Building a Marvel Search Engine

Emilio Ramos Monzalvo

Sharat Sripada

Joshua R Biggs-Bauer

IST 736 – Text Mining - Dr. Bolton

Fall 2021

Code: github.com/ramosem97/mcu_marvel_search_engine

Table of Contents

I. Introduction

I.A. Problem Statement

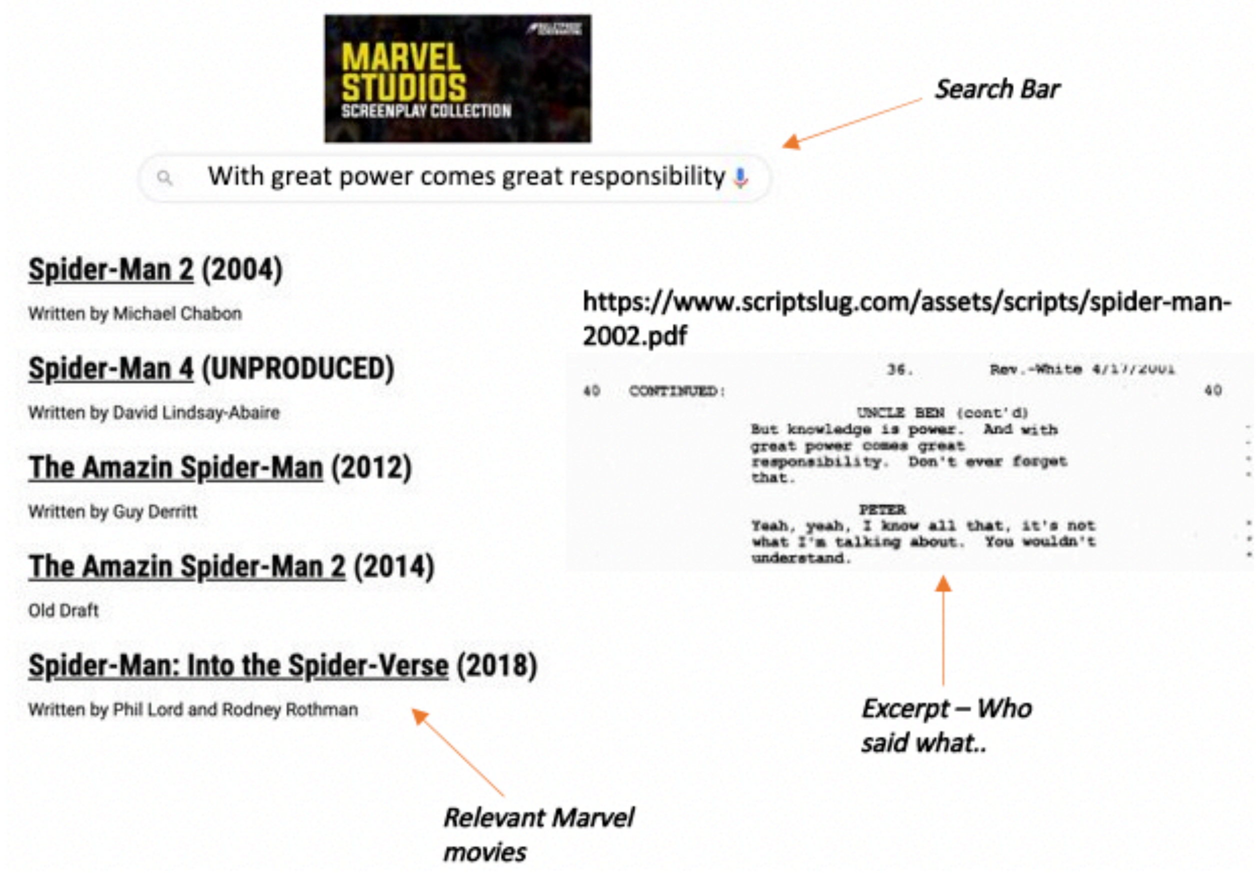
The MCU's phases one through three consists of twenty-three movies which adds up to more than three thousand minutes of viewing. This, of course, makes it challenging for someone that wants to watch the historically relevant movies when a new one for phase four is about to come out at the cinema. If one had no previous knowledge of what movies might be relevant, there would be no choice but to view all three thousand minutes of these first three phases. Phase four of the MCU is going to consist of twelve movies and thirteen tv shows, so even further into the future, getting up to date with the MCU will get exponentially harder with so many movies coming out.

I.B. Proposed Solution

Since the MCU's phase one through three have concluded, the approach would depend on using these movie scripts to build a large text corpus that can be used as a training dataset to find the most relevant topic a MCU fan is looking for. In other words, the movie scripts will be used to build a search engine like Google or Bing where one can query relevant dialogues, characters, and movies based on a simple phrase as the input. For example, if the new Spiderman movie is coming out, one can search the character "spiderman", and the search engine would return movies where the character was in like Spider-Man: Homecoming and dialog like: "PETER PARKER: Call me Spider-Man".

The goal is that this will assist those trying to reference the first three phases or get more familiar with the MCU without having to watch 3000 minutes of energetic or dull movies. Moreover, Marvel experts can also use this tool to find 'easter eggs' hidden in the dialog of the movies whether it is a reference to a known city associated with a character yet to appear again or better explanations of scenes that can be better grasped through the movie script. The Marvel Search Engine will be meant for all levels of Marvel Fandom. A prototype of the proposed solution can be seen on Image 1.

