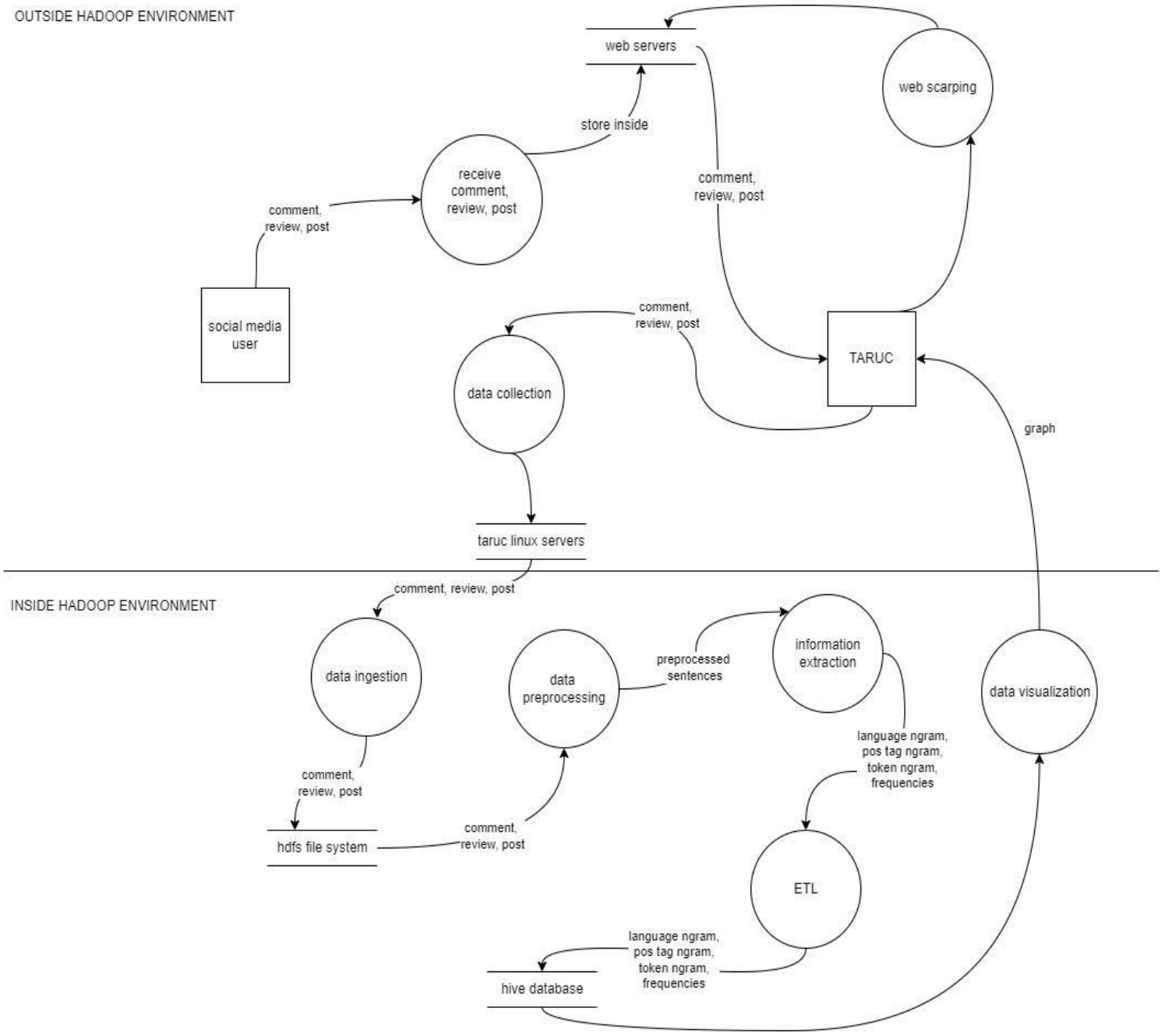


Figure 3 : Social Media DFD

2.7.1 Data Flow Diagram Wikipedia

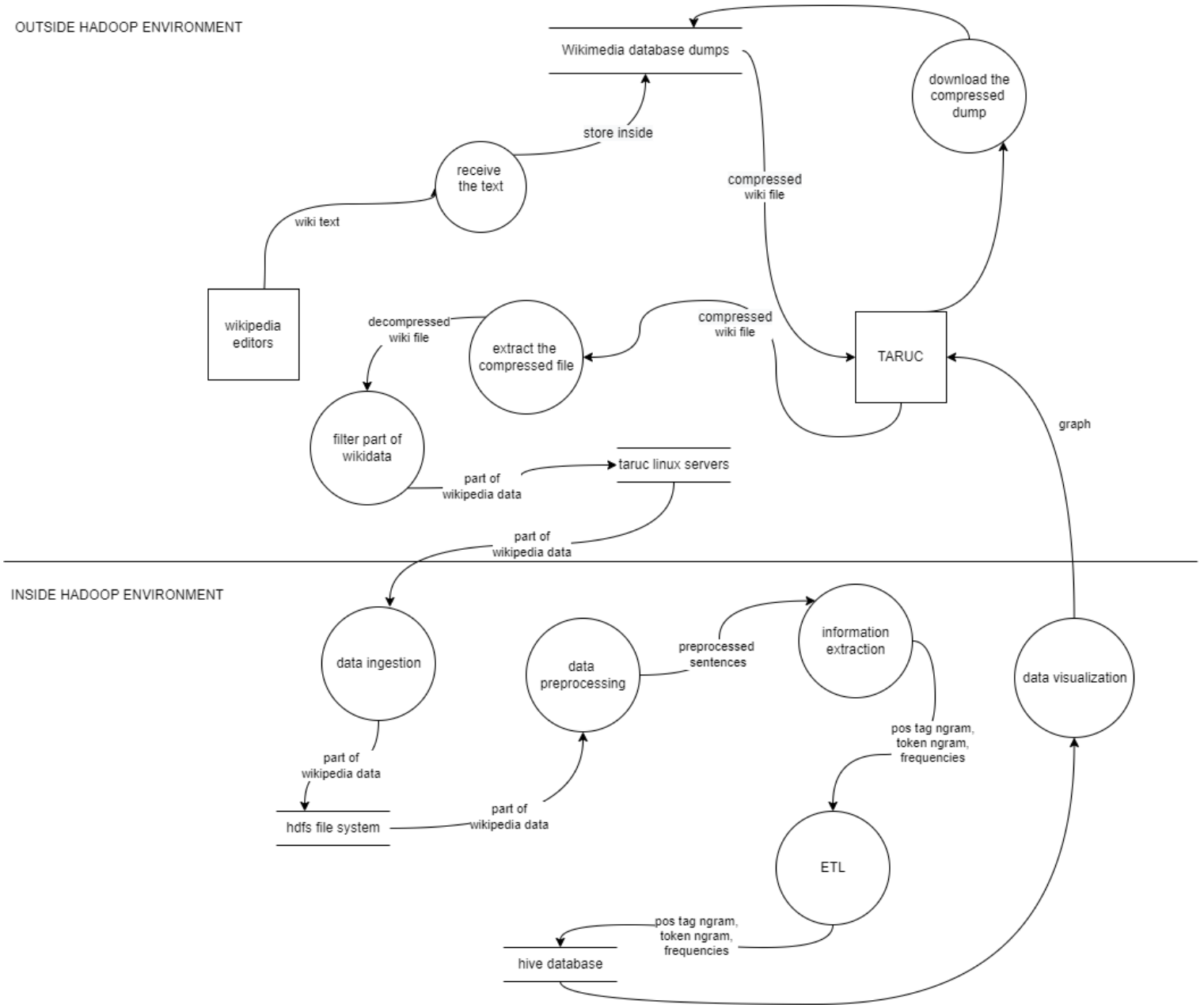


Figure 4 : Wikipedia DFD

3.0 Data Understanding and Pre-Processing

3.1 Data Understanding

There were 2 sources of text corpus collected and used, social media and Wikipedia. It is common knowledge that the data collected is always going to be 'dirty' to some extent, hence data preprocessing must be performed in order to cleanse the data so that the data can provide accurate and better insights.

Social Media dataset

The social media dataset is named as *data-comment-only.csv,* stored in csv format with 1GB of size. Before sentence tokenization, it contains 13,615,018 rows of data. The dataset contains social media of multiple languages such as Chinese, Korean, Japanese, Malay and English. The data also contains many unwanted elements such as emojis, which will be explained in the data preprocessing section (see Figure 5). There is also some news published in this dataset with its length significantly longer than the normal data.

Wikipedia dataset

For Wikipedia, there were 3 kinds of dataset: a 3.6GB *enwiki.txt*, a 0.4 GB *mswiki.txt*, and a 3.0GB *zhwiki.txt*. This data is stored in txt format and all are long paragraphs. Inside the data, it is discovered that the wikipedia has many markdown syntax or styling syntax. It also has some unnecessary sections such as category sections. The detailed understanding will be explained later in the data preprocessing section. The row count for English Wiki is 768,426, Malay Wiki 707,666 and Chinese Wiki 2,495,291. The Malay and English dataset have quite closer row count but much different file size, which means that the average length of each row from the Malay Wiki is quite short. Same thing happens to the Chinese wiki, despite having many rows but having an even smaller file size than the English wiki, indicating that the length is really short in each row.

3.2 Data Preprocessing

3.2.1 Data Preprocessing In Social Media

The diagram below illustrates the data preprocessing steps that are applied to the social media dataset.

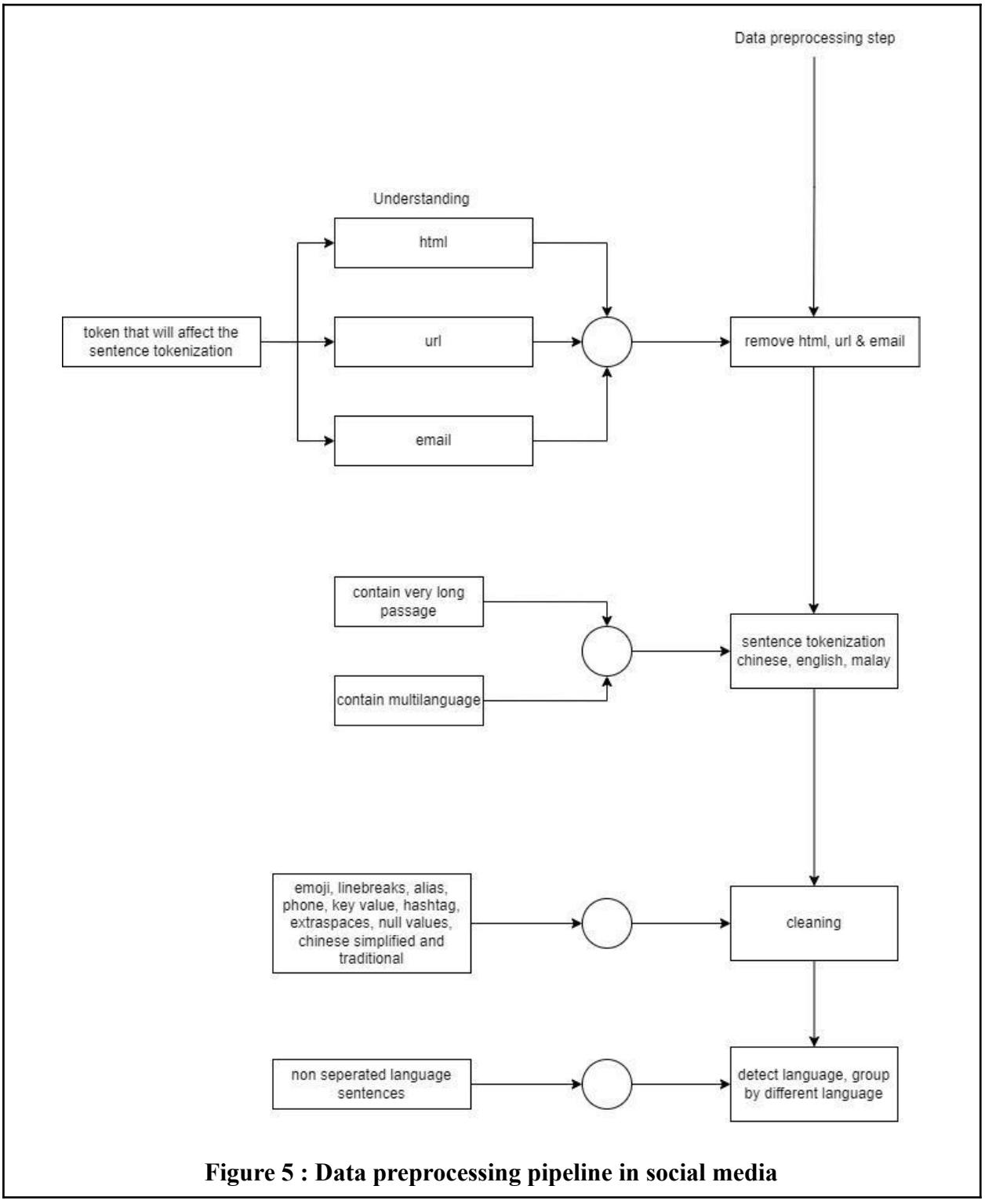


Figure 5 : Data preprocessing pipeline in social media

Step 1 : Remove the element that will affect the sentence tokenization
First step was finding and removing HTML tags, URL links and email addresses which can affect the results of sentence tokenization. The pyspark regex_replace function is used to remove these elements because it is fast.

Regex	**Target**
r'http\S+	url
r'<[^>]+>'	tags
r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'	email
r'@[\w]+'	alias

Step 2 : Sentence tokenization
Since the social media data contains long paragraphs such as news, sentence tokenization is a necessary step to break these long paragraphs into smaller sentences. Firstly, the nltk.sent_tokenize is used to split the english and malay paragraphs into sentences. It will create a column with an array of sentences. To flatten the array of sentences, the explode function from pyspark is applied to the dataframe. After that, in the same dataframe, the HarvestText library is used to tokenize the Chinese sentence.

Step 3 : Data cleaning
Some unwanted elements are identified and attempted to be removed using a different approach.

