**CE306 - Information Retrieval 2021**

**Assigment 1**

1702638

## Instruction

The menu.py programme pre-process the “Plot” values before indexing all 1000 document set to elastic search, with a custom elastic search mapping. The source data needs to be in the same folder as the python code.

After all document are being indexed to Elasticsearch a search window will pop up for user to search for specific document(Figure 2). Please do not close the programme or delete the “test.csv.temp” file before the popup message appear.(Figure1)

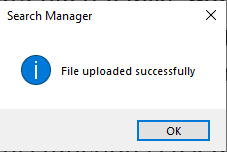


Figure 1

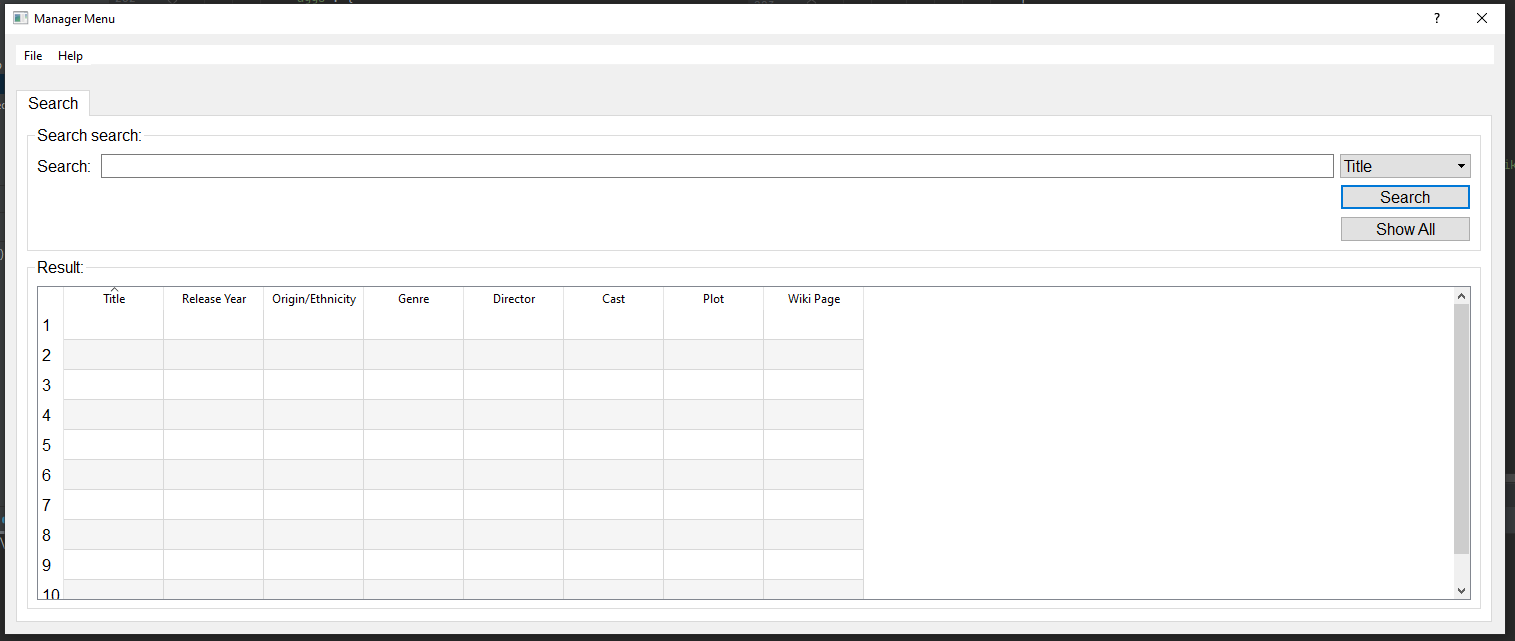


Figure 2 Search GUI

On the right-handside of the input box user can shoes which field/Column they want to search for(Figure 3).

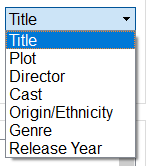


Figure 3 Drop down menu

The “Show All” button(Figure 4) will display all indexed documnent in the result table (Figure 5).

For Show All function, column sorting is enabled, user can click on the column header to change the sorting order.



