NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ4034 Information Retrieval

Group 7: Assignment Report

Name	Matriculation Number
Hsiao Chia Yu	U1821839L
Li Wei-Ting	U1823750J
Neo Shun Xian Nicholas	U1820539F
Sim Song Qin	U1820620J

Tables of Content

Links to the Presentation Video, Data and Source Code

YouTube Link to Presentation Video

Video Link

https://youtu.be/khasjY4OnBM

Title of the Video

CE/CZ4034 Group 7 Video Presentation

Data Files

Google Drive Link

https://drive.google.com/file/d/1HKwmMm2fs1s_ZW_1mitVDxPcfQdC6ZVY/view?usp=sharing

Title of Zip Folder

4034-group-7-assignment-data.zip

Source Codes & Libraries

Google Drive Link

https://drive.google.com/file/d/12CpsiGcW-tGmURcOvNDqoQDohBBbPY4B/view?usp=sharing

Title of Zip Folder

4034-group-7-assignment-source-code.zip

Saved Weights for the Classification tasks (if needed)

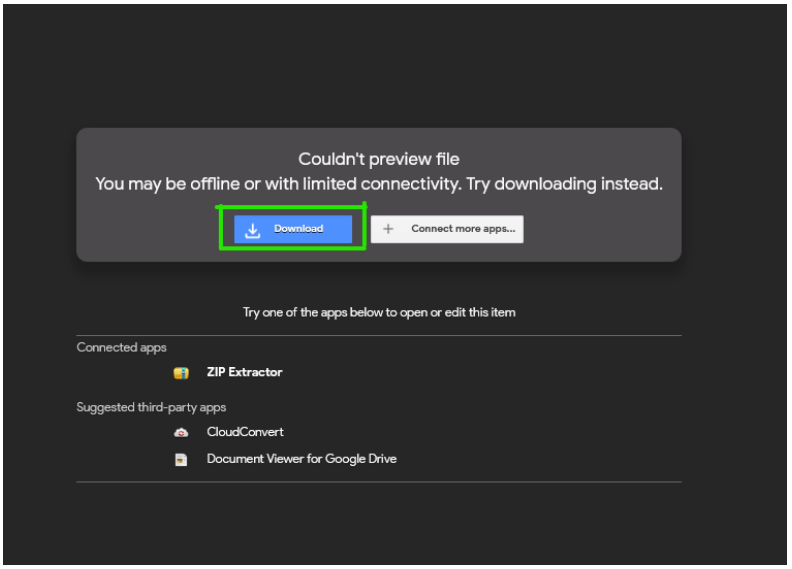
Google Drive Link

https://drive.google.com/drive/folders/1zB_-cqWTy18zcnilWp6rmMattbmvGgzX?usp=sharing

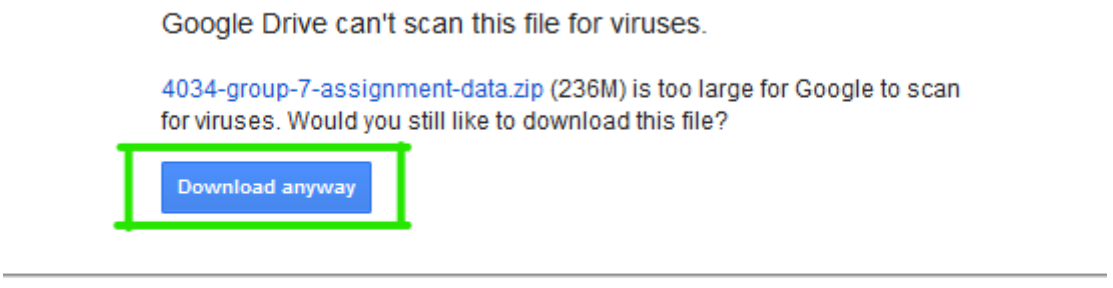
Note: A README.txt is included in this folder to explain how and where the saved weights can be placed into the folder with the source codes.