Target	**Regex**
line breaks and tabs	r"((\r\n\t)|[\n\v\t])+"
phone-number	r"((?:^|(?<=[^\w)]))(((\+?[01])|(\+\d{2}))[.-]?)?(\(?\d{3,4}\)?/?[.-]?)?(\d{3}[.-]?\d{4})(\s?(?:ext\.?|[#x-])\s?\d{2,6})?(?:$|(?=\W)))|\+?\d{4,5}[.-/]\d{6,9}"
Key-value pair	Keep only the value r"(\w+)(?=:):"
hashtag	#([a-zA-Z0-9_]{1,50})
Extra spaces	r'\s+'
numbers	[0-9]
symbols	☺|☻|\\\|\/|♥|\[|♦|♣|\.|♠|•|◘|○|◙|♂|♀|♪|♫|☼|►|◄|↕|‼|¶|\$|§|▬|↨|↑|↓|→|←|∟|↔|▲|▼|#|%|&|,|-|\'|:|;|<|=|>|@|]|_|`|¢|£|¥|Pts|ƒ|ª|º|¿|⌐|¬|½|¼|¡|«|»| ░|▒|▓|||┤|╡|╢|╖|╕|╣|||╗|╝|╜|╛|┐|└|┴|┬|├|─|┼|╞|╟|╚|╔|╩|╦|╠|═| ╬|╧|╨|╤|╥|╙|╘|╒|╓|╫|╪|┘|┌|█|▄|▌|▐|▀|α|ß|Γ|π|Σ|σ|µ|τ|Φ|Ω|δ|∞|φ|ε| ∩|≡|±|≥|≤|⌠|⌡|÷|≈|°|\+|·|·|√|ⁿ|\)|\(|²|■|\~|\.|\?|*|\!|\✓|\^|?|。|\。|\:|;|\ ″

Problems	Solution
Remove contractions	Applied before symbol / punctuation removal Using contractions python library
Simplified chinese and traditional chinese	Standardized into simplified chinese using chinese_converter python library
Emoji & unexpected element	Using the pyspark sentence function.
lowercase	Using pyspark lowercase function
NULL values	Using pyspark filter function

Step 4 : Filter out the sentence that belongs to malay, english and chinese only.
In order to eliminate the sentence that does not belong to Malay, English and Chinese, the lingua library is used to detect the language of each sentence. For multilingual sentences, the lingua library will find the closest distance to determine the majority language group for the sentence to be categorized into. Those sentences detected as “error” or “None” will be filtered out because they most likely do not belong to Malay, English and Chinese. The sentence will have one additional column named language to indicate what language the sentence belongs to.

```
|блюдо выглядит очень красиво и аппетино
|None|
|это не яйцо в пашот
|None|
|прывітанне усе размаўляюцьшто што
|None|
|수치를 느끼게 멘탈을 흔들어놔
|None|
|ステハゲ やないかい
|None|
|گڈ لکی مائی فیورٹ ان کا معیار بہت اچھا ہے
|None|
|우리칠 너무너무고생많이했고 이번활동도 건강하게 마무리했으면 좋겠어요
|None|
```

Figure 6 : Example of sentences detected as “None”

Step 5 : Word Count Statistics.
In the process of selecting keywords, a word count was done to check which words came up most frequently. The word count statistics is required to analyze the frequency of each word so that the size of the sentences extracted can be controlled. The word count is done by following procedure:

1. For each row, tokenize the sentence into words using the jieba tokenizer, and then flatMap the tokenized result into a 1d array-like structure.

2. Map each word to 1 so to become (word, 1) structure
3. Using reduce by key with addition operation by adding the 1 with a group of the same word.
4. The word count statistics is ready.

Step 6: Remove Stopwords & sort the word frequency by descending order

Stopwords are not required during keyword picking, and usually come with exceptionally high frequency, hence removal from the word count dataset is important. The stopwords is removed by following steps (code):

1. Get english and chinese stopwords from nltk.stopwords, merge them into python set.
2. Since nltk don't have malay stopwords set, the stopwords are obtained from this github repository : https://github.com/stopwords-iso/stopwords-ms
3. It is in the form of JSON hence python JSON library is required to extract the stopwords value.
4. Combine these stopwords into one python set.
5. Create a regex for each stopword by iterating and wrapping each stopword with ^ and $.
6. Join the regex list using "|", which means "or" in regex.
7. Now the regex is ready.
8. Using pyspark rlike and negation (~) to explicitly select those words not belonging to stopwords regex.

After the above procedure is finished, we use the pyspark Window function to rank the word based on frequency that is sorted descendingly and come out with the result as below.

word	word_count	id
good	4054219	1
seller	2789294	2
barang	1328427	3
sampai	1287290	4
delivery	1253564	5
fast	1217180	6
tq	1184559	7
thank	973309	8
ok	942106	9
beli	844194	10

Figure 7 : Top 10 words from Social Media Dataset

Step 7: Filter the sentences with keywords only

In social media corpus, the keywords chosen were within the topic of comments given about products in online shopping feedback and are as follow:

```
engKey = "seller|recommend|nice"
chiKey = "好|赞|不错|快|喜欢|谢|值|包装|卖家"
bmKey = "berbaloi|barang|cantik|murah"
```

Step 8: Create sentence length statistics

After filtering those sentences tied to keywords, the sentence length is analyzed by counting the number of tokens after tokenization. The tokenization is by using the jieba library because it can handle both English, Malay and Chinese data (Code). The procedure is as follows:

1. For each sentence, create a tokenize column
2. Using the pyspark "size" function to count the number of tokens as the sentence length.

Step 9: Remove those sentences with less than 5 tokens and too many tokens (outliers).

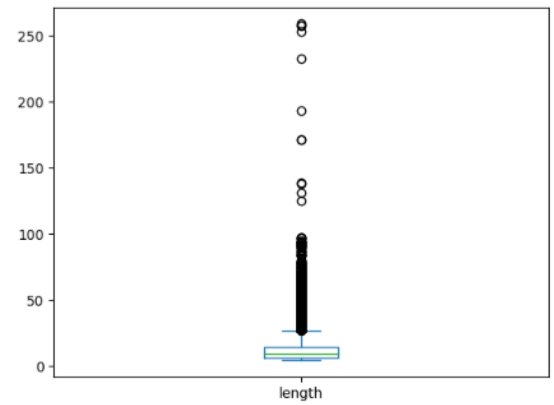
In Jupyter Notebook, the outlier removal step is carried out to remove those words that are considered too high in frequency since we aimed to analyze the dataset with average length. However, there is also a case that the true outliers did exist which contain the important information. Hence, in the combined version of the code, we let users control whether the outlier needed to be removed through the flag "remove_outliers".

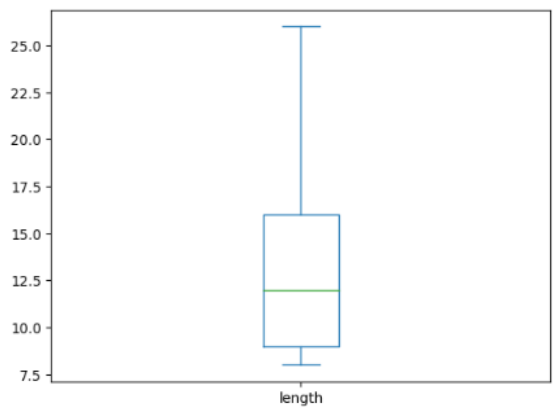
Sample size taken without filtering outliers

Chinese = 131108
Malay = 2702714
English = 1335543

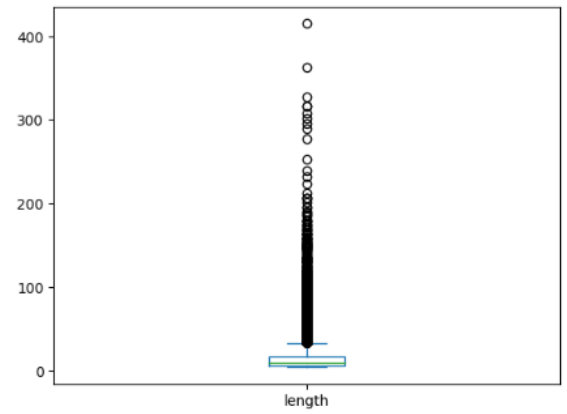
Before and After filtering outliers of in term of length of sentences

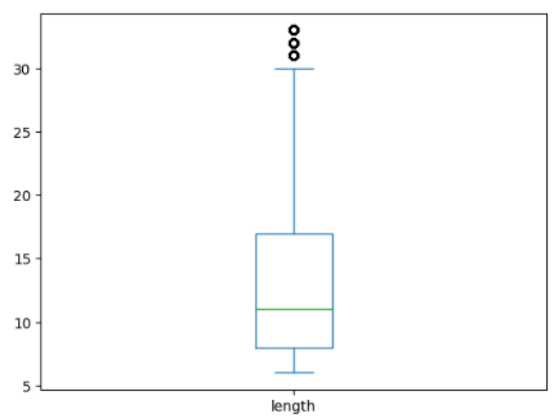
Malay



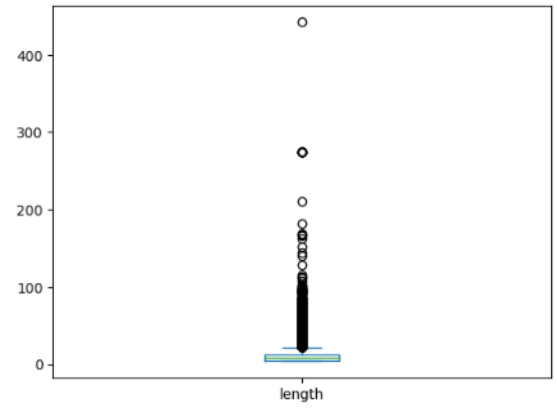


Chinese





English



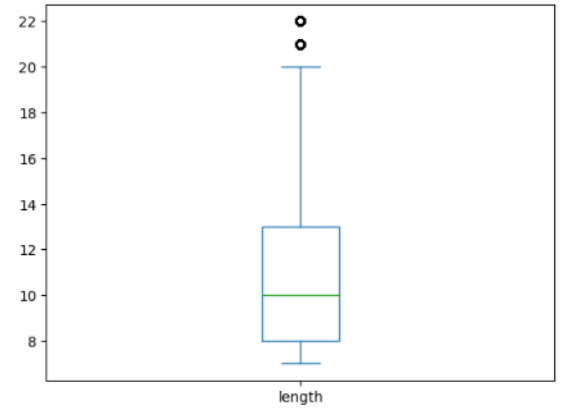


Figure 8 : Comparing result of before and after remove the outliers

Sample size taken after filtering outliers && taking the sample

nobc = 96961
nobm = 651316
nobi = 655400

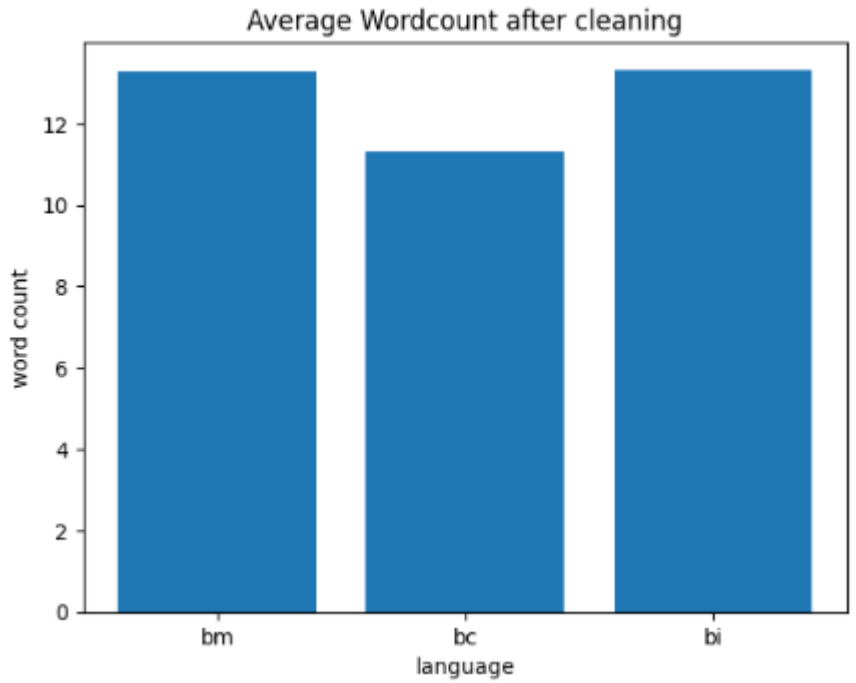


Figure 9 : Average Word Count after cleaning

After removing the outliers, the data preprocessing for social media is considered done. The size of data that is used for data analysis have the size of 120 MB.

```
[pc@g2 output]$ du -sh 11_TOPICS_SENTENCES_DATA
120M    11_TOPICS_SENTENCES_DATA
[pc@g2 output]$
```

Figure 10 : File size after removing the data

3.2.2 Data Preprocessing In Wikipedia

The diagram below illustrates the data preprocessing steps that are applied to the social media dataset.