[**Image 1**: Proposed Solution Prototype]


MARVEL STUDIOS
SCREENPLAY COLLECTION
Search Bar
With great power comes great responsibility
Relevant Marvel movies
Excerpt – Who said what..

Spider-Man 2 (2004)
Written by Michael Chabon

Spider-Man 4 (UNPRODUCED)
Written by David Lindsay-Abaire

The Amazin Spider-Man (2012)
Written by Guy Derritt

The Amazin Spider-Man 2 (2014)
Old Draft

Spider-Man: Into the Spider-Verse (2018)
Written by Phil Lord and Rodney Rothman

https://www.scriptslug.com/assets/scripts/spider-man-2002.pdf

```
                          36.        Rev.-White 4/17/2001
40   CONTINUED:                                           40
                       UNCLE BEN (cont'd)
             But knowledge is power.  And with
             great power comes great
             responsibility.  Don't ever forget
             that.

                         PETER
             Yeah, yeah, I know all that, it's not
             what I'm talking about.  You wouldn't
             understand.
```

I.C. Tools for Solution

The use of movie scripts requires for extensive and complex Text Mining techniques used to extract information and standardize the text to train a Machine Learning algorithm used for Search Engines. The approach can be broken up into five parts: PDF to Text, Relevant Text Extraction using Regular Expressions, Text Standardization, Search Engine Modeling using BM25 and Google's Universal Sentence Encoder (USE), and Dashboarding using the python library Dash.

To better serve the user, only the best performing algorithms on the Text Standardization and the Search Engine Modeling parts will be kept and used in the final presentation of the Marvel Search Engine. The algorithms on each part of the approach fully depend on one another, so it will also be crucial to create a working pipeline that can be modified for newer movies as the MCU's movie catalog will keep increasing.

II. Analysis

II.A. About the Data

The data collected for this project came from multiple sources. Naturally, there were inconsistent file formats when pulling data from multiple sources. This piece had to be resolved before any of the rest of the project moved forward to standardize all the scripts into one model-friendly format. When scripts are published, they are more likely to come as PDF's, but there already existed a public repository [Marvel-Dialog-NLP] with the PDF's converted into a text format for 70% or 16 out of 23 of the movie scripts. The remaining scripts were taken directly as PDFs from a movie script website [MCU-Script-PDF] which had to go through additional processing steps as the other movies. The breakdown of the scripts can be seen in Table 1.

[**Table 1**: MCU Phase 1 through 3 Movies and Format]

Movie Title	Release Year	MCU Phase	Format
Iron Man	2008	Phase 1	Text
The Incredible Hulk	2008	Phase 1	PDF
Iron Man 2	2010	Phase 1	Text
Captain America: The First Avenger	2011	Phase 1	Text
Thor	2011	Phase 1	Text
The Avengers	2012	Phase 1	Text

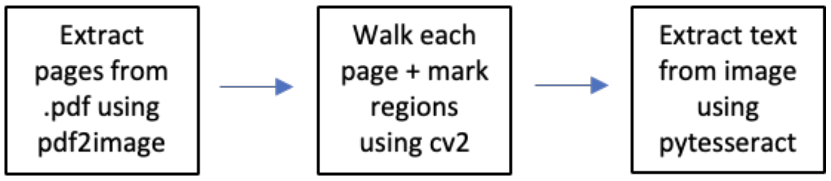
Iron Man 3	2013	Phase 2	Text
Thor: The Dark World	2013	Phase 2	Text
Captain America: The Winter Soldier	2014	Phase 2	Text
Guardians of the Galaxy	2014	Phase 2	PDF
Ant-Man	2015	Phase 2	Text
Avengers: Age of Ultron	2015	Phase 3	Text
Captain America: Civil War	2016	Phase 3	Text
Doctor Strange	2016	Phase 3	PDF
Spider-Man: Homecoming	2017	Phase 3	Text
Thor: Ragnarök	2017	Phase 3	Text
Guardians of the Galaxy Vol. 2	2017	Phase 3	PDF
Avengers: Infinity War	2018	Phase 3	Text
Ant-Man and the Wasp	2018	Phase 3	PDF
Black Panther	2018	Phase 3	PDF
Avengers: Endgame	2019	Phase 3	Text
Captain Marvel	2019	Phase 3	Text
Spider-Man: Far From Home	2019	Phase 3	PDF

II.B. PDF to Text

One of the preliminary challenges to solve was to identify source(s) of scripts and translate them to text, so they could be aggregated into data-frames and eventually mined using text analytics.

Most scripts were available in .pdf format (but some scanned pdf) and so the following pipeline of steps was setup using python libraries/modules - pdf2image, cv2 and pytesseract:

[**Flow-Diagram1**: Pipelines showing text processing]


Extract pages from .pdf using pdf2image
Walk each page + mark regions using cv2
Extract text from image using pytesseract

The process can be broadly thought of as a function of Optical Character Recognition (OCR). The final step is to pass the extracted text through a text sanitizer to remove unwanted strings, metadata-like characters and stop-words (imported from nltk.corpus for English).

II.C. Character Line Extraction

When a writer creates a movie script, they need to include four different types of commentary. Firstly, the general narration of the setting the scene is taking place at. This type of narration starts off the scene by introducing the characters and what event is taking place. A good example comes from the movie *Guardians of the Galaxy* where the first lines on the script are narration as seen in Quote 1. The example clearly introduces Peter Quill and his location and his current action.