Figure 4 Show All Button

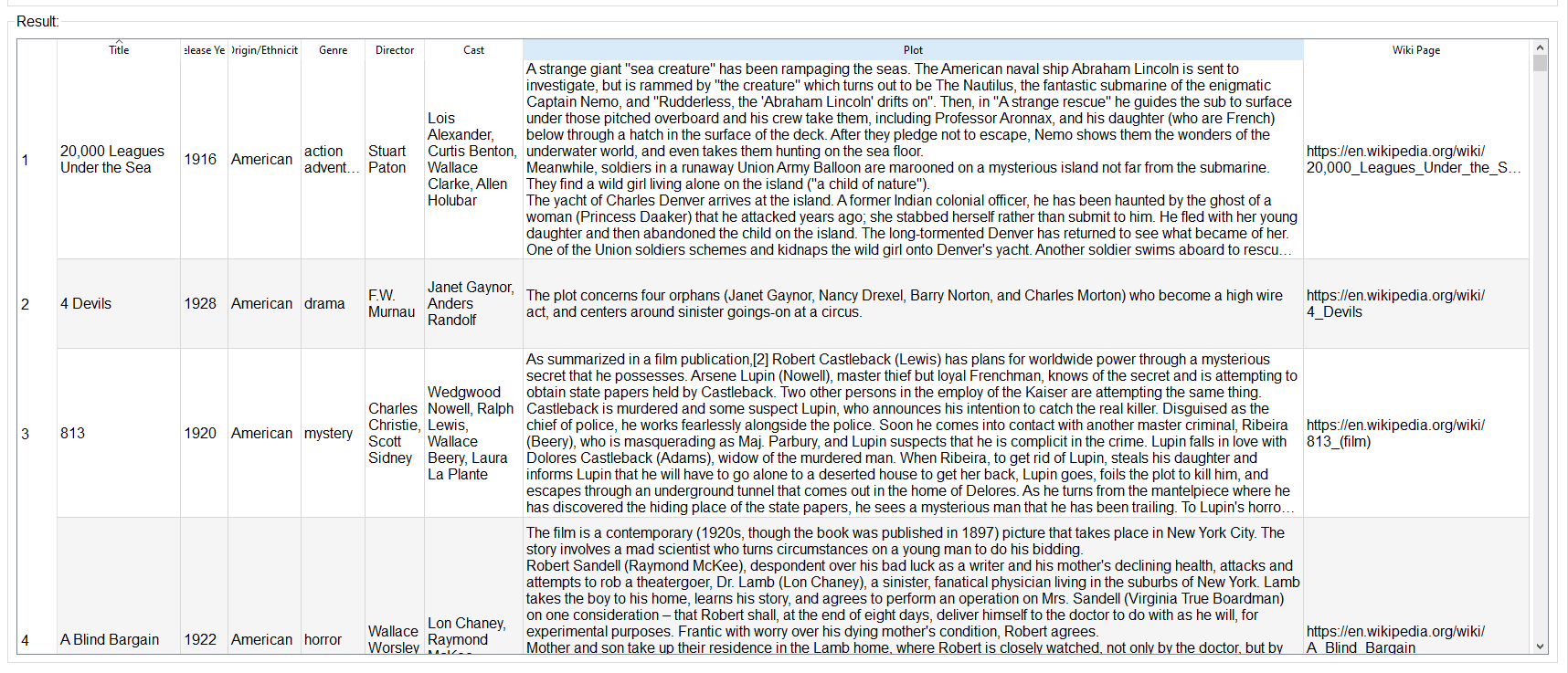


Figure 5 Show all

To perform normal search enter the key word in the input box, chose field/column name and hid the Search button. For search result the result table is order by most relevant movie to the least relevant movie. Unlike the show all result column sorting is disabled for keeping the original order.