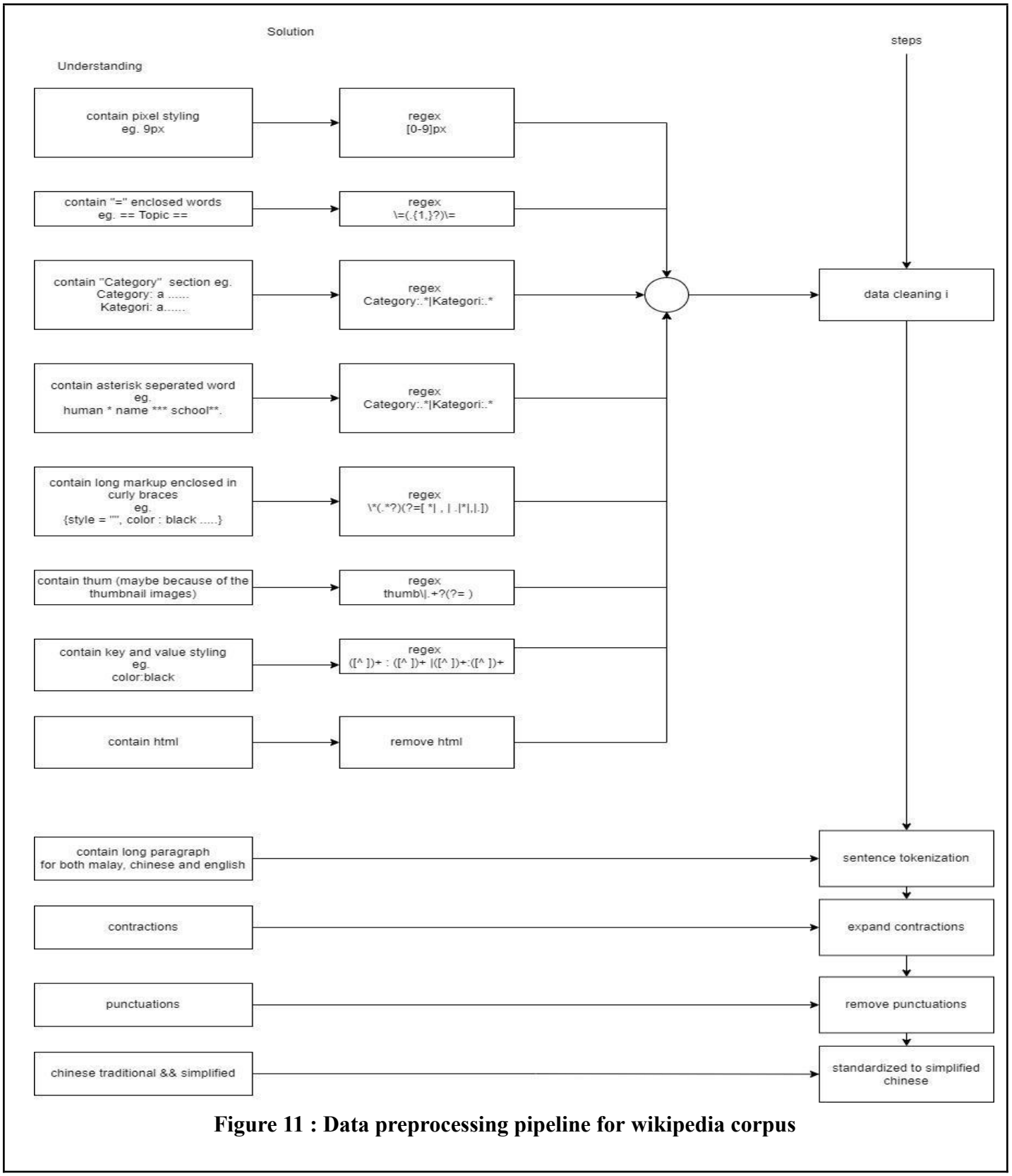


Figure 11 : Data preprocessing pipeline for wikipedia corpus

Step 1: Preprocessing data that will affect the sentence tokenization

First step was text cleaning with removing HTML tags and the following regular expressions to target the removal of specific elements within the corpus which can affect sentence tokenization using the pyspark regex_replace function.

Regular Expression	**Purpose**
"[0-9]px"	Clearing pixels
"\=(.{1,}?)\= "	Clearing words enclosed by "=" sign, e.g. ==Topic==
"Category:.*|Kategori:.*"	Clearing "Category: " or "Kategori: " section in the Wikipedia corpus
"*(.*?)(?=[*| , | .|*|,|.])"	Clearing inline CSS styles
"\{(["http\S+"^}]+)\}"	Clearing URL links
"thumb|.+?(?=)"	Clearing thumbnail images
"([^])+ : ([^])+ |([^])+:([^])+"	Clearing key-value pair type styling

The diagram below shows the effect of data cleaning for those markdown syntax, it is obvious that the second diagram has a clearer and more readable text compared to the first diagram. This also help the sentence tokenization can be more accurate

```
 Examination of China. A significant large number (more th
or even not receiving an offer, through JUPAS. The admissi
of Secondary Education (thus eligible to apply JUPAS) make
mparison, between the estimated entrance rate of JUPAS and
 programmes under JUPAS, this has prompted some students s
ows the school candidates achieving minimum Degree Programm
admitted, candidates not entering their programme of Main
EE (2003-2012) {| class="wikitable" |- ! Intake Year !widt
 for eligible students received offer |- | 2012 | 18,212 |
,979 | 70.5% |- | 2007 | 16,520 | 11,525 | 69.8% |- | 2006
uirement and offer in above table stands for degree-level
e programmes. #The list does not include Early Admission S
 width="250" | JUPAS Degree Intake from Main Round Offer !
9,119 | 43.08% |- | 2017 | 48,502 | 21,165 | 43.64% |- | 2
4,442 | 16,619 | 25.79% |} ==Sub-systems== Apart from the
eeds in other areas. ===Sub-system for applicants with a d
ent * Visceral disability * Speech impairment * Autism * M
s could provide to them on their admission. Applicants may
y nominate up to three students from his/her own school. T
emonstrated leadership abilities. The goal of this scheme
 outstanding achievements in extracurricular activities. A
tes they applies to, so the institutes may also consider t
ipate in more extracurricular activities during their seco
10/11. It was tailor-made for the Secondary 6 students who
y of Hong Kong or the Hong Kong University of Science and
ished in the academic year of 2011/12. == Footnotes == ==
ertificate of Education Examination * UCAS * Matriculation
```

Figure 12 : Data without cleaning those markdown syntax

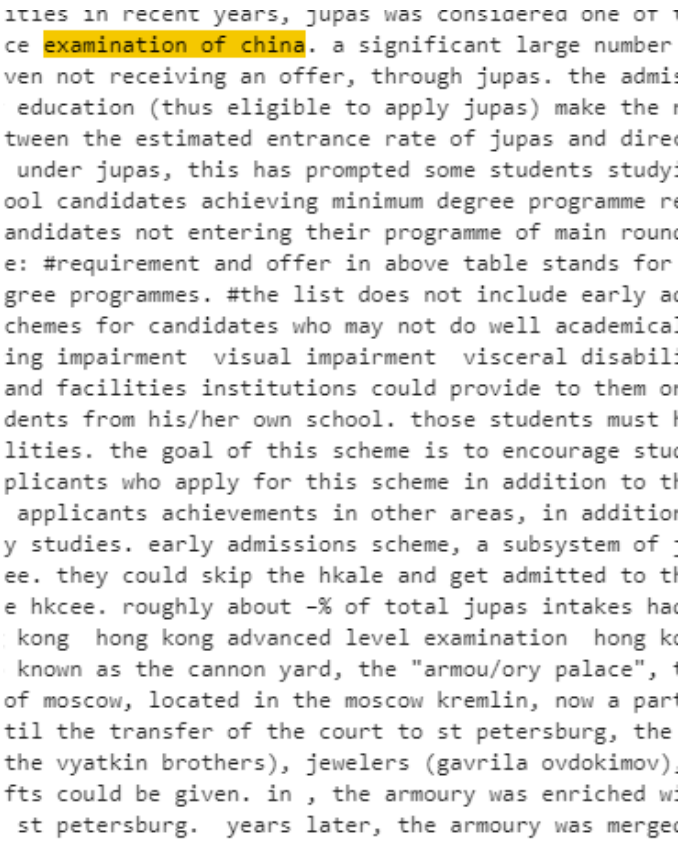


Figure 12 : Data after cleaning those markdown syntax

Step 2: Sentence tokenization

To break a large paragraph into sentences, the sentence tokenization is carried out. For English and Malay wiki data, the nltk sent_tokenize is used. While for chinese, the "HarvestText" library is used as sentence tokenizer. After the sentence tokenization, it will result in an array of strings, hence use the pyspark explode function to flatten the column into string column.

Step 3: Data cleaning

After the sentences are tokenized, some data cleaning procedure is carried out which includes those in social media in the previous section. It is noted that contraction expansion must be done before punctuation removal. All Chinese characters are also standardized into Simplified Chinese.

Step 4: Pyspark Sentence Function

The last procedure in data cleaning would be to put the sentences into the pyspark sentence function because it has the capability to remove punctuations, emoji, and also could do some sentence splitting.

Step 5: Word Count / Remove Stopwords / Ranking & Order Descendingly
The word Count program is the same as social media. The diagrams below illustrate the result of the word count.

Chinese Wiki

word	word_count	id
年	863100	1
行星	717160	2
年月日	714212	3
小行星	711310	4
一个	398627	5
人	369625	6
位于	315610	7
中国	279996	8
人口	276993	9
公里	276200	10

Malay Wiki

word	word_count	id
terletak	189075	1
komun	67626	2
kampung	66825	3
daerah	53922	4
perbandaran	51275	5
sekolah	50769	6
perancis	44752	7
filem	41292	8
kebangsaan	40883	9
google	34995	10

English Wiki

word	word_count	id
also	156098	1
new	133806	2
one	133231	3
first	129372	4
united	113027	5
states	108925	6
county	98812	7
city	98727	8
two	91922	9
american	86644	10

Figure 13 : Top 10 words with highest frequency in each wiki category

Step 6: Word Count / Remove Stopwords / Ranking & Order Descendingly
The word Count program is the same as social media. The diagrams below illustrate the result of the word count.

Step7: Choose the keywords and select the sentences that contain the keywords

The process is similar to the social media keyword filtering. For the Mandarin Wikipedia corpus, the keywords chosen were: 中国, 香港, 台湾, 日本, with the topic being the recent news of tensions between China and its territory, especially with Japan.

For the English Wikipedia corpus, the keywords chosen were: university, school, college, student, scholar, department, with the topic being education.

For the Malay Wikipedia corpus, the keywords chosen were: kampung, sekolah, daerah, barat, utara, komun, with the topic being rural communities.

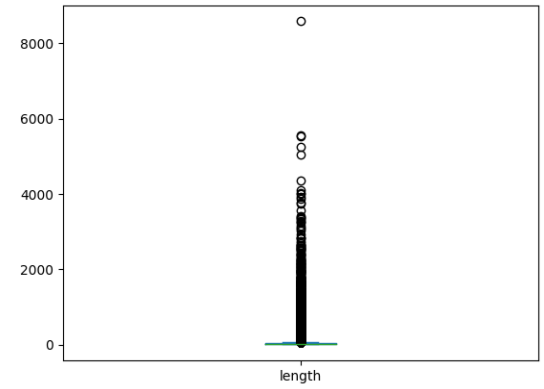
The row count after selecting the keyword is summarized in the table below:

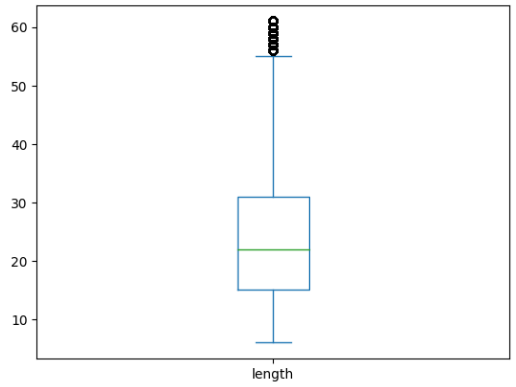
Language	Row Count
Malay wiki	377552
English wiki	814449
Chinese wiki	1048849

Step 8: Create sentence length statistics
Same procedure as social media one

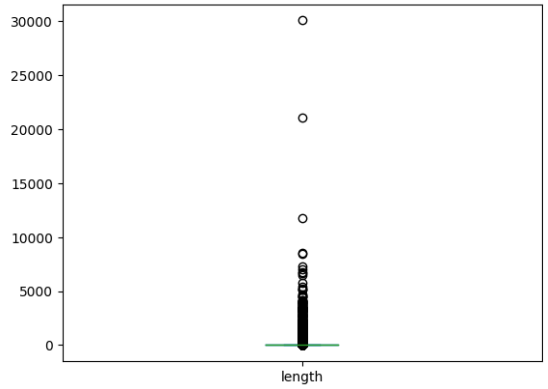
Step 9: Remove those sentences with less than 5 tokens and too many tokens (outliers).

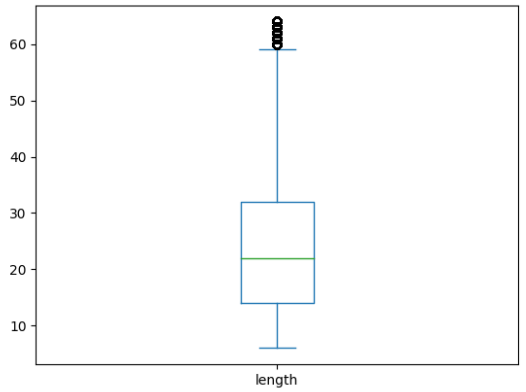
English wiki



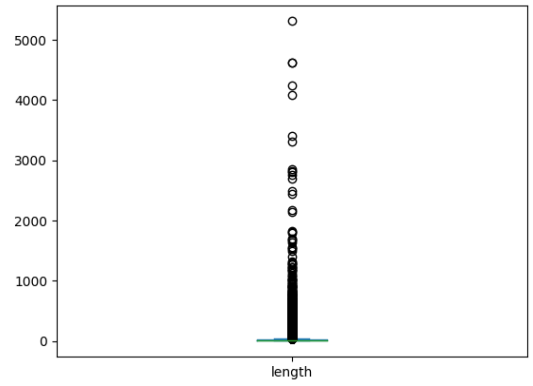


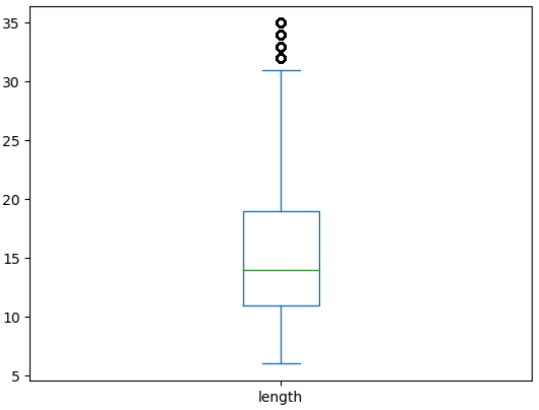
Chinese wiki





Malay wiki





After removing the outliers, the row count and data size is summarized as shown below:

Wiki	Row Count	File Size
English	715594	51M
Malay	318967	13M
Chinese	955879	52M

4.0 Data Storage and Structure Design

4.1 Data Storage

Local File System

In the data collection process, before the ingestion, the collected data is imported into the local File System and comes together with the OS, which in our case is CentOS 7. In jupyter notebook, most of the storage used is the local file system, except in the last stage that the file is stored into the hdfs before load into hive.

HDFS

The hdfs is used to store the data after the data ingestion process is done. In the combined version of code, the hdfs is also used to temporarily store the result of some preprocessed data and was deleted after the next step is completed. The file format used most in this project is avro.

HIVE

Hive is the database residing on hdfs that is used to store the output data that is finalized during the ETL process. The reason behind Hive was chosen for storing the data was because it is simple and provides the sql api for us to interact with the database. The data import also is simple since the entire dataframe can be written into the hive database. We can use the sql to do the analytics. As for MapReduce, it is well known as infamously difficult to program. The file format used in hive is Optimized Row Columnar (ORC) file format that is efficient for hive to store the data.