[**Quote 1**: Guardians of the Galaxy First Lines]

> First lines. Earth 1988. Young Peter Quill sits in the waiting room of a hospital, listening to t
> he "Awesome Mix" tape on his Walkman when his grandfather comes over to him.

The second, third and fourth type of commentary follows the character's dialog. T
he dialog is broken up into three parts, the first being the character's name, the dialog, and the narration of the character while speaking. The character's name can be easily identified by the way it is set apart from the rest of the text. In most cases, the name is written in the s
tyle of a title followed by a colon, the dialog. The other common setting makes the name be in all capital letters and the dialog follows it in the next line. The two examples can be seen i
n Quote 2 and Quote 3. Quote 2 can then be broken down as the character being 'Scott', the dialog as 'Wow. That is super cool. Come on.', and the commentary is clearly inside the brac
kets.

[**Quote 2**: Dialog from Ant-Man and the Wasp]

> Scott: Wow. That is super cool. Come on. [Scott and Cassie crawl through the cardboard maz
> e; a fake ant face shows up]

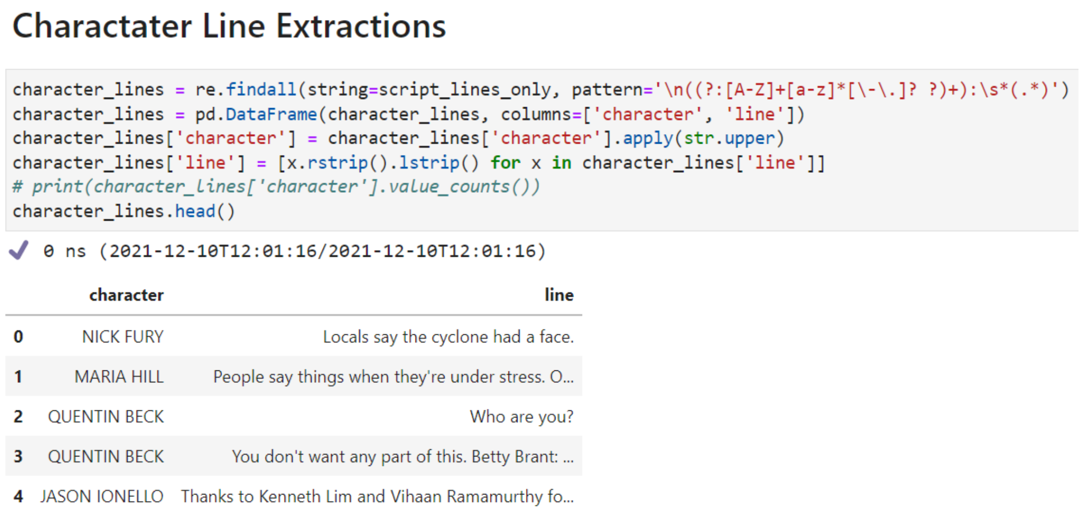
[**Quote 3**: Dialog from Dr. Strange]

> DOCTOR STRANGE

It’s amazing you kept him alive. Apneic, further brain stem testing after reflex test… I think I found th e problem, Dr. Palmer. You left a bullet in his head.

These patterns then make it straightforward to extract the dialog for each of the scripts in the MCU using regular expressions (regex). Regex’s ability for pattern recognition does not only return the text that includes dialog only, but it can also be used to return each part of the character’s dialog. In Code 1, the sample code for extracting the character and corresponding dialog is shown for the movie *Spider-Man: Far From Home*. The text is processed and then a table is outputted with every row representing a single line from the script with two columns containing the character’s name in the first one and the corresponding dialog in the second one. Through the dialog extraction, the only movie script that was unable to be cleaned was *The Incredible Hulk* due to the inability to account for tabs when converting a PDF to a Text file. Tabs were a crucial identifier of dialog, but the PDF to Text package would not consider it different from regular whitespace.

[**Code 1**: Regex for character dialog in Spider-Man: Far From Home]



Charactater Line Extractions

```
character_lines = re.findall(string=script_lines_only, pattern='\n((?:[A-Z]+[a-z]*[\-\.]? ?)+):\s*(.*)')
character_lines = pd.DataFrame(character_lines, columns=['character', 'line'])
character_lines['character'] = character_lines['character'].apply(str.upper)
character_lines['line'] = [x.rstrip().lstrip() for x in character_lines['line']]
# print(character_lines['character'].value_counts())
character_lines.head()
```

```
✓ 0 ns (2021-12-10T12:01:16/2021-12-10T12:01:16)
```

	character	line
0	NICK FURY	Locals say the cyclone had a face.
1	MARIA HILL	People say things when they're under stress. O...
2	QUENTIN BECK	Who are you?
3	QUENTIN BECK	You don't want any part of this. Betty Brant: ...
4	JASON IONELLO	Thanks to Kenneth Lim and Vihaan Ramamurthy fo...

II.E. Character Name Variations

Due to multiple processing steps used to extract text from a movie script and characters appearing on multiple movies, there appeared to be some discrepancies on the naming of the characters. In the movie Ant-Man, the main character ‘Scott Lang’, but his superhero name is ‘Ant-Man’. This means that when the mask comes on, the script switches the character’s real name into his superhero persona. Other examples of this come from simple name variations like removing the last name or punctuation like in the movie Doctor Strange where Dr. Stephen Strange can be referred to as ‘Dr. Strange’, ‘Doctor Strange’, ‘Stephen Strange’, ‘Stephen’, or ‘Dr. Stephen Strange’.