Figure 6 Search button

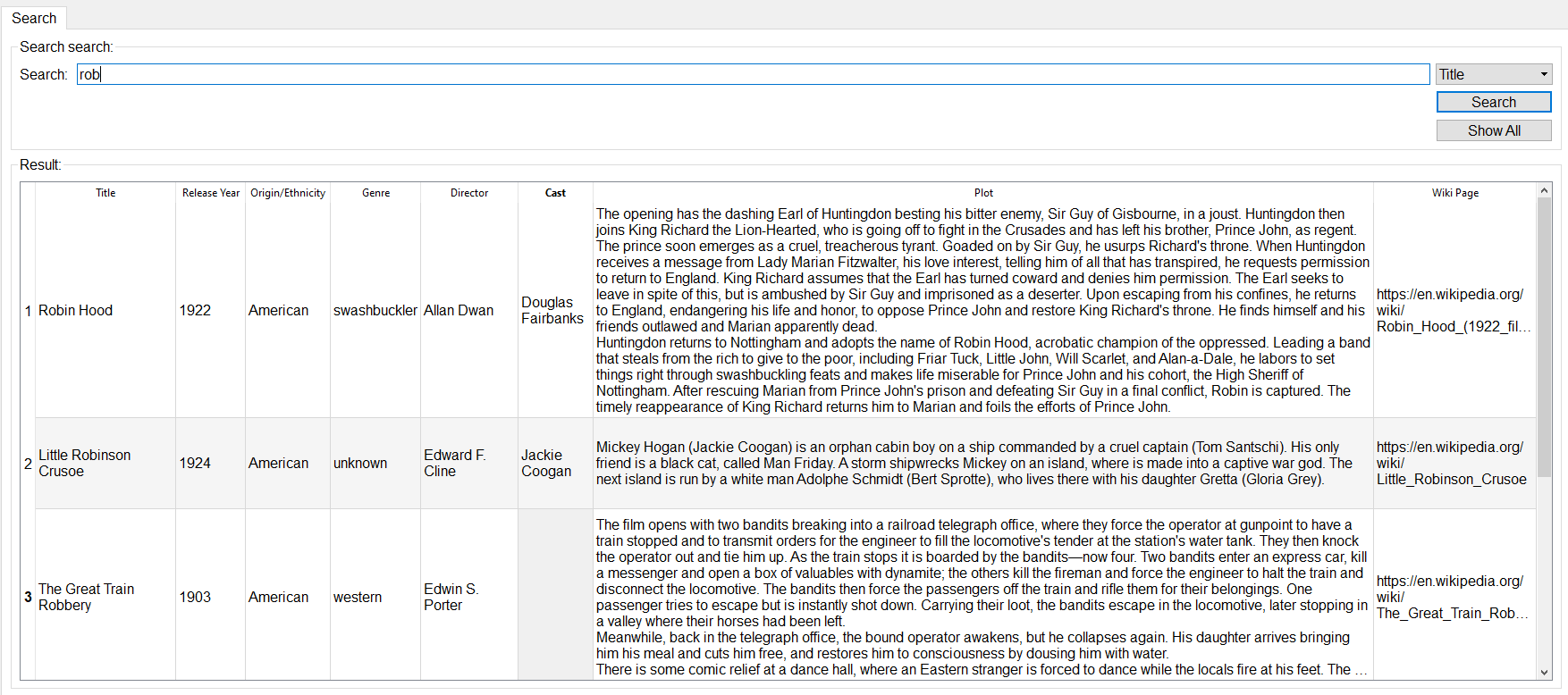


Figure 7 Search Result

## Core Program:

def indexing() – This function is used to do the pre-processing and indexing.

1. Indexing() import the original movie data file (i.e.wiki\_movie\_plots\_deduped.csv) to the program, and create a temporary list(temp\_list) to store 1000 data set plus CSV file header
2. Make a copy of the Plot data in a separate keyword list (temp\_key\_word)
3. Convert all text to lowercase and tokenise the plot paragraph
4. Remove all Stop word and filter out all words which are not useful using Part of Speech tagging
5. Lemmatised all keywords
6. Using TF-IDF scoring to remove words that are not useful, this steps will remove one-fith of the words with lowest TF-IDF score
7. Combine the processed keyword to the original CSV file
8. Create an index called “movies” with custom mapping setting and index the wiki movie documents to elastic search using “bluk” function