Custom Data Caching Structure

Due to the on-memory process being costly in the current infrastructure, and the data process a long time easily gone or unresponsive, the custom local caching is applied to decrease the burden on the memory by leveraging the larger disk storage spaces. So whenever the current process fails, at least the previous version of data is saved and can continue from that point. The saved data serve as a checkpoint. In jupyter notebook, this is done by using the local file system while in the combined version of code, this is done by using the hdfs.

For each step of data processing, the data stage would be saved, or written into the local file directory or hdfs, and refresh the spark session to free up the memory or even reboot the machine if it is necessary, and then load back into the environment. After the next step is completed, the local copy would be deleted to free up the local storage space. If the result is written into avro file format, it will become even smaller when loading into the hadoop cluster, hence saving the cost.

4.2 Structure Design

Just like the data warehouse, the data structure designed is not stored in third normal form hence there was redundancy. We choose the fast query rather than storage space because if not optimized in this way, the query runtime is just endless.

4.2.1 General Columns

The general columns are the attributes that are similar for both social media and wikipedia dataset.

GramTable

Column	Explanation
middle_key	Use to identify whether the middle of n5 or n3 gram is the keyword. true if the keyword is in the middle of the n-gram.
gram_type	Used to tell us which n_gram type for this row of data like bigram, trigram, n4 gram and so on.
sentence_id	User to link between source_data and gram_table, identifier for the sentence that the n-gram was generated from.
pos	The iteration number or index of possible n-gram generated from the sentence. For example: I feel hungry: I feel => pos = 1 Feel hungry => pos = 2 This is accomplished by using the pyspark posexplode function.
token_gram	The word / tokens sequence up to n number of words based on generated n-gram.
token_gram_f	Frequency of the occurring token pattern of the n-gram, this is a pre aggregated field, and might be redundant but make query much faster.
tag_gram	POS tag of each token in the generated n-gram which corresponds to the token_gram.
tag_gram_f	Frequency of the occurring POS-Tag pattern of the n-gram, pre aggregated field for fast analysis.
uniqueID	The number that is unique across the table.

SourceData Table

Column	Explanation
sentence_id	User to link between source_data and gram_table, unique sentence identifier.
original	Contain the original sentence
tag	Contain the part of speech tagging from original sentence
token	The tokens in a sentence after word tokenization

4.2.1 Social Media Structure Design

For storing the social media data in an organized way, the database named "social_media_db" is created to store the table related to social media. To specify which database or namespace to be used, the spark.sql("use social_media_db") is issued, else the system would just use the default namespace.

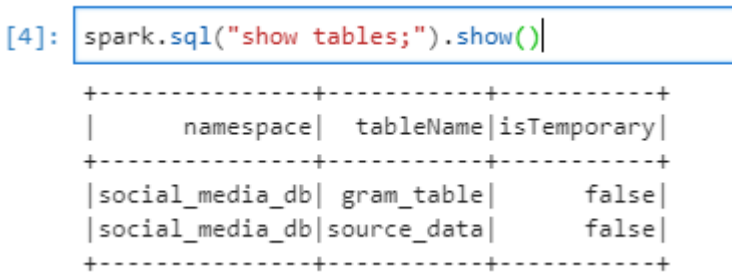


Figure 14 : Tables in the namespace social_media_db

The special column that make gram_table different in social media dataset is the lang_gram and lang_gram_f, where the lang_gram is the language pattern corresponding to the token_gram and the lang_gram_f is the frequency of the occurring language pattern of the n-gram, it is the pre-aggregated field that counting the frequency of the lang_gram, in other words, the language pattern frequency. The chinese_tag here is used to store the tag generated from jieba, before standardizing it to normal part of speech tagging, hence containing null because some rows are not Chinese sentences. The language column is used to identify the language of the sentences.

Figure 15 : Social Media GramTable and source_data schema

4.2.2 Wikipedia Structure Design

There is a different design between the code in Jupyter notebooks and the combined version for wikipedia. In the Jupyter Notebook, all the three languages are put together into one single table while in the combined code version, the structure is divided into three tables for 3 different languages wiki data.

Jupyter Notebook Data Structure Version

In jupyter notebook, the wikipedia data is stored inside the default namespace, which contains the source_data and wiki_gram table.

```
+---------+-----------+-----------+
|namespace|  tableName|isTemporary|
+---------+-----------+-----------+
|  default|source_data|      false|
|  default|  wiki_gram|      false|
+---------+-----------+-----------+
```

Figure 16 : Wikipedia Schema in Default Namespace (Jupyter Notebook version)

SourceData Table

```
spark.sql("desc source_data;").show()
```

```
+-----------+-------------+-------+
|   col_name|    data_type|comment|
+-----------+-------------+-------+
|sentence_id|       bigint|   null|
|   original|       string|   null|
|        tag|array<string>|   null|
|   language|       string|   null|
|      token|array<string>|   null|
| normal_tag|array<string>|   null|
+-----------+-------------+-------+
```

GramTable

```
spark.sql("desc wiki_gram;").show()
```

```
+-----------+---------+-------+
|   col_name|data_type|comment|
+-----------+---------+-------+
|sentence_id|   bigint|   null|
|        pos|      int|   null|
| token_gram|   string|   null|
| tok_gram_f|   bigint|   null|
|   tag_gram|   string|   null|
| tag_gram_f|   bigint|   null|
|  gram_type|   string|   null|
|containsKey|  boolean|   null|
| middle_key|  boolean|   null|
|   language|   string|   null|
+-----------+---------+-------+
```

Figure 17 : Wikipedia Schema (Jupyter Notebook version)

Combined code Data Structure Version

There is adjustment for the data structure design in the combined code version. The tables related to wikipedia are all stored into the database named under "wikipedia_db". There are six tables because for each language, it requires two tables which are a gram table and source data to hold the data.

```
: spark.sql("show tables;").show()
```

```
+------------+--------------+-----------+
|   namespace|     tableName|isTemporary|
+------------+--------------+-----------+
|wikipedia_db| gram_table_en|      false|
|wikipedia_db| gram_table_ms|      false|
|wikipedia_db| gram_table_zh|      false|
|wikipedia_db|source_data_en|      false|
|wikipedia_db|source_data_ms|      false|
|wikipedia_db|source_data_zh|      false|
+------------+--------------+-----------+
```

Figure 18 : Tables inside namespace wikipedia_db

Actually the gram table structure and source data structure is identical for all the three languages, as shown in the diagrams below. This is to make sure the data structure is consistent and easier to remember and understand when doing the query.

English Gram Table Structure

```
spark.sql("desc gram_table_en;").show()
```

```
+------------+---------+-------+
|    col_name|data_type|comment|
+------------+---------+-------+
| sentence_id|   bigint|   null|
|         pos|      int|   null|
|  token_gram|   string|   null|
|token_gram_f|   bigint|   null|
|    tag_gram|   string|   null|
|  tag_gram_f|   bigint|   null|
|   gram_type|      int|   null|
| containsKey|  boolean|   null|
|  middle_key|  boolean|   null|
|    uniqueID|   bigint|   null|
+------------+---------+-------+
```

English Source Data Structure

```
spark.sql("desc source_data_en;").show()
```

```
+-----------+-------------+-------+
|   col_name|    data_type|comment|
+-----------+-------------+-------+
|sentence_id|       bigint|   null|
|   original|       string|   null|
|        tag|array<string>|   null|
|      token|array<string>|   null|
|   uniqueID|       bigint|   null|
+-----------+-------------+-------+
```

Malay Gram Table Structure

```
spark.sql("desc gram_table_ms;").show()
```

```
+------------+---------+-------+
|    col_name|data_type|comment|
+------------+---------+-------+
| sentence_id|   bigint|   null|
|         pos|      int|   null|
|  token_gram|   string|   null|
|token_gram_f|   bigint|   null|
|    tag_gram|   string|   null|
|  tag_gram_f|   bigint|   null|
|   gram_type|      int|   null|
| containsKey|  boolean|   null|
|  middle_key|  boolean|   null|
|    uniqueID|   bigint|   null|
+------------+---------+-------+
```

Malay Source Data Structure

```
spark.sql("desc source_data_ms;").show()
```

```
+-----------+-------------+-------+
|   col_name|    data_type|comment|
+-----------+-------------+-------+
|sentence_id|       bigint|   null|
|   original|       string|   null|
|        tag|array<string>|   null|
|      token|array<string>|   null|
|   uniqueID|       bigint|   null|
+-----------+-------------+-------+
```

Chinese Gram Table Structure

```
spark.sql("desc gram_table_zh;").show()
```

```
+------------+---------+-------+
|    col_name|data_type|comment|
+------------+---------+-------+
| sentence_id|   bigint|   null|
|         pos|      int|   null|
|  token_gram|   string|   null|
|token_gram_f|   bigint|   null|
|    tag_gram|   string|   null|
|  tag_gram_f|   bigint|   null|
|   gram_type|      int|   null|
| containsKey|  boolean|   null|
|  middle_key|  boolean|   null|
|    uniqueID|   bigint|   null|
+------------+---------+-------+
```

Chinese Source Data Structure

```
spark.sql("desc source_data_zh;").show()
```

```
+-----------+-------------+-------+
|   col_name|    data_type|comment|
+-----------+-------------+-------+
|sentence_id|       bigint|   null|
|   original|       string|   null|
|        tag|array<string>|   null|
|      token|array<string>|   null|
|   uniqueID|       bigint|   null|
+-----------+-------------+-------+
```

5.0 Task 1 - Language Classification

Language classification is only conducted in social media dataset as the data in social media can have mixed languages in a sentence whereas Wikipedia data does not. To carry out the language classification, there are few libraries that can be used. In our case, we are using the **Lingua** library to determine the language of the text given. The language classification is divided into two levels, which is in sentence level and word level.

5.1 Sentence Level Language Classification

After the data is being tokenized, it is time to detect the language of the sentences. Firstly, the Lingua library is imported and the we use the LanguageDetectorBuilder to build the LanguageDetector object, where the languages are setted as Language.ENGLISH, Language.CHINESE, Language.MALAY because these three languages are our focus. After that, the normal python function that is used to detect the language is created using the LanguageDetector. In order to apply this python function into dataframe, it is required to use the **"udf"** function wrapper from pyspark.sql.functions. After the udf is created, finally it can be applied to the pyspark dataframe by using the **withColumn** method. The detected language is stored in the column named "language" in the dataframe. Diagram below shows the distribution of languages based on the sentences.

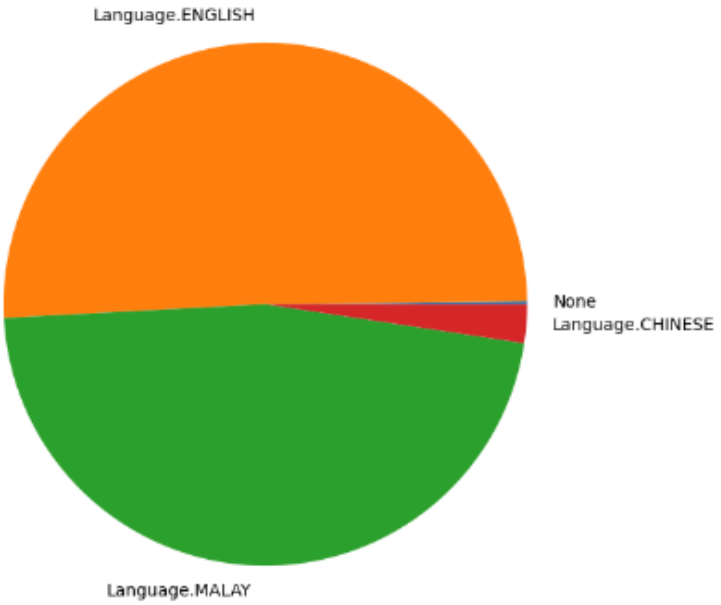


Figure 19 : Languages classification by sentences in social media corpus

5.2 Word Level Language Classification

It is required to detect the language of each token to form the language pattern using n-gram in the later process. The steps of word level language classification is as follows:

1. Create vocabulary / unique token from current filtered sentences.
2. Retrieve the previous word count vocabulary which already has language detected.
3. Left join current vocabulary with the word count vocabulary.
4. Null sets are those with language labels absent.
5. Detect the Null set, which is small.
6. Merge the null set with the non null set.
7. The vocabulary with language is well prepared.
8. Replace the tokens with language labels.

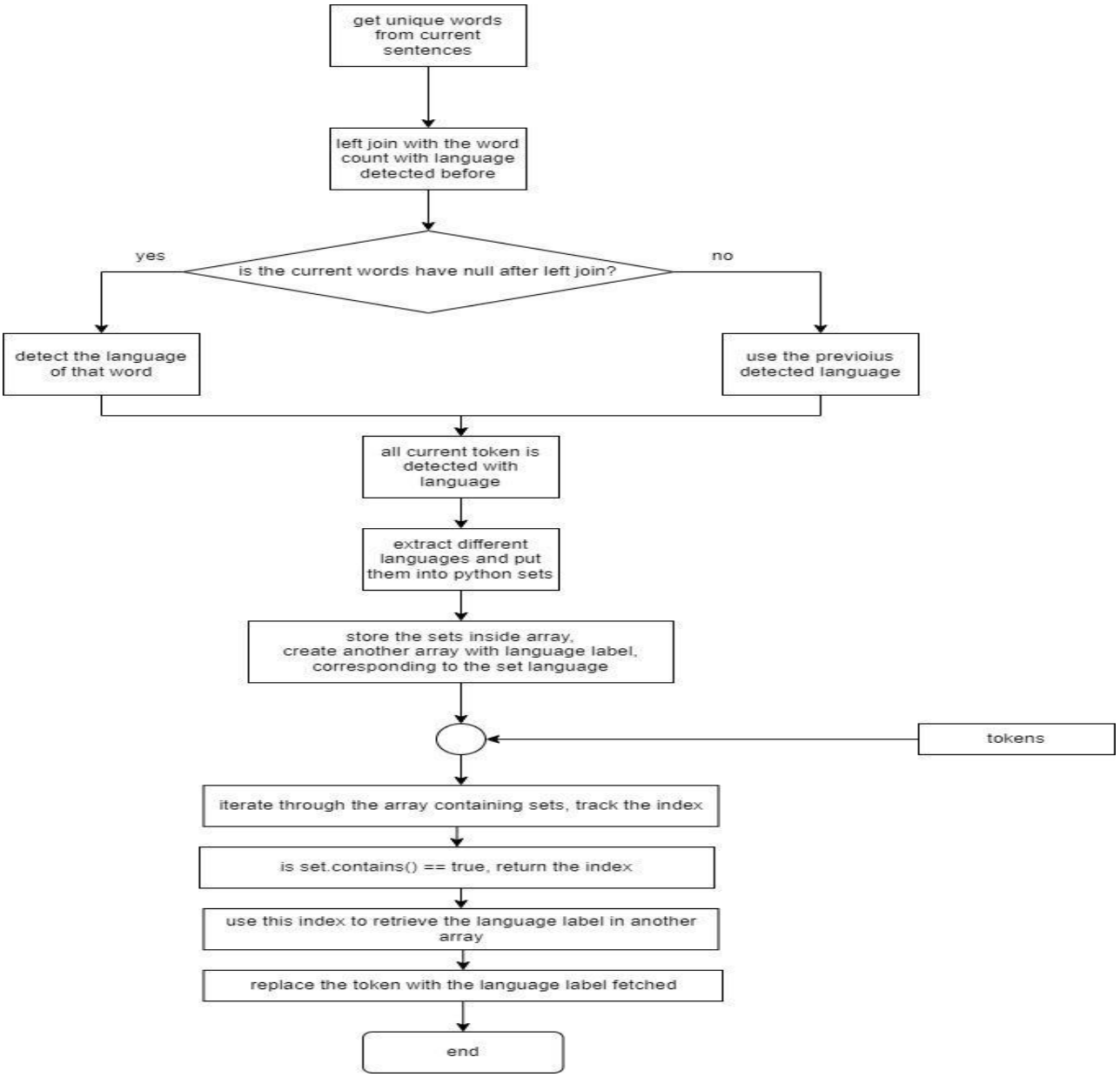


Figure 21 : Language detection for each token

The vocabulary associated with the language label is classified into python sets, because set data structures have access efficiency of O(1) which is extremely fast. After that, to detect each word in a sentence, we only need to replace the correct language label associated with the set of words by using the contains method. It is required to detect the language of each token to form the language pattern using n-gram in the later process. The steps of word level language classification is as follows:

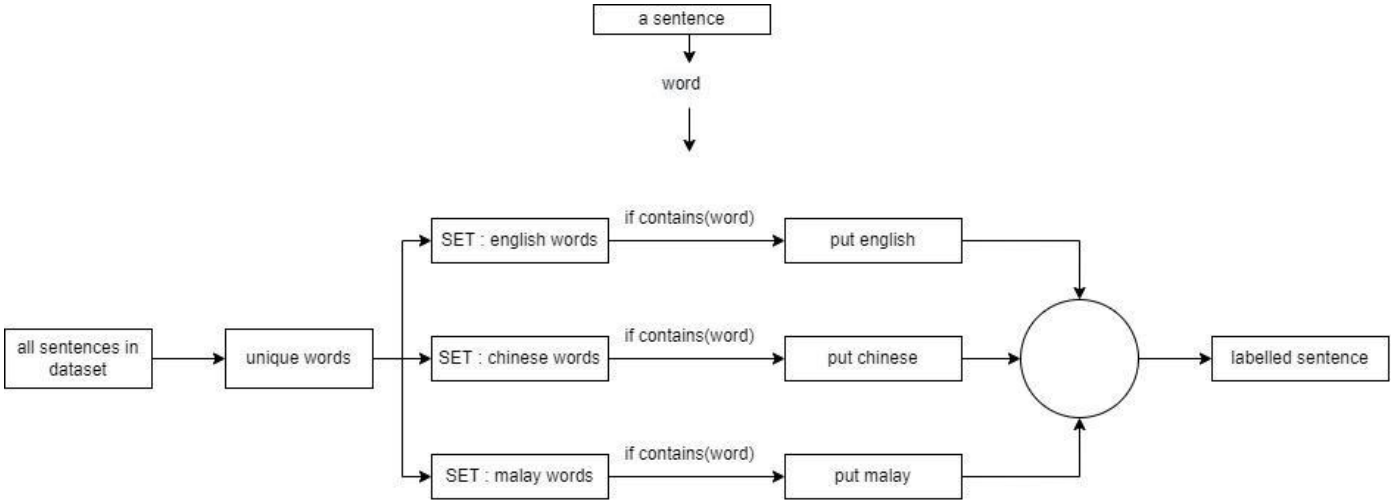


Figure 22 : Detection in action

Language detection for sentence

```
from pyspark.sql.functions import col, udf
from lingua import Language, LanguageDetectorBuilder
languages = [Language.ENGLISH, Language.CHINESE, Language.MALAY]


nonDistDetector = LanguageDetectorBuilder.from_languages(*languages).build()
withDistDetector =
LanguageDetectorBuilder.from_languages(*languages).with_minimum_relative_distance(0.05).build()


def lingua_detect_dist(text):
    global withDistDetector

    try:
        if (withDistDetector == None):
            withDistDetector = \

LanguageDetectorBuilder.from_languages(*languages).with_minimum_relative_distance(0.05).build()
        return str(withDistDetector.detect_language_of(text))

    except:
        return "error"
```

```
def lingua_detect(text):
    global nonDistDetector

    try:
        if (nonDistDetector == None):
            nonDistDetector = LanguageDetectorBuilder.from_languages(*languages).build()
        return str(nonDistDetector.detect_language_of(text))
    except:
        return "error"

detectWithDistUDF = udf(lambda z: lingua_detect_dist(z))
detectUDF = udf(lambda z: lingua_detect(z))
```

Apply UDF to the sentences

```
df = df.withColumn("lang", detectUDF(F.col("text"))) ## detectUDF is custom made UDFa
```

Apply UDF to the unique words

```
vocabLang = df.withColumn("lang", detectUDF(F.col("word"))) ## detectUDF is custom made UDF
```

Effectively replace token with its corresponding language using Python set

```
oov = set(oovSet.rdd.flatMap(lambda x : x).collect())
bi = set(engSet.rdd.flatMap(lambda x : x).collect())
bm = set(bmSet.rdd.flatMap(lambda x : x).collect())
bc = set(bcSet.rdd.flatMap(lambda x : x).collect())
replacement = [bi, bm, bc, oov]
languages = ["en","ms","zh","oov"]



def get_lang(text):
    for i in range(4):
        if text in replacement[i]:
            return languages[i]
    return "oov"

def get_lang_arr(arr):
    return [get_lang(x) for x in arr]


get_lang_udf = udf(lambda x : get_lang_arr(x), ArrayType(StringType()))
```

Diagram below shows the distribution of languages based on each word.

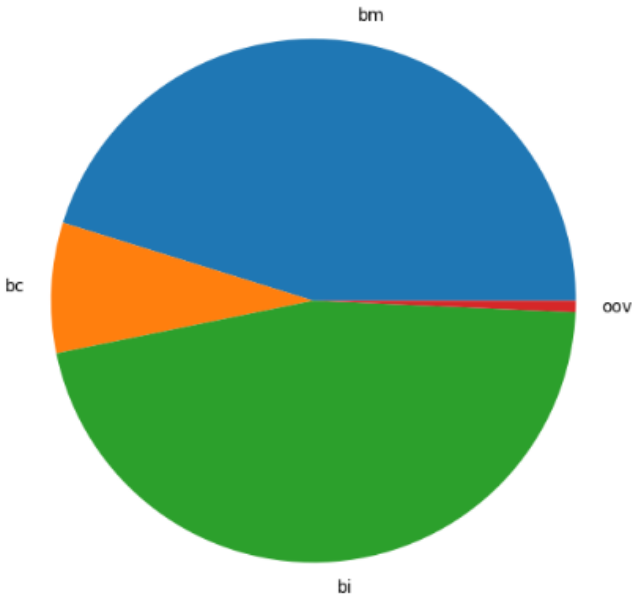


Figure 23 : Languages classification by each unique words from the social media corpus

Source code for Language detection

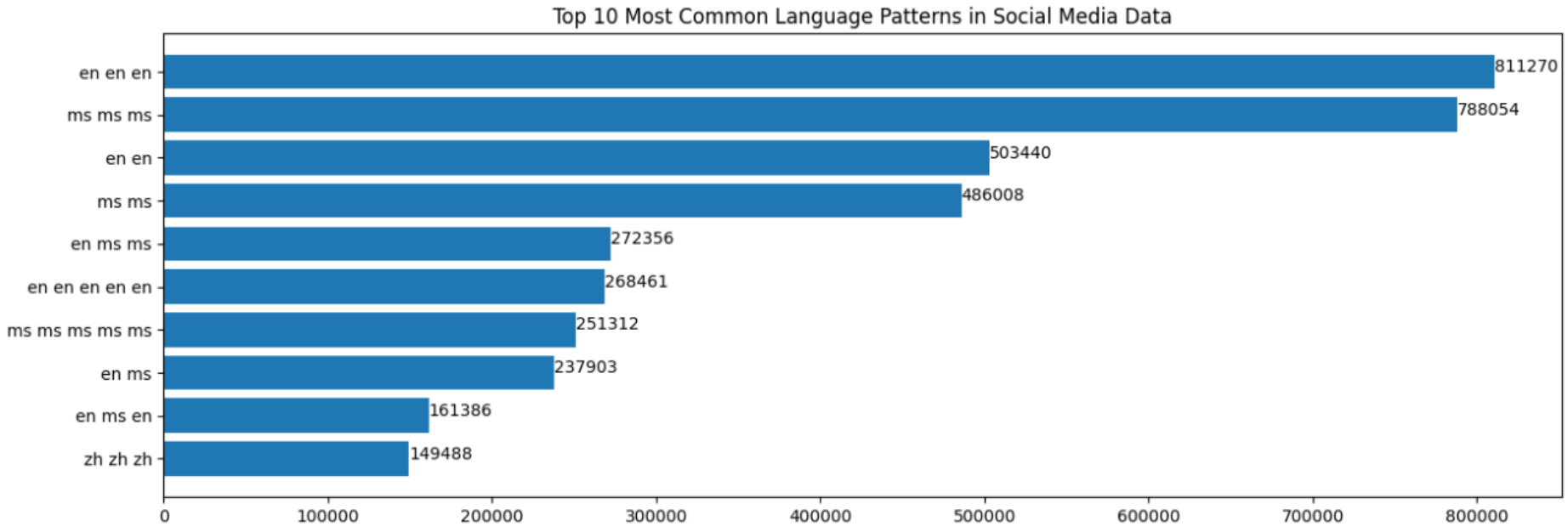
```
|[kualiti, yg, bagus, trusted, seller, barang, sampai, tanpa, kerosakan]
|[ms, en, ms, en, en, ms, ms, ms, ms]
|[memang, beli, dgn, seller, ni, dah, lama, dah]
|[ms, ms, en, en, ms, ms, ms, ms]
|[terima, kasih, seller, nanti, repeat, order, lgi]
|[ms, ms, en, ms, en, en, ms]
|[cuma, seller, lambat, sikit, prepare, utk, pos]
|[ms, en, ms, ms, en, ms, en]
|[brg, sampai, dlm, baik, tp, seller, peram, ari, ls, oder, baru, pos, jnt, mmg, laju]
|[en, ms, en, ms, ms, en, ms, ms, en, en, ms, en, en, ms, ms]
|[banyak, kali, dah, repeat, order, dengan, seller, ni]
|[ms, ms, ms, en, en, ms, en, ms]
|[layanan, terbaikkk, respon, terbaikkk, tindakan, seller, terbaikkk, seller, x, layak, klu, bintang]
|[ms, ms, en, ms, ms, en, ms, en, en, ms, ms, ms]
```

Figure 24 : Outcome of Language detection for each token

```
|
|[包装，来，的，时候，非常，好，quality，is，good]
|[zh，zh，zh，zh，zh，zh，en，en，en]
|
|[食材，都，很，新鲜，材料，十足，如图所示，般，好，应有尽有]
|[zh，zh，zh，zh，zh，zh，zh，zh，zh，zh]
|
|[太，可爱，了，又，好，萌]
|[zh，zh，zh，zh，zh，zh]
|
|[好，想，吃]
|[zh，zh，zh]
|
|[包装，很，好，还，会，在，买]
|[zh，zh，zh，zh，zh，zh，zh]
|
|[送货，速度，很快，产品包装，很，好，质量，也，是，很，赞]
|[zh，zh，zh，zh，zh，zh，zh，zh，zh，zh，zh]
|
|[货物，都，完整，的，收到，卖家，的，服务，很，好，我，很，满意，谢谢，卖家]
|[zh，zh，zh，zh，zh，zh，zh，zh，zh，zh，zh，zh，zh，zh，zh]
|
|[东西，包装，得，很，好，东西，是，normal，expiry，date]
|[zh，zh，zh，zh，zh，zh，zh，en，en，en]
|
```

Figure 25 : Outcome of Language detection for each token

Social Media Corpus Results



6.0 Task 2 - POS Tag

Part-Of-Speech (POS) Tagging is conducted for both social media and Wikipedia corpus by assigning a label to each token in a sentence to indicate the grammatical patterns of a sentence. For the social media dataset, POS tagging is conducted after performing the language classification by using the Malaya library for tagging Malay and English words, and Jieba library for tagging Chinese words. For the Wikipedia dataset, the Spark NLP library is used for tagging the English corpus; Malaya library is used for tagging the Malay corpus; Jieba library is used for tagging the Chinese corpus.

6.1 POS Tagging Chinese Sentences

The procedure of POS tagging chinese sentences in social media chinese and wikipedia chinese is the same.

It is noted that if the jieba pos tagger encountered the english words, it will tag the token with the labels “eng”, to resolve this problems, the python translation library is used to improve this situation. However, there are limitations in the python translation library such as quantity of words allowed to be detected and some required payment for api key, if the translation package is unavailable, just tag the english words as X. The pos tag with translation capability is as following:

1. load the wikipedia chinese data and then use the jieba library to get the token and tag.
2. After that, check whether the tag consists of english, if yes, then get the indices as well as get the english token.
3. Next, replace the translated word to the original string.
4. Furthermore, use jieba to do the POS tag again.
5. If the token is equal to the token generated previously, return the non-translated token and translated tag else the token and the tag generated previously will be returned.
6. Lastly, convert the tag to compatible with Malaya notation.
7. Now a column with 2D array is formed, use the flatmap to separate the 2D array of token and tag into two columns of 1d array.