Within a single movie script, it is simpler to standardize the character name variation, but when aggregating the 23 movies there are up to 754 unique character names. Comparing names manually is not an option to find similar names in the list, so fuzzy matching was used to score each name against each other by computing the Levenshtein Distance. Once the results were computed, the names with matches were manually checked against each other to see if the character was indeed the same one.

[**Table 2**: Fuzzy Matching Score]

Character	**Matches: Score**
TONY STARK	{'A1 TONY': 100, 'HOWARD STARK': 100, 'MARIA STARK': 100, 'MORGAN STARK': 100, 'STARK’S ENGINEER…
STEVE ROGERS	{'A1 STEVE': 100}
THOR	{'A1 THOR': 100, 'ACTOR THOR': 100}
PETER PARKER	{'PETER QUILL': 100, 'TOWNIE PETE': 89}
NATASHA ROMANOFF	{'A1 NATASHA': 100}
PETER QUILL	{'MEREDITH QUILL': 100, 'PETER PARKER': 100, 'QUILL': 100, 'TOWNIE PETE': 89, 'YOUNG QUILL': 100}
PEPPER POTTS	{"PEPPER'S ASSISTANT": 86}
DOCTOR STRANGE	{'4F DOCTOR': 100, 'DOCTOR': 100, 'DOCTOR WEST': 100, 'OBADIAH STANE': 83, 'YOUNG DOCTOR': 100}
LOKI	{'ACTOR LOKI': 100}
SCOTT LANG	{'CASSIE LANG': 100, 'MAGGIE LANG': 100, 'SCOTT': 100}
JAMES RHODES	{'JAMES': 100, 'JAMES "JIM" PAXTON': 100, 'JAMES MONTGOMERY FALSWORTH': 100}
GAMORA	{'YOUNG GAMORA': 100}
JANE FOSTER	{'JAN': 86, 'JANET': 89}
T'CHALLA	{"T' CHALLA": 100}
QUILL	{'MEREDITH QUILL': 100, 'PETER QUILL': 100, 'YOUNG QUILL': 100}
HAPPY HOGAN	{'HAPPY': 100, "HAPPY'S NURSE": 83}
CLINT BARTON	{'BARON WOLFGANG VON STRUCKER': 91, 'LAURA BARTON': 100, 'LILA BARTON': 100}
SAM WILSON	{'COACH WILSON': 100}
HANK PYM	{'HANK': 100, 'PYM TECH SECURITY GUARD': 100}

III. Models

III.A. Best Match (BM) - 25

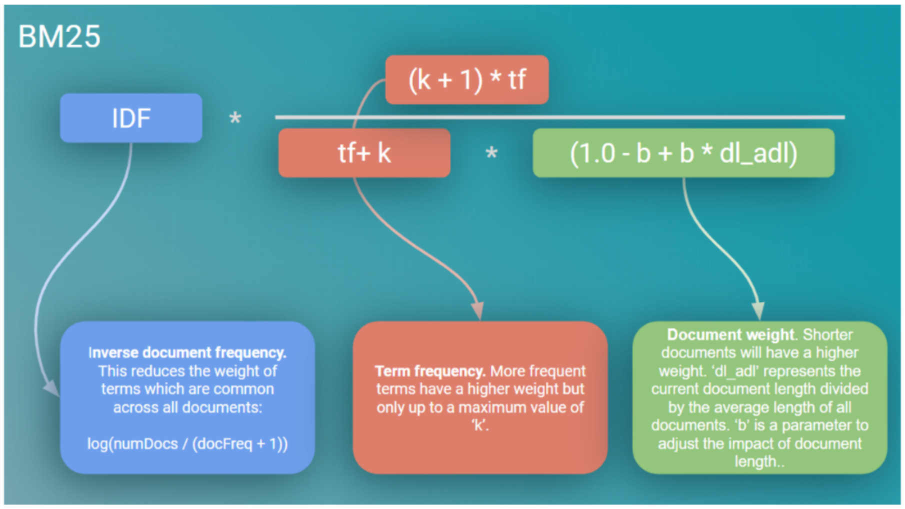
For a long time, Elastic search backed by simple term-frequency and inverse document frequency (TF-IDF) was the norm. More recent search engines like Azure Cognitive Search however have switched to BM-25. BM-25 is essentially a bag-of-words retrieval function that ranks a set of documents based on query terms appearing in each document, regardless of the proximity within the document. It can be thought of as an improved version of the TF-IDF implementing the following two key refinements:

- **Term Frequency saturation:** BM25 provides diminishing returns for the number of terms matched against documents. If one is looking to search for a specific term which is very common in documents, then there would come a point where the number of occurrences of this term becomes less useful to the search

- **Document length:** BM25 considers document length in the matching process. If a shorter article contains the same number of terms that match as a longer article, then the shorter article is likely to be more relevant.

As a result, BM-25 can be mathematically represented as below - importantly introducing two tunable hyper-parameters k and b to adjust term frequency saturation and document length:

[**Image 2**: BM25 Formula and Procedure]


BM25
IDF
*
(k + 1) * tf
tf+ k
*
(1.0 - b + b * dl_adl)
Inverse document frequency. This reduces the weight of terms which are common across all documents:
log(numDocs / (docFreq + 1))
Term frequency. More frequent terms have a higher weight but only up to a maximum value of 'k'.
Document weight. Shorter documents will have a higher weight. 'dl_adl' represents the current document length divided by the average length of all documents. 'b' is a parameter to adjust the impact of document length..