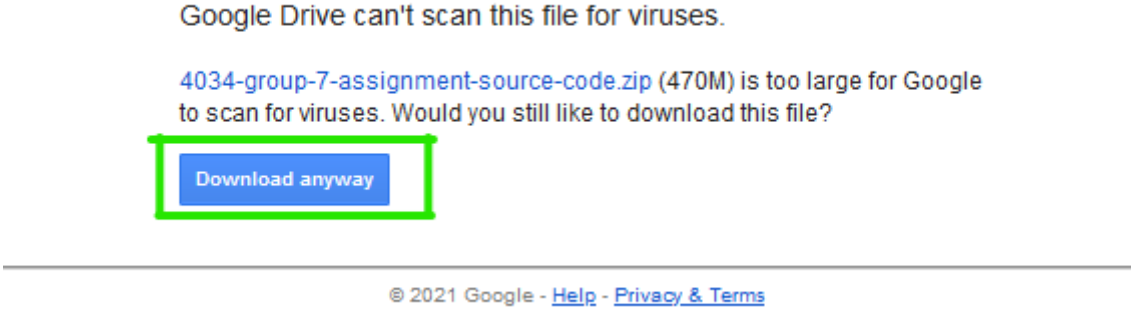
Small note about the two Zip Folders uploaded on Google Drive

As both zip folders (Data Files and Source Codes & Libraries) are quite large in sizes, there will not be any previews of the zip folders. The screenshot below shows what you might see after clicking the two links leading to the zip folders in Google Drive. Please proceed to click on the 'download' button.



Then proceed to click on 'Download anyway' to download and access the two zip folders.





Work Breakdown

Song Qin, Wei-Ting: Crawling, User Interface, Presentation Video

Chia Yu: Indexing and Querying using Solr

Nicholas: Classification

Introduction

Singapore is a food paradise as acknowledged by many foreigners and locals. There are just so many food options to choose from and sometimes this might be troublesome for one to look for food options he/she wants to try on. As such, food review platforms provide information that update users with the comments and ratings made by other users on a particular stall or restaurant, hoping that the user will make an informed decision for their next meal.

In this project, we have developed an information retrieval system for sentiment analysis on Singapore restaurant data crawled from trip-advisor. We first crawl our data using a python library named BeautifulSoup. Then, we use SOLR as the backend of the system to perform indexing and querying on the dataset. We then move on to information extraction to gain some insights on our crawled dataset. Afterwards, we use various deep learning approaches to do sentiment classification on the data based on a supervised learning approach and also various ways to enhance classification. Lastly, we implemented all of these using PySimpleGUI, a basic Graphical User Interface (GUI) to enhance user experiences.

Crawling

Overview

In this section, we will be doing a two-level web crawling on tripadvisor's Singapore restaurants landing page to get the customers' comments, ratings, cuisines of the restaurants they have been to. The first level is to crawl the restaurant names, restaurant types (e.g Italian, European, Asian, Cafe etc.) and also the webpages that lead to the details of the restaurants. The second level is to crawl the specific details of restaurants, such as the reviewer's name, ratings and comments given, based on the websites crawled in the first level.

Question 1

1.1 How the corpus is crawled and stored

For both levels, we make use of BeautifulSoup to crawl our data from tripadvisor. Frequent inspection of the web elements is required in these two levels.

1.1.1 First Level Crawling

For the first level, we wrote a small test case first to crawl one set of data from the web to see if our crawling code works perfectly fine before we iterate it for multiple restaurants from multiple websites. After scrolling through a few entries, we realised that there are restaurants that were labelled as 'sponsored', which are repetitive as we proceed to crawl our data. Hence, we need to eliminate the 'sponsored' entries, which have the same tag ID as the targeted restaurant details we want to crawl, so that there will not be repeated entries. We observed that the restaurant labelled 'sponsored' appears after every 5 entries we want to crawl. Hence, we wrote a code to skip the crawl for every 6th instance of the data. Figure 1.1.1 shows an example of the data that we want to crawl and the 'sponsored' entry. Figure 1.1.2a and 1.1.2b shows how we observed the inspected elements from the webpage. Figure 1.1.3 shows the code snippet of how we make a request to the webpage. Figure 1.1.4 shows the code snippet for the small test and the crawled output.