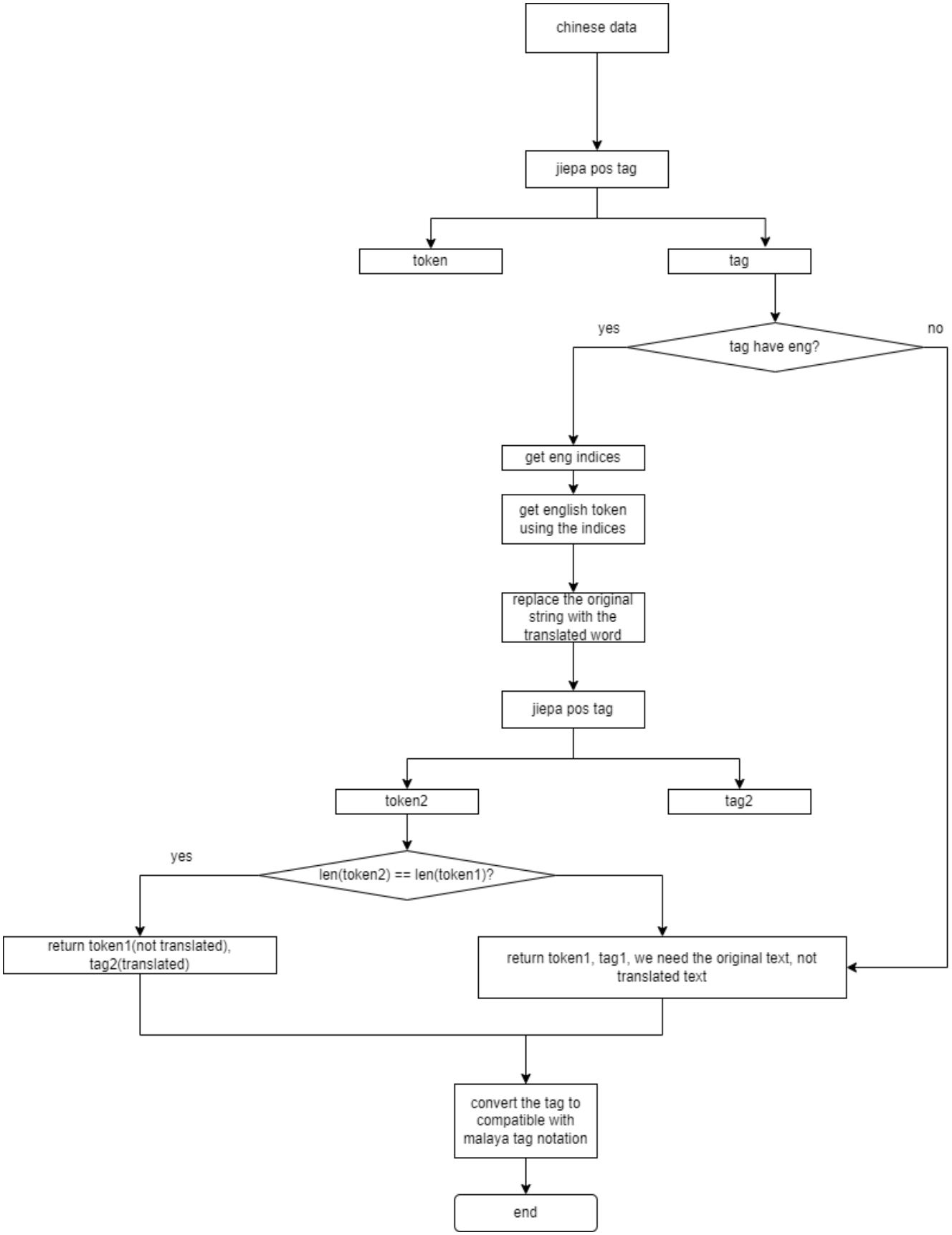


Figure 26 : Flowchart of POS tag chinese

Pos tagging on Social Media Chinese and wikipedia

```
from translate import Translator
from pprint import pprint
from more_itertools import locate
import translators as ts
import jieba
import jieba.posseg as pseg
from translate import Translator


def translate(text):
    translator= Translator(to_lang="Chinese")
    return translator.translate(text)

def find_indices(l, item_to_find):
    return locate(l, lambda x: x == item_to_find)

def get_token_tag(text):
    words = pseg.cut(text, use_paddle=True) #paddle模式
    w =[]
    t =[]
    for a, b in words:
        if a != " ":
            w.append(a)
            t.append(b)
    return w,t


def tag_chinese_with_translate_capability(text):
    r = get_token_tag(text)
    iw = r[0]
    it = r[1]
    initialLength = len(iw)
    if "eng" in it:
        engIndex = find_indices(it, "eng")
        engWords = []
        for i in engIndex:
            engWords.append(iw[i])

        for i in engWords:
            translated = translate(i)
            text = text.replace(i, translated)


        nr = get_token_tag(text)

        lengthAfterTranslate = len(nr[0])

        if(lengthAfterTranslate == initialLength):
            return iw, nr[1]
        else:
            return iw,ir

    else:
        return iw,it

udfTranslateTagChinese = udf(lambda x : tag_chinese_with_translate_capability(x), ArrayType(StringType()))

if bc.count() > 0:
        chineseTag = bc.withColumn("chinese_tag", udfTranslateTagChinese(bc.text))
```

```
chineseTag = chineseTag.rdd.map(lambda x : (x.text, x.chinese_tag[0], x.chinese_tag[1], x.lang)).\
    toDF(["original","token","chinese_tag", "language"])

init_replacement_dict(jieba_replace_csv)

withNormalTag = chineseTag.withColumn("tag", stdTokenUDF(col("chinese_tag")))\
    .withColumn("language", lit("chinese"))
```

The outcome of the POS tagging is shown in below tables.

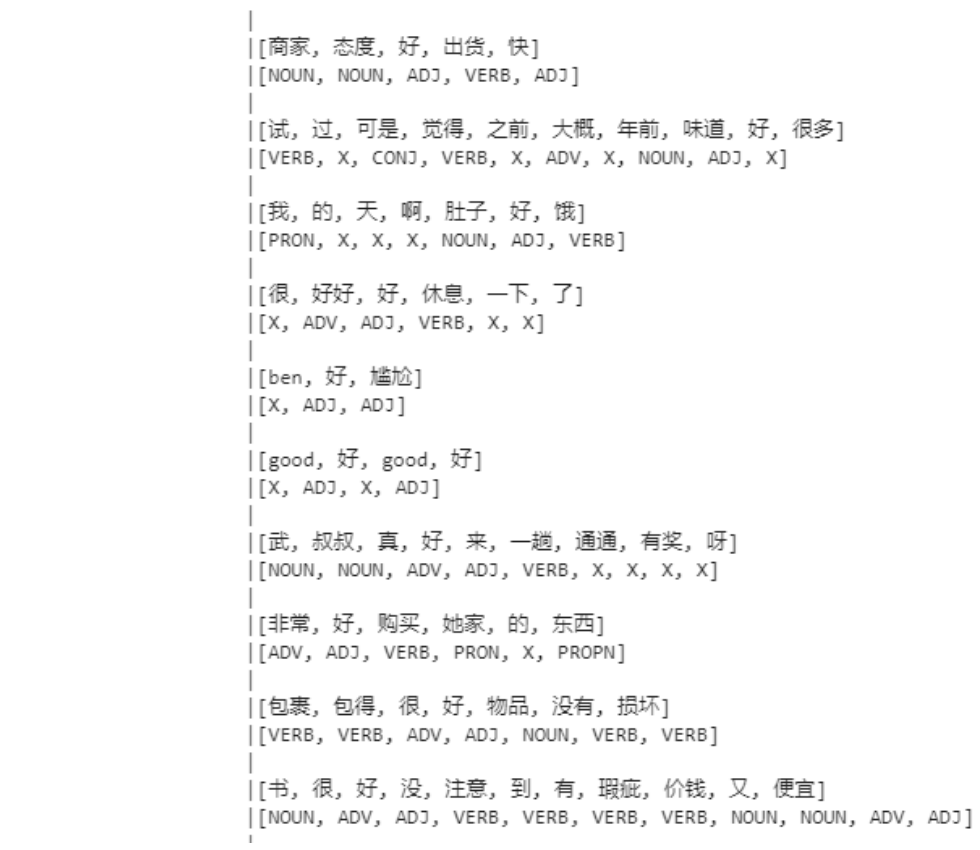


Figure 25 : Outcome of POS tagging for social media chinese

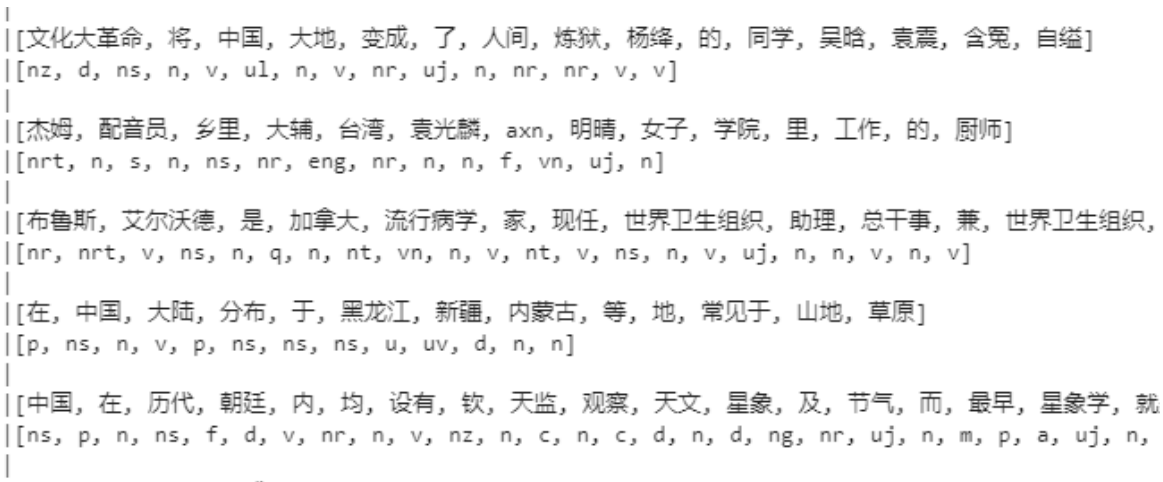


Figure 26 : Outcome of POS tagging for wikipedia chinese

6.2 POS Tagging English Sentences

Wikipedia

SparkNLP is used to POS tag the wikipedia english dataset because the language is quite pure without mixture. Before using the sparkNLP, it is required to include the packages in the Spark Submit arguments like the following:

The steps of using the sparkNLP is as follows:

1. Create DocumentAssembler
2. Create SentenceDetector
3. Create Tokenizer
4. Create PosTagger using the model `"pos_ud_ewt"`
5. Fit these object into the pipeline
6. Fit the dataframe into the pipeline, note that the column name must be renamed as sentence and then transform the dataframe.
7. Using the pyspark selectExpr to retrieve the postag and the token

```
import sparknlp
from sparknlp.base import *
from sparknlp.annotator import *
from pyspark.ml import Pipeline
from pyspark.sql.functions import udf, col
import os

document_assembler = DocumentAssembler() \
.setInputCol("sentence") \
.setOutputCol("document")

sentence_detector = SentenceDetector() \
.setInputCols(["document"]) \
.setOutputCol("sentence")

tokenizer = Tokenizer().setInputCols(["document"]).setOutputCol("token")

posTagger = PerceptronModel.pretrained("pos_ud_ewt", "en") \
.setInputCols(["document", "token"]) \
.setOutputCol("pos")

pipeline = Pipeline(stages=[
document_assembler,
sentence_detector,
tokenizer,
posTagger])


df = read_avro(spark, hdfs_working_path+"sentence_without_out")
```

```
df = df.withColumnRenamed("text", "sentence")
result = pipeline.fit(df).transform(df)
final = result.selectExpr("token.result as token","pos.result as pos")

print("==========BI SENTENCES WITH TAGGING==========")
write_avro(final, hdfs_working_path+"bi_sentence_with_tagging")
```

Social Media

Malaya library is used to POS tag the english sentence from social media. This is because it contains many mixtures of Malay and english. Malaya also has the capability to post English sentences. Note that using the Malaya library is actually a tedious task that will consume most of the time in this task.

The step of pos tag is as following:

1. Create a user defined function that returns a token and tag if passed the text in it.
2. Wrap the function with the pyspark UDF so that it can be applied to the dataframe.
3. Read the file
4. Split the file into n files with each file about 1000 rows.
5. Iterate the splitted files and write the pos tag result into avro file format.
6. Delete all the previous splitted files.
7. Merge all the files with pos tag and token.
8. Delete all the splitted files.
9. Convert the column with 2d array into 2 columns of 1d array, token and tagging.
10. done

```
import malaya
from pyspark.sql.functions import col, udf, size, split
from pyspark.sql.types import StringType, ArrayType

model = None
def posTagBMBI(text):
    global model
    if model == None:
        model = malaya.pos.transformer(model = 'albert', quantized=False)
        r = model.predict(text)
        return [x[0] for x in r],[x[1] for x in r]
    else:
        r = model.predict(text)
        return [x[0] for x in r],[x[1] for x in r]

posUDF = udf(lambda x : posTagBMBI(x), ArrayType(ArrayType(StringType())))


df = read_avro(spark, hdfs_working_path+"bm_sentence_without_out")
df = assign_id_column(df)
write_avro(df, hdfs_working_path+"bm_with_id")
df = read_avro(spark, hdfs_working_path+"bm_with_id")

numberOfRows = df.count()
splitInto = math.ceil(df.count()/1000)
segments = get_segments(df, splitInto)


# split to smaller file to easier processing
for i in range(splitInto):
    print("write " + str(i))
    write_avro(segments[i].select("text"), f"{hdfs_working_path}IN/{i}")
    print("LEN = " + str(segments[i].count()))


print(f"the file is splitted into {splitInto} files to ease the prcessing")
```

```
for i in range(splitInto):
    df = read_avro(spark, f"{hdfs_working_path}IN/{i}")
    df.show(5, False)
    tag = df.withColumn("tag", posUDF(df.text))
    write_avro(tag, f"{hdfs_working_path}OUT/{i}")
    df.unpersist()
    print("done execution !")


remove_from_hdfs(f"{hdfs_working_path}IN")


## merge the file to one and remove the out directory
alt = Alternator()


for i in range(splitInto):
    if i == 0:
        merge = read_avro(spark, f"{hdfs_working_path}OUT/{i}")
        write_avro(merge, f"{hdfs_working_path}Merge/{alt.num}")
    else:
        merge = read_avro(spark, f"{hdfs_working_path}Merge/{alt.num}")
        df = read_avro(spark, f"{hdfs_working_path}OUT/{i}")
        union = unionAll(merge, df)
        alt.alternate()
        write_avro(union, f"{hdfs_working_path}Merge/{alt.num}")
        remove_from_hdfs(f"{hdfs_working_path}Merge/{alt.get_alternate()}")

remove_from_hdfs(f"{hdfs_working_path}OUT")
df = read_avro(spark, f"{hdfs_working_path}Merge/{alt.num}")
```

```
|
|[he, matriculated, at, lincoln, college, oxford, june, and, graduated, ba]
|[PRON, VERB, ADP, NOUN, NOUN, NOUN, PROPN, CCONJ, VERB, NOUN]
|
|[on, april, the, university, of, edinburgh, awarded, her, the, honorary, degree, of, lld, in, recognition, of, her, servi
|[ADP, PROPN, DET, NOUN, ADP, PROPN, VERB, PRON, DET, ADJ, NOUN, ADP, NOUN, ADP, NOUN, ADP, PRON, NOUN, PART, NOUN]
|
|[its, second, home, was, a, building, still, standing, at, the, corner, of, pinkney, and, anderson, streets, which, event
pated, africanamericans, before, the, american, civil, war]
|[PRON, ADJ, NOUN, AUX, DET, NOUN, ADV, VERB, ADP, DET, NOUN, ADP, NOUN, CCONJ, NOUN, NOUN, PRON, ADV, VERB, DET, NOUN, NO
|
|[in, jacobi, became, a, professor, and, in, a, tenured, professor, of, mathematics, at, königsberg, university, and, held
|[ADP, NOUN, VERB, DET, NOUN, CCONJ, ADP, DET, ADJ, NOUN, ADP, NOUN, ADP, NOUN, NOUN, CCONJ, VERB, DET, NOUN, SCONJ]
|
|[the, nearest, state, college, is, westfield, state, university, and, the, nearest, state, university, is, the, universit
|[DET, ADJ, NOUN, NOUN, AUX, ADJ, NOUN, NOUN, CCONJ, DET, ADJ, NOUN, NOUN, AUX, DET, NOUN, ADP, NOUN, ADV]
|
|[pastor, taught, at, the, university, of, innsbruck, first, as, a, lecturer, then, as, professor, of, modern, history]
|[NOUN, VERB, ADP, DET, NOUN, ADP, NOUN, ADV, ADP, DET, NOUN, ADV, SCONJ, NOUN, ADP, ADJ, NOUN]
|
```