Working of BM-25 can further be illustrated through an example:

[**Example 1:** President Lincoln BM25]

- Consider the following query - 'President Lincoln' as the search query for a document set comprising 500,000 documents.
  - 'President' occurs in say 40,000 documents
  - 'Lincoln' occurs in say 300 documents
  - Given a document D, say 'President' occurs 15 times and 'Lincoln' occurs 25 times
  - K1 = 1.2, b = 0.75 (set empirically) -> K = 1.11 (if dl_adl = 0.9)
- Finally, BM25 for word 'President' and 'Lincoln) for document D would be 20.66
- Similarly, for a few more documents we derive some BM-25 as below:

[**Table 3:** President Lincoln BM25 Frequency]

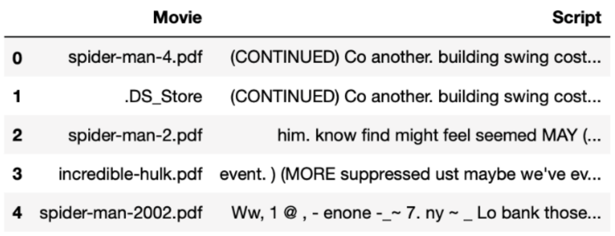
Frequent of 'President'	**Frequency of 'Lincoln'**	**BM-25 score**
15	25	20.66
15	1	12.74
15	0	15
1	25	18.2
0	25	15.66

- Table 3 demonstrates that it is possible for a document containing many occurrences of a single important term to score higher than a document containing both query terms (15.66 versus 12.74)

III.A.1 Model and results

Implementing BM-25 is incredibly simple owing to availability of library *rank-bm25.* Once the data was sufficiently translated and cleaned, it is aggregated into a data-frame as below:

[**Image 3**: Result data-frame for BM-25]

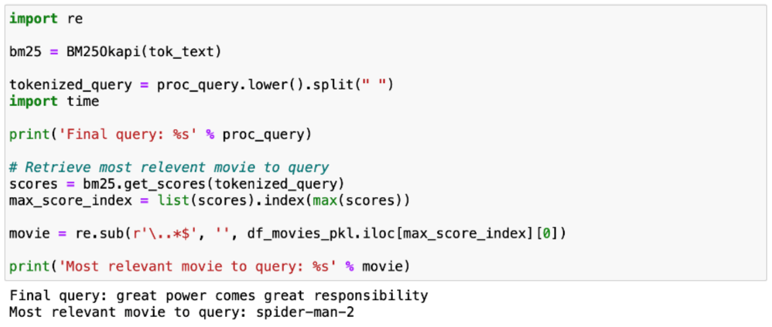


	Movie	Script
0	spider-man-4.pdf	(CONTINUED) Co another. building swing cost...
1	.DS_Store	(CONTINUED) Co another. building swing cost...
2	spider-man-2.pdf	him. know find might feel seemed MAY (...
3	incredible-hulk.pdf	event.) (MORE suppressed ust maybe we've ev...
4	spider-man-2002.pdf	Ww, 1 @ , - enone -_~ 7. ny ~ _ Lo bank those...

Text in the Script column is passed through the tokenize function in *SpaCy.* The resulting data-structure of list of lists is passed to bm-25 to extract two results namely:

- Given a search query locate the movie using the *bm-25.get_scores()* method

[**Code 2**: bm-25 in action using rank-bm25 library]

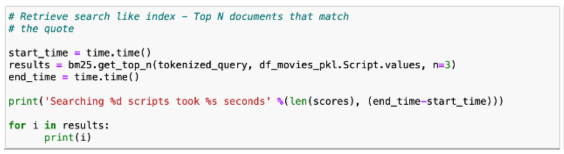


```
import re

bm25 = BM25Okapi(tok_text)

tokenized_query = proc_query.lower().split(" ")
import time

print('Final query: %s' % proc_query)

# Retrieve most relevent movie to query
scores = bm25.get_scores(tokenized_query)
max_score_index = list(scores).index(max(scores))

movie = re.sub(r'\..*$', '', df_movies_pkl.iloc[max_score_index][0])

print('Most relevant movie to query: %s' % movie)
```

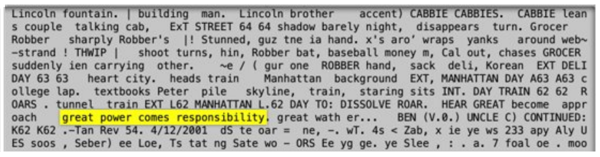
```
Final query: great power comes great responsibility
Most relevant movie to query: spider-man-2
```

- Given a search query locate relevant texts comprising excerpts or parts of speech using the *bm-25.get_top_n()* method