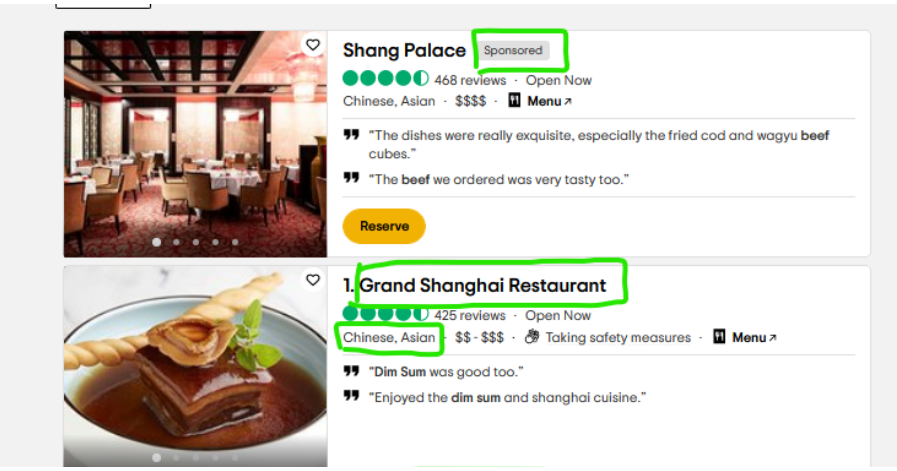


Figure 1.1.1: Example of the data we want to crawl and the 'sponsored' entry (boxed in green)

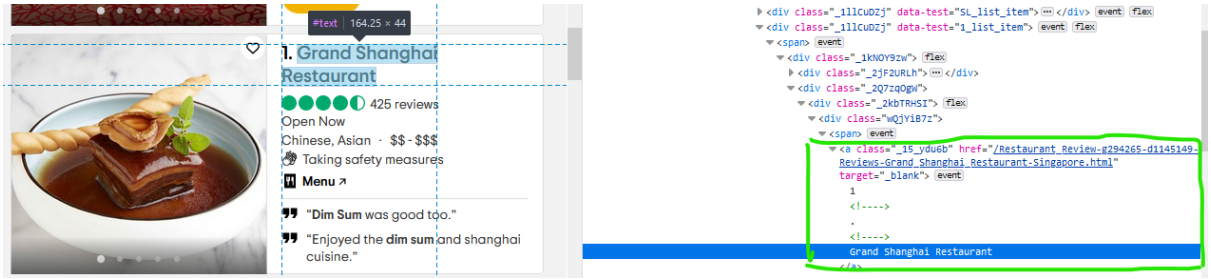


Figure 1.1.2a: Inspecting the restaurant's name and review webpage

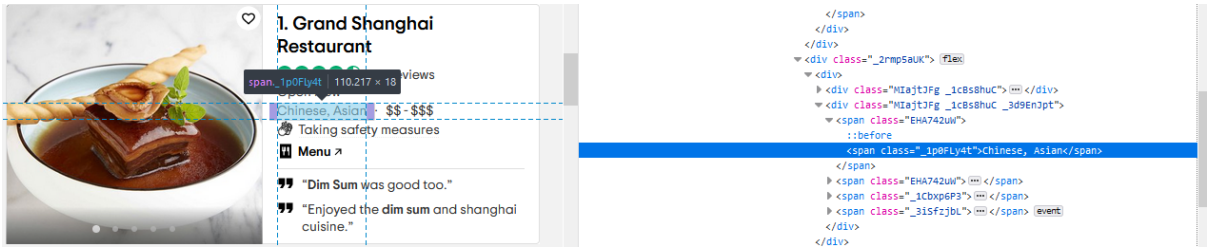


Figure 1.1.2b: Inspecting the restaurant's cuisine or type

```
# flag variable to check the scrape
# if unsuccessful scrape, try again
unsuccessful = True
#count = 0
while unsuccessful:
    url = 'https://www.tripadvisor.com.sg/Restaurants-g294265-Singap
    # make request to the site
    response = get(url)
    html_soup = BeautifulSoup(response.text,'html.parser')

    # find all with the same class tag
    res_containers = html_soup.find_all('div', class_='_1llCuDZj')
    print(type(res_containers))
    # check the number entries in a page and to determine the next p
    print(len(res_containers))

    # pause the loop
    sleep(randint(2,5))

    if len(res_containers) != 0:
        unsuccessful = False
```

```
<class 'bs4.element.ResultSet'>
37
```

Figure 1.1.3: Code snippet of how a request is made to the webpage

```
# get name of first stall
stall_name = first_comment.find('a', class_='_15_ydu6b').text
stall_name
```

```
'1. Grand Shanghai Restaurant'
```

```
base = 'https://www.tripadvisor.com.sg'
cust_name = []
webpage_list = []
temp_str = ''
for i,j in enumerate(res_containers):
    # to remove ads that may be repetitive
    if i%6 == 0:
        continue
    else:
        cust_name.append(j.find('a', class_='_15_ydu6b').text)
        temp_str = base + j.find('a', class_='_15_ydu6b').get('href')
        webpage_list.append(temp_str)
```

```
# scrape website
first_comment = res_containers[1]
#first_comment
```

```
base = 'https://www.tripadvisor.com.sg'
```

```
webpage = first_comment.find('a', class_='_15_ydu6b').get('href')
base+webpage
```

```
'https://www.tripadvisor.com.sg/Restaurant_Review-g294265-d1145149-Reviews
-Grand_Shanghai_Restaurant-Singapore.html'
```

```
# scrape restaurant type
res_type = first_comment.find_all('span', class_='_1p0FLy4t')
res_type[2].text
```

```
'Chinese, Asian'
```