Figure 27 : Outcome of POS tagging for wikipedia english using spark nlp

```
|
|[seller, are, so, responsive, and, reallyyyyy, helpfull]
|[NOUN, VERB, ADJ, ADJ, CCONJ, PROPN, VERB]
|
|[super, fast, seller, send, out, after, make, payment]
|[ADV, ADJ, NOUN, VERB, NOUN, ADP, VERB, NOUN]
|
|[good, service, and, responsive, seller, but, i, hope, it, will, last, more, than, a, year]
|[PROPN, NOUN, CCONJ, NOUN, NOUN, CCONJ, PROPN, VERB, PRON, VERB, VERB, ADV, ADP, NOUN, NOUN]
|
|[tqvm, seller, and, shopee]
|[NOUN, NOUN, CCONJ, NOUN]
|
|[nice, thank, you, seller, good, job, quality, is, good, the, best]
|[ADJ, PROPN, PRON, PROPN, PROPN, NOUN, NOUN, PROPN, ADJ, NOUN, ADJ]
|
|[recommend, beli, dkat, seller, ni, gaisek]
|[VERB, VERB, ADP, NOUN, DET, NOUN]
|
|[hope, seller, bole, check, condition, brg, dlu, sbelumpost]
|[PROPN, PROPN, ADV, VERB, NOUN, NOUN, ADJ, NOUN]
|
|[first, time, beli, kat, seller, ni]
|[ADV, ADV, VERB, ADP, NOUN, DET]
|
```

Figure 28 : Outcome of POS tagging for social english using malaya

6.3 POS Tagging Malay Sentences

The POS tagging for malay wiki and malay social media is the same as POS tagging the english data in social media, see here.

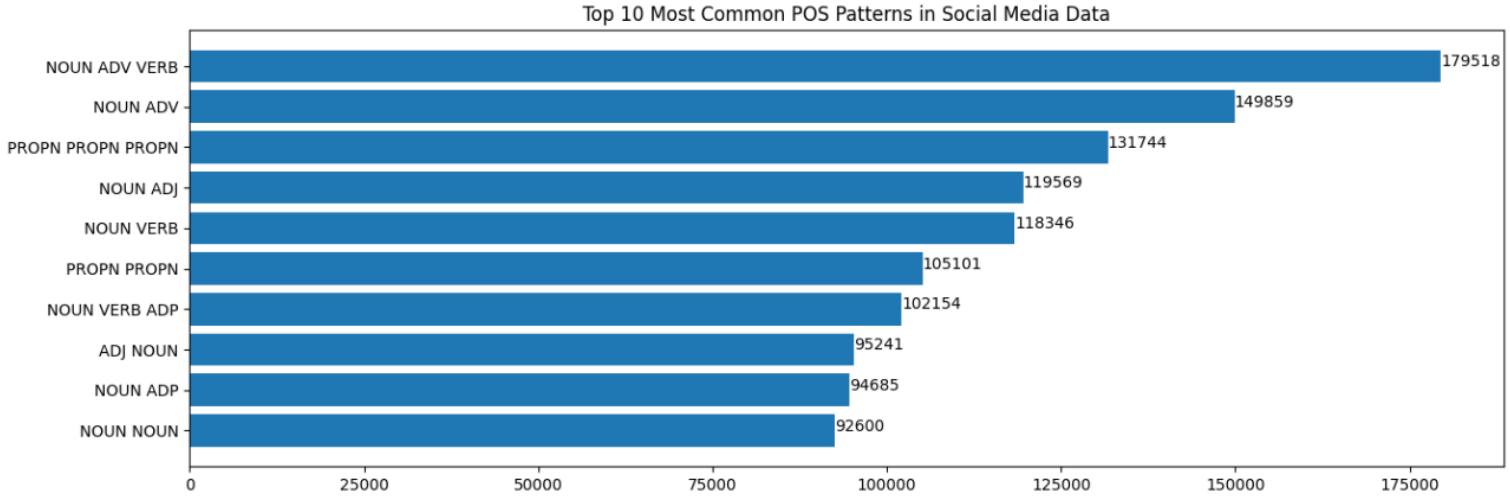
```
|
|[memang, beli, dgn, seller, ni, dah, lama, dah]
|[ADV, VERB, ADP, NOUN, NOUN, ADV, ADJ, ADV]
|
|[terima, kasih, seller, nanti, repeat, order, lgi]
|[NOUN, NOUN, NOUN, ADV, VERB, NOUN, ADV]
|
|[cuma, seller, lambat, sikit, prepare, utk, pos]
|[ADV, NOUN, ADJ, ADV, VERB, ADP, NOUN]
|
|[brg, sampai, dlm, baik, tp, seller, peram, ari, ls, oder, baru, pos, jnt, mmg, laju]
|[NOUN, VERB, ADP, ADJ, CCONJ, NOUN, PROPN, PROPN, PROPN, NOUN, ADJ, NOUN, NOUN, ADV, ADJ]
|
|[banyak, kali, dah, repeat, order, dengan, seller, ni]
|[ADV, NOUN, ADV, VERB, NOUN, ADP, NOUN, DET]
|
|[layanan, terbaikkk, respon, terbaikkk, tindakan, seller, terbaikkk, seller, x, layak, klu, bintang]
|[NOUN, ADJ, NOUN, ADJ, NOUN, NOUN, ADJ, NOUN, PART, ADJ, SCONJ, NOUN]
|
|[maaf, barang, sampai, awal, dah, cuma, lambat, review, penghantaran, laju, seller, pon, peramah, terima, kasih, terbaik, nanti, habis,
|[NOUN, NOUN, VERB, ADJ, ADV, ADV, ADJ, NOUN, NOUN, NOUN, NOUN, PART, NOUN, VERB, NOUN, ADJ, DET, ADJ, PRON, VERB]
|
|[terima, kasih, seller, dan, ninjavan]
|[NOUN, NOUN, NOUN, CCONJ, NOUN]
|
```

Figure 29 : Outcome of POS tagging for malay social media

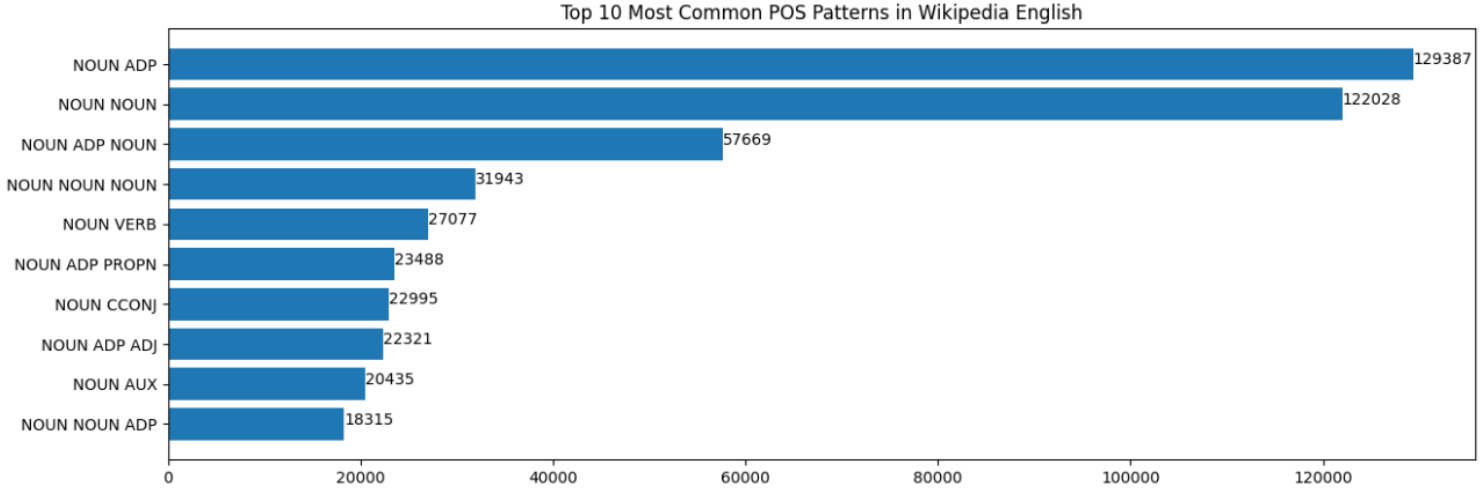
```
|
|[majhakot, merupakan, sebuah, kampung, yang, terletak, di, nepal]
|[NOUN, VERB, DET, NOUN, PRON, VERB, ADP, PROPN]
|
|[ialah, komun, di, jabatan, meurtheetmoselle, di, timurlaut, perancis]
|[AUX, NOUN, ADP, NOUN, NOUN, ADP, NOUN, NOUN]
|
|[gro, ronnau, merupakan, sebuah, kawasan, perbandaran, terletak, di, daerah, segeberg, schleswigholstein, jerman]
|[PROPN, NOUN, VERB, DET, NOUN, NOUN, VERB, ADP, NOUN, NOUN, PROPN, NOUN]
|
|[merupakan, sebuah, daerah, yang, terletak, di, wilayah, kazakhstan, utara, kazakhstan]
|[VERB, DET, NOUN, PRON, VERB, ADP, NOUN, PROPN, PROPN, PROPN]
|
|[kelantan, merupakan, sebuah, negeri, di, utara, malaysia, dengan, keluasan, lebih, kurang, km]
|[PROPN, VERB, DET, NOUN, ADP, NOUN, PROPN, ADP, NOUN, ADV, ADV, SYM]
|
|[cikareo, selatan, merupakan, sebuah, desa, yang, terletak, dalam, daerah, kecamatan, wado, kabupaten, sumedang, provinsi, jawa, barat, indonesia]
|[NOUN, NOUN, VERB, DET, NOUN, PRON, VERB, ADP, NOUN, NOUN, PROPN, NOUN, NOUN, PROPN, PROPN, PROPN, PROPN]
|
|[terdapat, buah, desa, di, dalam, daerah, kecamatan, air, rami]
|[VERB, NOUN, NOUN, ADP, ADP, NOUN, PROPN, NOUN, NOUN]
|
|[saintjeandetholome, ialah, komun, di, jabatan, hautesavoie, di, wilayah, rhone, alpes, di, perancis]
|[NOUN, AUX, NOUN, ADP, NOUN, NOUN, ADP, NOUN, PROPN, NOUN, ADP, NOUN]
|
|[terdapat, buah, desa, kelurahan, di, dalam, daerah, kecamatan, gianyar]
|[VERB, NOUN, NOUN, NOUN, ADP, ADP, NOUN, PROPN, PROPN]
|
```