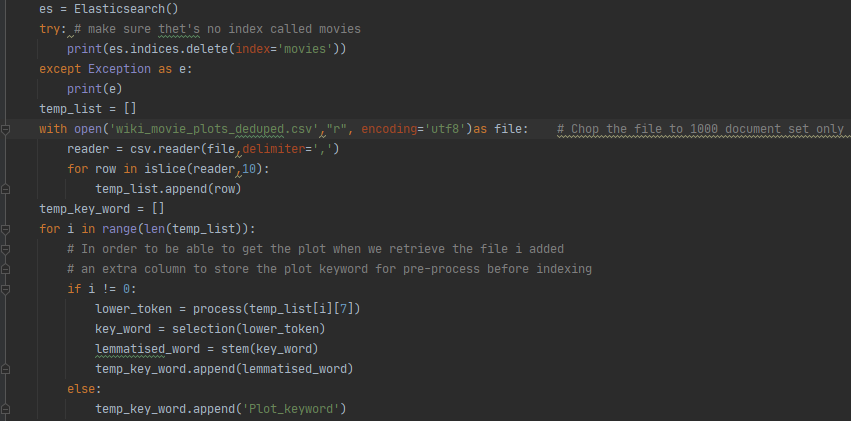
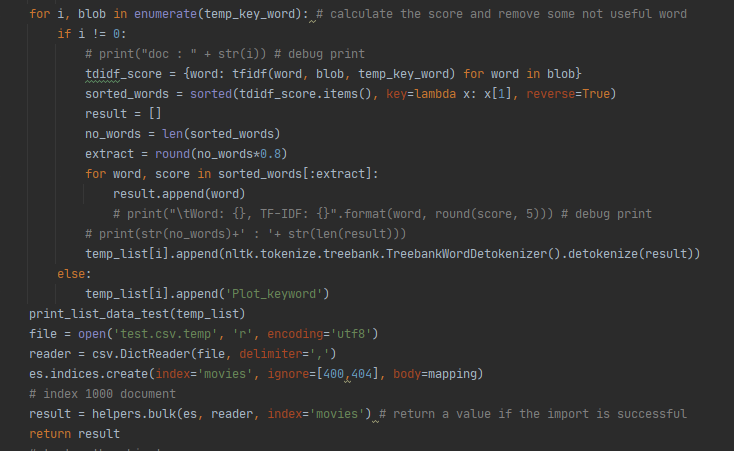
 

Figure 8 indexing()

def process(to\_process): -- This function is used to perform Sentence Splitting, Tokenization and Normalization

1. Takes a string as argument and perform normalization by converting all character to lowercase using lower()
2. Use NLTK word\_tokenizer function to convert the paragraph to token
3. Return a list with all words are separate list item

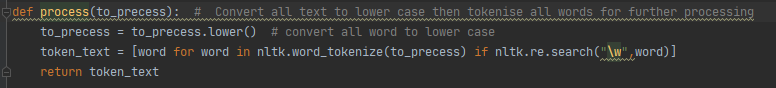


Figure 9 process()

def selection(tagged\_word): -- This functionis use to filter out some words which are not useful in distinguishing between relevant/non-relevant documents

1. Selection takes a list as argument and use NLTK stopwords list to remove words which are not useful, for example have, too, such, didn't, why, now
2. Using the NLTK Part-of-speech Tagging, which use to compute how informative a word is by catagorise a word in different categories. For this occation we want to extract only Noun, Verb, Participle, Adjective

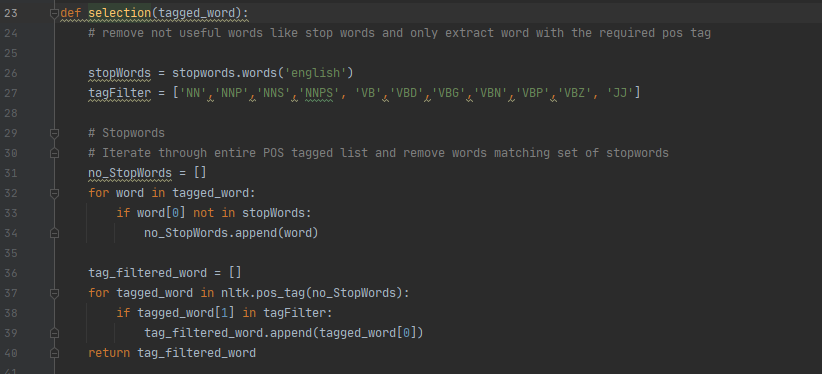


Figure 10 selection()

def stem(key\_word): -- This function is use for reducing inflected words to their word stem, so that a word in various inflected forms will not be treated as different word. For this task I chose to use the lemmatize function instead of using the porterstemming function. Stemming usually based on heuristics, thus it commonly suffer from overstemming or understemming for example “University and universe”. To get better result I use lemmatize which uses wordnet to get better result for these wiki movie data, lemmatize not only reducing inflected words to their word stem but also morphologically changes them.

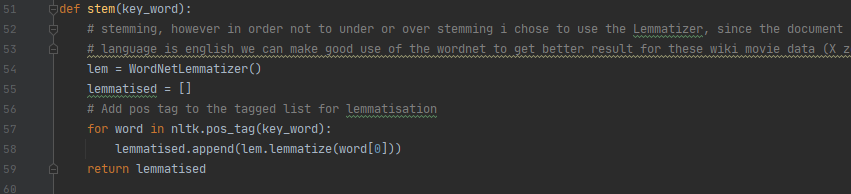


Figure 11 stem()

def tfidf(word,document,wordlist): -- This function is used to calculate the TFIDF for each word in each document. I perform this after so that that a word lemmatize in various inflected forms will not be treated as different word