Figure 1.1.4: Simple crawler code and the crawled output

After the small test code, we proceed to expand the code and make it into iterations to crawl more restaurants' data. In total, we crawled a total of 1013 restaurants' data for the first level crawl and saved our data into a .csv file for the second level crawling. Figure 1.1.5 shows some entries of the data crawled from the webpage.

18	17. Fu Lin Men Chinese Restaurant (SRC)	Chinese, Asian	https://www.tripadvisor.com.sg/Restaurant_Review-g294265-d15608624-Reviews-Fu_Lin_Men_Chinese_Restaurant_SRC-Singapore.html
19	18. Waterfall Ristorante Italiano	Italian, Mediterranean	https://www.tripadvisor.com.sg/Restaurant_Review-g294265-d3952172-Reviews-Waterfall_Ristorante_Italiano-Singapore.html
20	19. Bar-Roque Grill	French, Steakhouse	https://www.tripadvisor.com.sg/Restaurant_Review-g294265-d4504877-Reviews-Bar_Roque_Grill-Singapore.html
21	20. Spice Brasserie	Seafood, Asian	https://www.tripadvisor.com.sg/Restaurant_Review-g294265-d5423955-Reviews-Spice_Brasserie-Singapore.html
22	21. Hua Ting Restaurant	Chinese, Asian	https://www.tripadvisor.com.sg/Restaurant_Review-g294265-d794020-Reviews-Hua_Ting_Restaurant-Singapore.html

Figure 1.1.5: Some of the entries crawled

1.1.2 Second Level Crawling

For the second level crawling, we will first load the .csv that was generated from the first level crawling. We then looked for the fields that we required and noted its id from the HTML tag. As such we crawled the reviewer's name, rating and comment. Figure 1.1.6 shows the data we want to crawl and Figure 1.1.7a, 1.1.7b, 1.1.7c shows how we observed the inspected elements from the webpage.

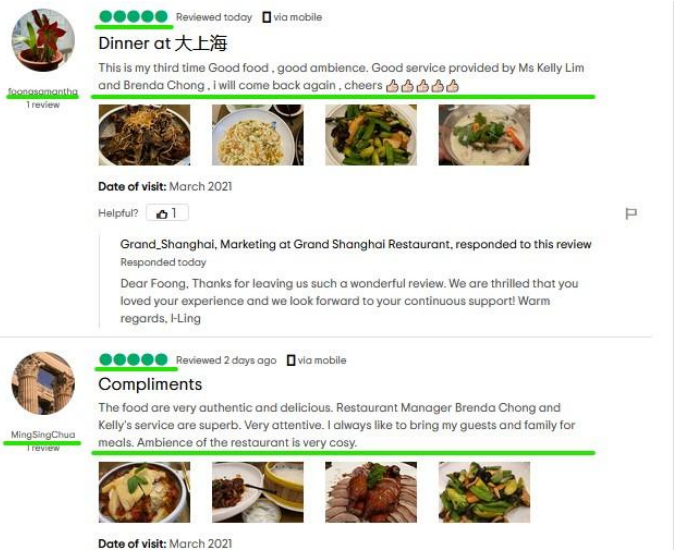


Figure 1.1.6: Example of the data we want to crawl (underlined in green)