[**Code 3**: bm-25 get_top_n method and results]



```python
# Retrieve search like index - Top N documents that match
# the quote

start_time = time.time()
results = bm25.get_top_n(tokenized_query, df_movies_pkl.Script.values, n=3)
end_time = time.time()

print('Searching %d scripts took %s seconds' %(len(scores), (end_time-start_time)))

for i in results:
      print(i)
```



```
Lincoln fountain. | building  man.  Lincoln brother    accent) CABBIE CABBIES.  CABBIE lean
s couple  talking cab,   ExT STREET 64 64 shadow barely night,  disappears  turn. Grocer
Robber   sharply Robber's  |! Stunned, guz tne ia hand. x's aro' wraps  yanks   around web~
-strand ! THWIP |   shoot turns, hin, Robber bat, baseball money m, Cal out, chases GROCER
suddenly ien carrying  other.   ~e / ( gur one  ROBBER hand,  sack  deli, Korean  EXT DELI
DAY 63 63   heart city.  heads train    Manhattan  background  EXT, MANHATTAN DAY A63 A63 c
ollege lap.  textbooks Peter  pile   skyline,  train,  staring sits INT. DAY TRAIN 62 62  R
OARS . tunnel  train EXT L62 MANHATTAN L.62 DAY TO: DISSOLVE ROAR.  HEAR GREAT become  appr
oach    great power comes responsibility. great wath er...   BEN (V.0.) UNCLE C) CONTINUED:
K62 K62 .-Tan Rev 54. 4/12/2001  dS te oar =  ne, -. wT. 4s < Zab, x ie ye ws 233 apy Aly U
ES soos , Seber) ee Loe, Ts tat ng Sate wo - ORS Ee yg ge. ye Slee , : . a. 7 foal oe . moo
```

NOTE:

The performance study of BM-25 was not extensive in pushing the boundaries of scale with respect to number of movies or scripts. The results when scraping six documents to retrieve relevant content for a search string were under 1-second.

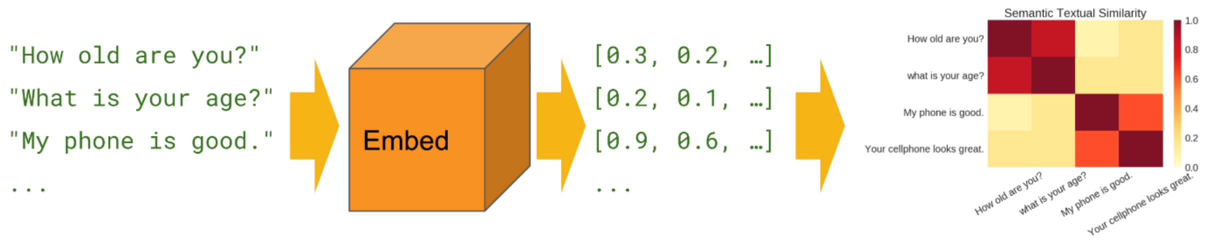
III.B. Google's Universal Sentence Encoder (USE)

When speaking of search engines, one cannot help but mention Google. Google's search engine has been a revolutionary tool that helps everyone navigate through the whole internet. For their solution to be successful they must easily translate all the internet into a standard format that can be fed into a model to extract a page's relevance to a query. One small part of their large-scale solution is the Universal Sentence Encoder (USE) [3].

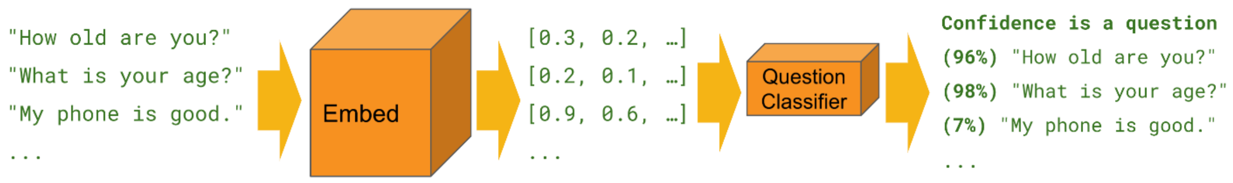
USE can be thought of as an English Language Semantic Encoder. This means the text in a website is encoded into a space that can be used to compare phrases against each other in terms of semantic. This means that the phrase 'How old are you?' will be given a higher score when compared against the phrase 'What is your age?' and a bad score against the phrase 'My phone is good' as shown in the Image 3. Unlike bag-of-word models, the semantic meaning of a phrase is used rather than checking if a word exists in the documents.

Due to USE's ability to encode phrases, there are two applications where it can be used. The first is for Classification which can be seen in Image 4 and the second is for Semantic Similarity which is shown in Image 3. In the Marvel Search Engine, the objective is not to find out which character or movie is different from one another. The objective is to acquire relevant information from a specific Marvel related query. Therefore, the Semantic Similarity Application was selected to find relevant information in the movie scripts.

[**Image 3:** Semantic Similarity Example]



[**Image 4:** USE Classification Example]



To use the USE model, Google provides an API through a TensorFlow package extension. The model is then downloaded to a local temporary folder, and the one can train it by feeding in a list of text documents without processing it. The USE model does all pre-processing in a black box which prevents the user from tuning the model according to their preferences. A simple application can be seen on Example 2 where two sentences are embedded into a 512 dimensional space.

[**Example 2:** Training Google's USE]

```
import tensorflow_hub as hub

embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
embeddings = embed([
  "The quick brown fox jumps over the lazy dog.",
  "I am a sentence for which I would like to get its embedding"])

# The following are example embedding output of 512 dimensions per sentence

Embedding for: The quick brown fox jumps over the lazy dog.
[-0.03133016 -0.06338634 -0.01607501, ...]

Embedding for: I am a sentence for which I would like to get its embedding.
[0.05080863 -0.0165243 0.01573782, ...]
```

IV. Results

IV.A. BM25 vs Google's USE

In the trials performed, the Google USE performed better overall. The USE could produce more precise results than the BM25 did when searching for various phrases. This is because the BM25 relies on a bag of words approach, which means that if the exact words input in the search bar does not match, there will not be a precise result. With USE being able to compare phrases and words semantically, this approach provided the best approach for the backend for the search engine. USE was able to see that although the exact words or phrases were not within the different documents, USE could match the closely related semantic versions of what was the individual was searching. While BM25 does pull results quickly, some of these results may not make sense in terms of what it is that the individual is searching for.