Figure 30 : Outcome of POS tagging for malay wikipedia data

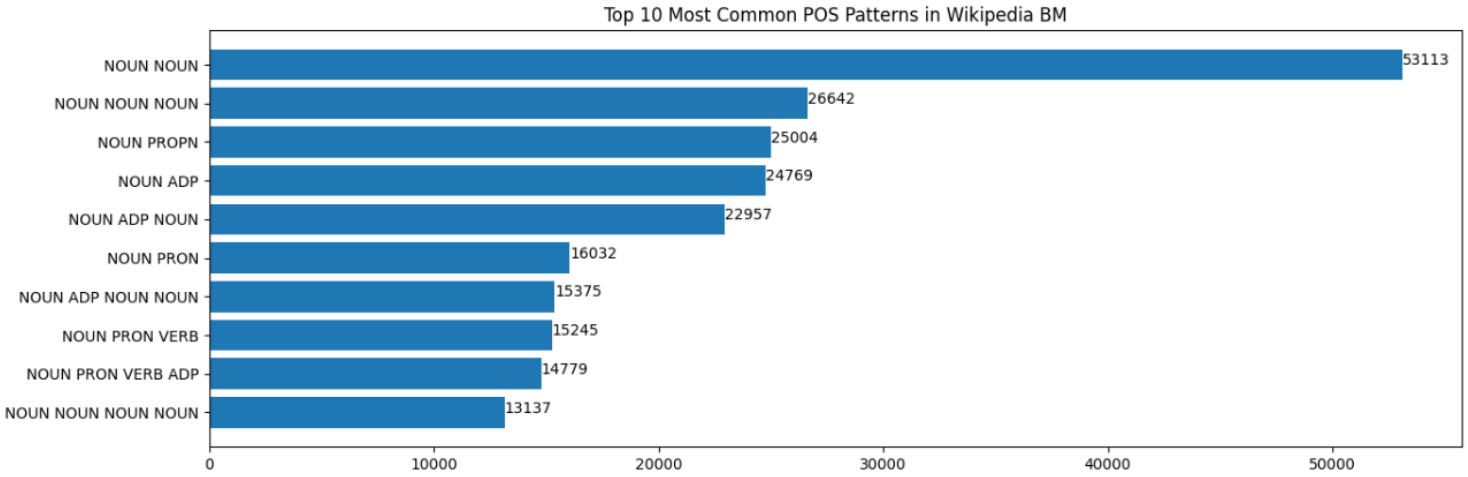
Social Media Corpus Results



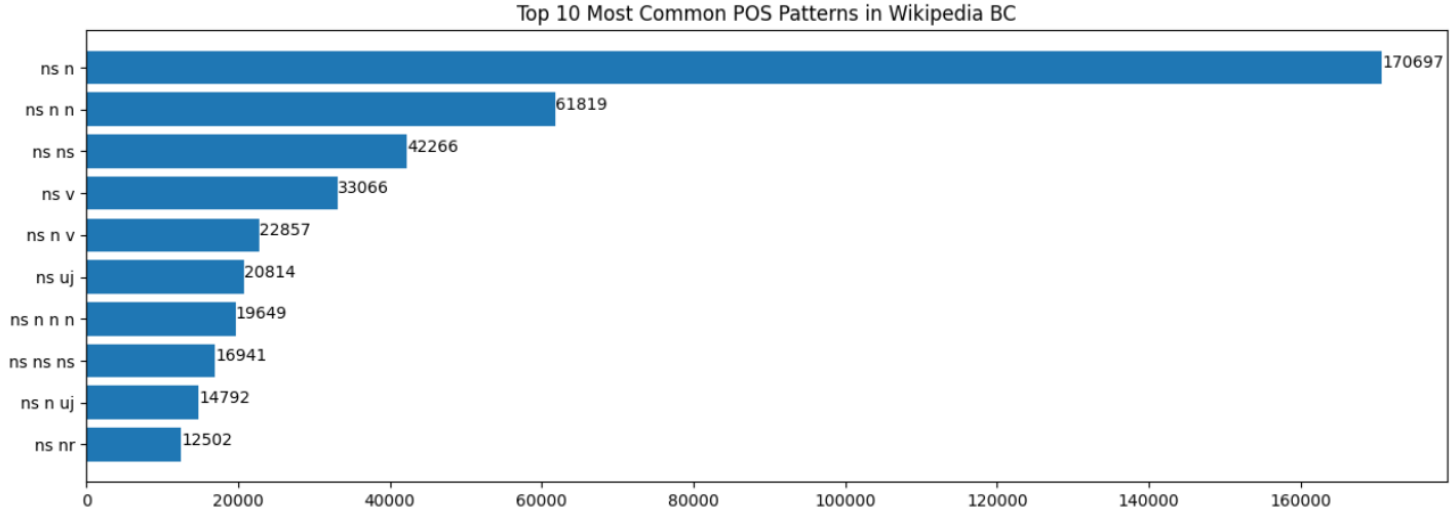
Wikipedia English Corpus Results



Wikipedia Melayu Corpus Results



Wikipedia Chinese Corpus Results



7.0 Result and Discussion

Task 1: Language Detection

Social Media Corpus Result

The most popular pattern that exists in sentences is the trigram “en en en” with 811,270 possible variations throughout the corpus, which is followed closely by the trigram “ms ms ms” with 788,054 variations. This indicates that in social media, there is still a large tendency to use a single language, most popularly English and Malay, in the majority of their sentences. The only time we see mixed language in this top 10 grouping is in the bigram pattern of “en ms” and trigram patterns of “en ms ms” and “en ms en”, indicating that in social media sentences, only minor parts of the sentence are in mixed language. This is further backed up by the five-gram patterns of “en en en en en” and “ms ms ms ms ms” which indicate the presence of a majority language in most social media sentences. Oddly, the trigram “zh zh zh” was much rarer, indicating that the use of the Chinese language may only belong to a niche group of social media users.

Task 2: POS Tagging

Social Media Corpus Result

The most popular pattern is the trigram “NOUN ADVerb VERB” with 179,518 possible variations throughout the corpus. This indicates at least some element of basic English grammar in a good amount of social media sentences with a combination of a noun, an adverb, and a verb in the correct order. Other patterns like the bigram “NOUN VERB” and “ADJective NOUN” also indicate some form of correct basic grammar, with a verb following a noun and an adjective in front of the noun being described. However, bigram patterns like “NOUN ADJective” and “NOUN ADPosition”, and trigram patterns like “NOUN VERB ADPosition” could indicate broken English grammar, but it could also be grammar in other languages like Malay, where the adjective follows the noun instead or just an incomplete POS pattern. Other patterns mostly consist of just general nouns and proper nouns, which could be indicative of a name consisting of multiple nouns or a lack of grammatical accuracy in sentences regardless of single or mixed languages.

Wikipedia English Corpus Result

The most popular pattern is the bigram “NOUN ADPosition” with 129,387 possible variations throughout the corpus, followed closely by the bigram “NOUN NOUN” with 122,028 possible variations. The random adposition followed by a noun may seem odd and could just be an incomplete POS pattern when displayed as a bigram, with the trigrams of “NOUN ADPosition NOUN”, “NOUN ADPosition PROPerNoun”, “NOUN ADPosition ADJective”, and “NOUN NOUN ADPosition” are a bit more complete. The remaining patterns such as the “NOUN VERB” and “NOUN AUXiliary” are just common patterns expected in a basic sentence.

Wikipedia Malay Corpus Results

The most popular pattern is the bigram “NOUN NOUN” with 53,113 possible variations throughout the corpus. This noun and noun combination resonates with most of the n-grams within the top 10 grouping, even up to the four-gram pattern with 13,137 possible variations, indicating the possibility of the corpus containing lots of names or the Malaya library tagging words incorrectly as nouns. The more complex patterns in this grouping are

the trigram “NOUN PRONoun VERB” and four-gram “NOUN PRONoun VERB ADPosition”, which were less frequent with 15,245 and 14,779 possible variations respectively.

Wikipedia Chinese Corpus Results

The most popular pattern is the bigram “ns n” with 170,697 possible variations throughout the corpus. This location name followed by a noun could indicate the description of an event involving an unspecified item at a location. This item could be an individual, with the presence of the bigram “ns nr”, indicating a human name following a location name. It can also be seen that “ns” tag exists throughout this grouping, which could indicate an intense reference to place names or just incorrect tagging by the Jieba library. This incorrect tagging could be backed up by the bigram “ns v”, which indicates a place name being followed by a verb, which is very odd since a place name is inanimate and would not be capable of actions. Some odd patterns like bigram “ns uj” and trigram “ns n uj” are hard to explain since the “uj” tag is not even mentioned within the Jieba library documentation.

Keyword +1, +2, -1, -2 analysis for wikipedia
The below data is to show the next word and previous word statistics.

[16]:
```
q("select distinct token_gram, tok_gram_f from wiki_gram where middle_key = T
```

```
+-----------------------------------------------+----------+
|token_gram                                     |tok_gram_f|
+-----------------------------------------------+----------+
|school elementary school elementary school     |100       |
|school district school data for                |38        |
|school primary school primary school           |27        |
|school middle school and high                  |24        |
|school middle school high school               |20        |
|university graduate school of journalism       |18        |
|college championship college football national|18        |
|college or university in the                   |17        |
|school high school high school                 |16        |
|university the university of california        |16        |
+-----------------------------------------------+----------+
only showing top 10 rows
```

[18]:
```
q("select distinct token_gram, tok_gram_f from wiki_gram where middle_key = T
```

```
+---------------------------------+----------+
|token_gram                       |tok_gram_f|
+---------------------------------+----------+
|kampung di daerah tutong brunei  |26        |
|daerah plzen utara wilayah plzen |7         |
|kampung di daerah an phu         |7         |
|sekolah bagi sekolah ini ialah   |6         |
|daerah daerah barat daya mukim   |6         |
|kampung raja daerah besut negeri |6         |
|utara mukim daerah seberang perai|4         |
|barat jawa barat kompas indonesia|4         |
|daerah kecil kampung gajah perak |4         |
|utara selari utara ke selari     |4         |
+---------------------------------+----------+
only showing top 10 rows
```

[19]:
```
q("select distinct token_gram, tok_gram_f from wiki_gram where middle_key = T
```

```
+------------------------+----------+
|token_gram              |tok_gram_f|
+------------------------+----------+
|中国 大陆 台湾 常 栖息  |256       |
|台湾 以及 中国 大陆 的  |62        |
|台湾 道 台湾 府 为      |52        |
|台湾 小吃 台湾 料理 台湾|43        |
|中国 大陆 香港 澳门 台湾|31        |
|台湾 府 台湾 县 知县    |29        |
|台湾 是 中国 的 一部分  |23        |
|台湾 称 中国 大陆 港澳  |22        |
|中国 大陆 香港 和 台湾  |18        |
|台湾 担任 台湾 府 台湾 |16        |
+------------------------+----------+
only showing top 10 rows
```

References

1. Kissanc (2021). *File:Data-pipeline.png*. Available at: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=File:Data-pipeline.png [Accessed: 1 October 2022].
2. Maheshkar, S. (2020). *Auto-Completion using N-Gram Models*. [online] Available at: https://www.kaggle.com/code/sauravmaheshkar/auto-completion-using-n-gram-models/notebook [Accessed: 1 October 2022]
3. Liling, T. (2019). *N-gram Language Model with NLTK*. [online] Available at: https://www.kaggle.com/code/alvations/n-gram-language-model-with-nltk [Accessed: 1 October 2022]
4. Tanya. (2020). *POS Tagging Using RNN*. [online] Available at: https://www.kaggle.com/code/tanyadayanand/pos-tagging-using-rnn [Accessed: 2 October 2022]
5. Khan, U.I. (2022). *Part Of Speech Tagging & Named Entity Recognition — NLP*. [online] Available at: https://towardsdev.com/part-of-speech-tagging-named-entity-recognition-nlp-aa12ca29dde4 [Accessed: 2 October 2022]
6. Johnson, D. (2022). *POS (Part-Of-Speech) Tagging & Chunking with NLTK*. [online] Available at: https://www.guru99.com/pos-tagging-chunking-nltk.html [Accessed: 2 October 2022]
7. Nabanita, R. (2020). *Building a Text Normalizer using NLTK ft. POS tagger.* [online] Available at: https://towardsdatascience.com/building-a-text-normalizer-using-nltk-ft-pos-tagger-se713e611db8 [Accessed: 3 October 2022]
8. Luck, C. (2018). *A Simple Solution to Run Multiple PySpark Instances on Multiple Jupyter Notebooks Simultaneously.* [online] Available at: https://medium.com/luckspark/a-simple-solution-to-run-multiple-pyspark-instances-on-multiple-jupyter-notebooks-simultaneously-3cdf28170ab9 [Accessed: 3 October. 2022]
9. Xiaoyuan, W. (2021) *Topic Modeling and Sentiment Analysis on Twitter Data Using Spark*. [online] Available at: https://towardsdatascience.com/topic-modeling-and-sentiment-analysis-on-twitter-data-using-spark-a145bfcc433 [Accessed: 4 October 2022]
10. Daniel, E. (2022) *Language Translation Using Python*. [online] Available at: https://towardsdatascience.com/language-translation-using-python-bd8020772ccc [Accessed: 3 October 2022]
11. Satyajit, M. (2019) *You Can Blend Apache Spark And Tensorflow To Build Potential Deep Learning Solutions*. [online] Available at: https://medium.com/@ssatyajitmaitra/you-can-blend-apache-spark-and-tensorflow-to-build-potential-deep-learning-solutions-9298e9fe8f6c [Accessed: 5 October 2022]
12. teavanist (2019) *Install Tensorflow (CPU) on Windows 10*. [online] Available at: https://medium.com/@teavanist/install-tensorflow-cpu-on-windows-10-4acbec6a71b7 [Accessed: 5 October 2022]
13. Fuentes, L.G. (2021) *Translation in Python, the sustainable way; Using deep-translator library*. [online] Available at: https://luis-alberto-g-efe.medium.com/translation-in-python-the-sustainable-way-using-deep-translator-library-b8dab9be25cf [Accessed: 5 October 2022]
14. Abhijith, C. (2021) *How to Add Text Labels to Scatterplot in Python (Matplotlib/Seaborn)*. [online] Available at:

https://towardsdatascience.com/how-to-add-text-labels-to-scatterplot-in-matplotlib-seaborn-ec5df6afed7a [Accessed: 6 October 2022]
15. Maria, K. (2019) *Adding sequential IDs to a Spark Dataframe*. [online] Available at: https://towardsdatascience.com/adding-sequential-ids-to-a-spark-dataframe-fa0df5566ff6 [Accessed: 7 October 2022]
16. Pinar, E. (2020) *PySpark and SparkSQL Basics*. [online] Available at: https://towardsdatascience.com/pyspark-and-sparksql-basics-6cb4bf967e53 [Accessed: 7 October 2022]
17. Charu, M. (2020) *Machine Learning Model deployment using Spark*. [online] Available at: https://towardsdatascience.com/machine-learning-model-deployment-using-spark-585e80b2eae1 [Accessed: 8 October 2022]
18. Sidath, A. (2020) *Machine Learning Classifiers*. [online] Available at: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623 [Accessed: 8 October 2022]
19. Timothy, T. (2020) *Creating Word Embeddings for Out-Of-Vocabulary (OOV) words such as Singlish*. [online] Available at: https://towardsdatascience.com/creating-word-embeddings-for-out-of-vocabulary-oov-words-such-as-singlish-3fe33083d466 [Accessed: 9 October 2022]
20. Harshith (2020) *Text Preprocessing in Natural Language Processing*. [online] Available at: https://towardsdatascience.com/text-preprocessing-in-natural-language-processing-using-python-6113ff5decd8 [Accessed: 10 October 2022]
21. Kajal, Y. (2021) *Cleaning & Preprocessing Text Data by Building NLP Pipeline*. [online] Available at: https://towardsdatascience.com/cleaning-preprocessing-text-data-by-building-nlp-pipeline-853148add68a [Accessed: 11 October 2022]
22. Sergei, I. (2019) *Left Join with Pandas Data Frames in Python*. [online] Available at: https://towardsdatascience.com/left-join-with-pandas-data-frames-in-python-c29c85089ba4 [Accessed: 11 October 2022]
23. Allison, S. (2020) *Natural Language Processing with PySpark and Spark-NLP*. [online] Available at: https://towardsdatascience.com/natural-language-processing-with-pyspark-and-spark-nlp-b5b29f8faba [Accessed: 11 October 2022]Had