1. Calculate the occurrence of a keyword in a document, then divide it by the number of keyword words in a document
2. Calculate the inverse document frequency, the more common word the lower the value is (Find the retio of number document to the number of document containing keyword, then take a log of the value )
3. Calculate the tfidf score and return the word list with the score back to the main method

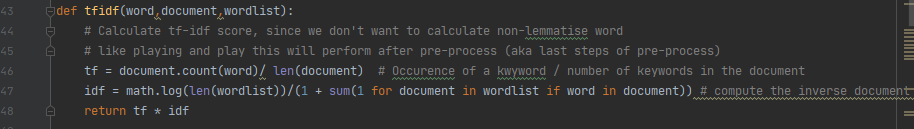


Figure 12 tfidf()

## Custom mapping:

mapping = {  
 "settings": {  
 "number\_of\_shards": 2,  
 "number\_of\_replicas": 1,  
 "index":{  
 "analysis":{  
 "tokenizer": {  
 "comma": {  
 "type": "pattern",  
 "pattern": ","  
 }  
 },  
 "analyzer":{  
 "normal\_analyzer":{  
 "tokenizer":"standard",  
 "filter":"lowercase",  
 },  
 "url\_token": {  
 "tokenizer": "uax\_url\_email"  
 },  
 "comma": {  
 "type": "custom",  
 "tokenizer": "comma"  
 }  
 }  
 }  
 }  
 },  
 "mappings": {  
 "properties": {  
 "Cast": {  
 "type": "text",  
 "analyzer": "comma",  
 "search\_analyzer": "normal\_analyzer"  
 },  
 "Director": {  
 "type": "text",  
 "analyzer": "comma",  
 "search\_analyzer": "normal\_analyzer"  
 },  
 "Genre": {  
 "type": "keyword"  
 },  
 "Origin/Ethnicity": {  
 "type": "keyword"  
 },  
 "Plot": {  
 "type": "text"  
 },  
 "Plot\_keyword" : {  
 "type": "text",  
 "analyzer":"normal\_analyzer"  
 },  
 "Release Year": {  
 "type": "integer"  
 },  
 "Title": {  
 "type": "text"  
 },  
 "Wiki Page": {  
 "type": "text",  
 "analyzer": "url\_token"  
 }  
 }  
 }

}

In order to get the best result, I use a custom mapping setting to specify the analyser, tokeniser and the data type. For the default mapping, elastic search will mis under stand the data type as Keyword, which will affect the quality of the result, in the setting I specified the analyser (how elastic tokenise the document after indexing), the search\_analyser(how elastic search tokenise the search keyword), also I specified the specific tokeniser for specific field, for example I specified the wiki page field using “uax\_url\_email” which is a special tokeniser for URL and email address.

## Searching

GET /movies/\_search{

"\_source": ["Title","Release Year","Genre","Director","Origin/Ethnicity","Plot","Wiki Page","Plot\_keyword","Cast"],

"size": 20,

"min\_score": 0.5,

"query": {

"bool": {

"must": [{

"match\_phrase\_prefix": {

"Genre": {

"query": "{}".format(str(self.SearchEdit.text()))

}

}

}]

}

},

"aggs": {

"auto\_complete": {

"terms": {

"field": "title.keyword",

"order": {

"\_count": "desc"

},

"size": 25

}

}

}

})

In the search manager, most of the search query are based on the above query.

"min\_score": 0.5 🡪 Showing only the most relevant result

"match\_phrase\_prefix" 🡪 Making sure the return document is in the same order as the search phrase

Terms Aggregation 🡪 Generate buckets for each of the documents and place all of the results separately.

Descending order 🡪 Show the most relevant result first

## Discussion

For this assignment, I chose to use the basic python elasticsearch client api to interact with the elasticsearch engine. Before indexing I also use NLTK library to filter for useful keyword in distinguishing between relevant/non-relevant documents, for example, changing ‘working’, ‘worked’ to ‘work’ but also make sure we don’t under or over stemming words. I also created a userfriendly GUI with PyQt5 with a preset search query for users to search for their movie more easily.

Regarding possible improvements and extensions, since we are only using the basic elasticsearch library and there are a lot of limitation on creating search query and mapper setting. To make the search more flexible switching to Elasticsearch DSL, a high-level library which aim to help with writing and generating running queries, should make the search GUI more userfriendly and flexible. Apart from searching query, changing the order of each pre-processing steps and setting or using a different library (for example BM25) might help improve the search result.