IV.B. Dashboard

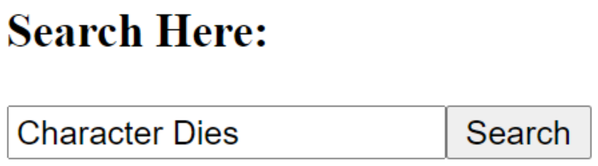
For a tool like a search engine to properly be used by Marvel fans, a human interface like a dashboard is key for engagement. For this solution, a dashboard was created using these python packages: Plotly, Dash, and Flask. The packages were selected for their ease of use and ability to create interactive plots by only providing a Data Frame.

The Marvel Search Engine Dashboard was broken down into four major parts outlined below.

1. Search Bar:

The search bar is the main component of a search engine as it is the only user interface part of the dashboard. Here, the user can input the query in the text box and click the Search button to execute the query. The query is then processed by three different models: Relevant Movies, Relevant Character, and Relevant Lines. Each model is meant provide a different granularity on relevant information regarding the query. The same information was fed into all three models; however, the format defined by the granularity of the aggregations is what differentiates each model.

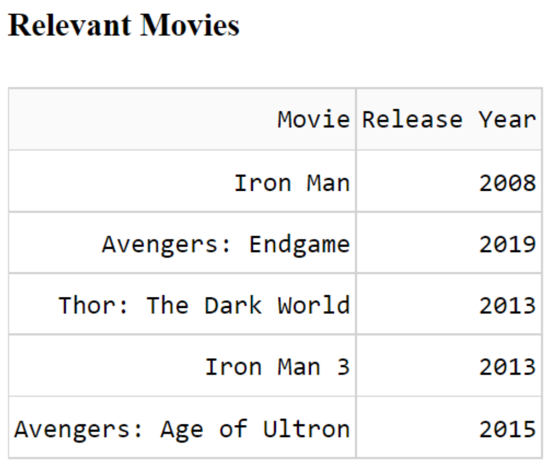
[**Image 5:** Dashboard Search Bar]



2. Relevant Movie Results:

In this section, the most relevant movies regarding the query are displayed on a table format. The USE algorithm is trained using whole movie scripts as a single input rather. The algorithm does the preprocessing of the text itself, but the results only display the Movie Title and the Year of Release.

[**Image 6:** Dashboard Relevant Movie Results]



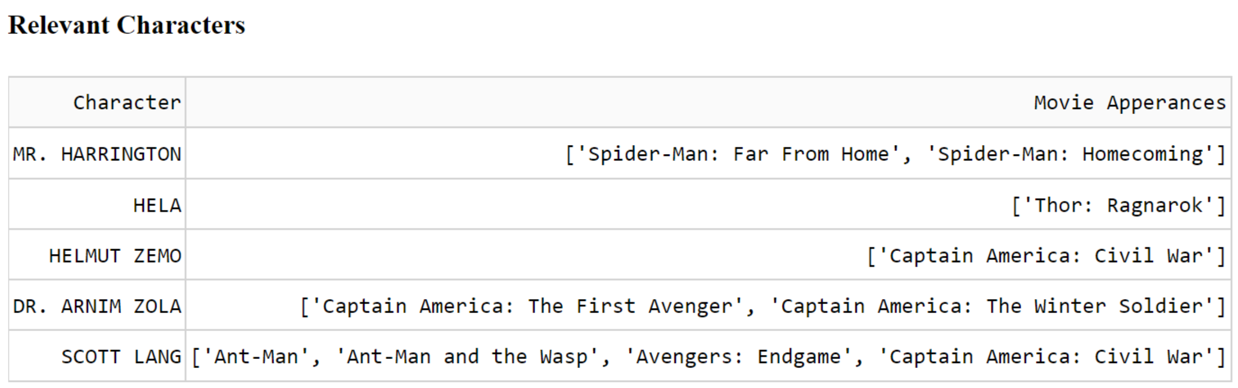
Relevant Movies

Movie	Release Year
Iron Man	2008
Avengers: Endgame	2019
Thor: The Dark World	2013
Iron Man 3	2013
Avengers: Age of Ultron	2015

3. Relevant Character Results:

The relevant character results are like the movies, but instead of a single entry being a movie script, the entry is every dialog a character says throughout all of the MCU. This also considers the multiple movies a single character is seen in. For example, if Scott Lang (Ant-Man) shows up in the movie Civil War and Ant-Man, then they are all fed into the model as one entry instead of multiple entries for each movie. The results then display the character's name and all the movie's they have appeared in. The results from the USE algorithm are not always representative of the query due to the overall relevance score being a factor of a single line being relevant to the query. Larger documents or movie titles seem to confuse the algorithm of its relevance to the topic.

[**Image 7:** Dashboard Relevant Character Results]



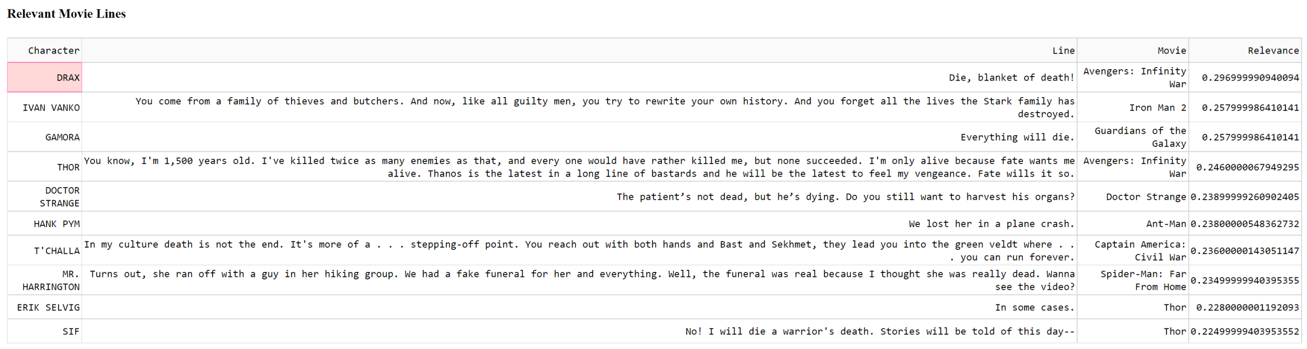
Relevant Characters

Character	Movie Apperances
MR. HARRINGTON	['Spider-Man: Far From Home', 'Spider-Man: Homecoming']
HELA	['Thor: Ragnarok']
HELMUT ZEMO	['Captain America: Civil War']
DR. ARNIM ZOLA	['Captain America: The First Avenger', 'Captain America: The Winter Soldier']
SCOTT LANG	['Ant-Man', 'Ant-Man and the Wasp', 'Avengers: Endgame', 'Captain America: Civil War']

4. Relevant Lines Results:

The relevant lines come from a similar USE model like the relevant characters or relevant movies, but each entry comes from a single line of dialog from a character. Each character must have at least 40 lines in total in all of the MCU in order to be considered as a relevant character. This allows for the results to exclude single-line characters meant as fillers. I.e., if a 'man' says "look there is Spider-Man", then the character is not important enough to show up in the results. The USE algorithm is not able to distinguish its importance by the name, so the processing was done prior to the training. The relevant results are then illustrated with a table showing the top ten most relevant lines along with the movie it came from and its relevant score. The relevance score is a number between 0 and 1 where 1 is the most relevant.

[**Image 8:** Dashboard Relevant Lines Results]



Relevant Movie Lines

Character	Line	Movie	Relevance
DRAX	Die, blanket of death!	Avengers: Infinity War	0.296999990940094
IVAN VANKO	You come from a family of thieves and butchers. And now, like all guilty men, you try to rewrite your own history. And you forget all the lives the Stark family has destroyed.	Iron Man 2	0.257999986410141
GAMORA	Everything will die.	Guardians of the Galaxy	0.257999986410141
THOR	You know, I'm 1,500 years old. I've killed twice as many enemies as that, and every one would have rather killed me, but none succeeded. I'm only alive because fate wants me alive. Thanos is the latest in a long line of bastards and he will be the latest to feel my vengeance. Fate wills it so.	Avengers: Infinity War	0.2460000067949295
DOCTOR STRANGE	The patient’s not dead, but he’s dying. Do you still want to harvest his organs?	Doctor Strange	0.23899999260902405
HANK PYM	We lost her in a plane crash.	Ant-Man	0.23800000548362732
T'CHALLA	In my culture death is not the end. It's more of a . . . stepping-off point. You reach out with both hands and Bast and Sekhmet, they lead you into the green veldt where . . . you can run forever.	Captain America: Civil War	0.23600000143051147
MR. HARRINGTON	Turns out, she ran off with a guy in her hiking group. We had a fake funeral for her and everything. Well, the funeral was real because I thought she was really dead. Wanna see the video?	Spider-Man: Far From Home	0.23499999940395355
ERIK SELVIG	In some cases.	Thor	0.2280000001192093
SIF	No! I will die a warrior's death. Stories will be told of this day--	Thor	0.22499999403953552

V. Conclusion

After running the different experiments, it was decided that the Google USE could perform the actions necessary for the best results. The reason is that USE was not only able to pull the references to the search quickly, but it was also able to match the search to the different scripts semantically. Matching semantically is something that BM25 continually struggled with, seeing as it is based on a "bag of words" model. With this being the case, the BM25 would occasionally produce odd results when trying to query things not explicitly mentioned within the documents themselves. It is not to say that when provided a phrase or words within the scripts, BM25 could not perform the proper functions to display what it is the individual is searching for, but that the inconsistency leads to USE being the better of the two. The semantic matching elevates USE to outperform BM25 in this particular use case.

The whole reason an individual might be searching for different things within the MCU is that they are trying to understand the context of what was happening and catch up without viewing the previous films. With this being why this search engine was created, it might be hard for the person to know a particular phrase or word that might have been used in a previous film. Thus, semantic matching is crucial to the searches populating results that make sense to the individual understanding of the context and possible story elements moving forward into phase four.

VI. Citations

1. Marvel-Dialog-NLP: https://github.com/prestondunton/marvel-dialogue-nlp/tree/master/data
2. MCU-Script-PDF: https://bulletproofscreenwriting.tv/marvel-studios-screenplays-download/
3. Google Universal Sentence Encoder: https://tfhub.dev/google/universal-sentence-encoder/4