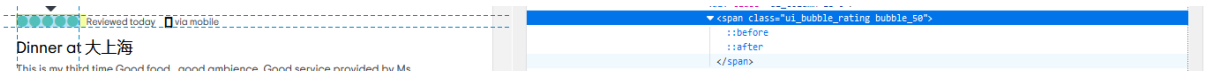


Figure 1.1.7a: Inspecting the rating made by a user

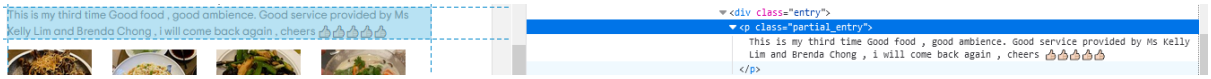


Figure 1.1.7b: Inspecting the comment made by a user

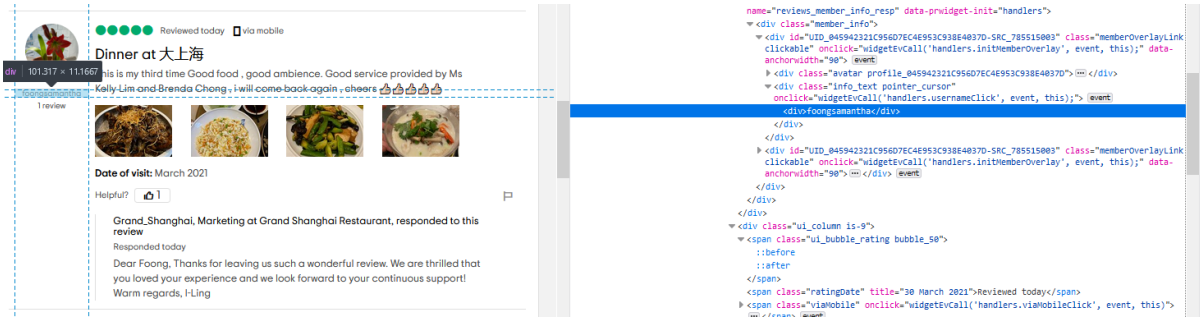


Figure 1.1.7c: Inspecting the username of the reviewer

We then make the program crawl iteratively through the 1013 restaurants based on the url generated from the first level crawling. We crawled 100 user data (reviewer's name, rating and comment) from each restaurant. There may be some crawling issues along the way. As such the total number of records is less than 101300 entries. The crawled Figure 1.1.8 shows some entries of the data crawled from the webpage.

7783	The RANCH Steakhouse By A'	American, Steakhouse	HIT_1	3 We ordered 2 x Roasted Bone Marrow (36 SGP $,) ,1 x Black Angus Ribeye 350 g (49 SGP $) , x Hokkaido Wagyu Sirloin 400g (98 SGP $),1 x Field Mush
7784	The RANCH Steakhouse By A'	American, Steakhouse	KaruTA	4 Celebrated a family member's Birthday at this steakhouse. Ambiance was great & we were greeted by very helpful & friendly staff. Had the dry-aged Pc
7785	The RANCH Steakhouse By A'	American, Steakhouse	IngunnCecilie	4 We were at this restaurant twice while we were in Singapore. First time with friends, and second time as a couple. Both times we had a really good mea
7786	The RANCH Steakhouse By A'	American, Steakhouse	hmblakeway	5 Without a doubt the best steak I have had in Singapore in the last 6 months. I had stopped going out to eat steak in Singapore because the quality just c
7787	The RANCH Steakhouse By A'	American, Steakhouse	TAR1293	3 My food was good, but my boyfriend ordered a steak that was supposed to have been cooked medium but when it arrived and we complained that it v
7788	The RANCH Steakhouse By A'	American, Steakhouse	JazhelL	4 I missed the entrance but was lucky to saw one of their crew around Clarke Quay and got the right directions finally! Advise to look for the cab drop off
7789	The RANCH Steakhouse By A'	American, Steakhouse	jusoh64	5 Ranch is indeed a good place for good steak at reasonable price. Service was excellent and so was decor and ambience. For those who dont fancy steak
7790	The RANCH Steakhouse By A'	American, Steakhouse	leedon72	5 A night in Clarke quay and not sure where we were gonna eat even though weâ€™ve been to Clarke quay dozens and dozens of times however we had
7791	The RANCH Steakhouse By A'	American, Steakhouse	KooYuMin	4 Great service esp by the Assistant Manager! Beef was great serving was generous. Tenderloin was just right. But the T bone was overdone, requested

Figure 1.1.8: Some of the entries crawled

1.2 What kind of information users might like to retrieve from the crawled corpus with sample queries

Our project focuses on assisting users to find food reviews and food cuisines. Therefore, the information that users might like to retrieve in order for them to make informed decisions are as follows:

- Name of the restaurant
- Type of the restaurant (e.g Italian, European, Asian etc.)
- Reviewer's rating
- Reviewer's comment

1.3 Number of records, words and types (i.e. unique words) in the corpus

Initially, we have a total of 97190 records after crawling. As some users may submit their review multiple times, there will be duplications of records in the corpus. As such, we filtered out the duplicates by matching the records using the reviewer's name and the reviewer's comment. After removal of duplicates, we are left with 88042 records in the corpus. Figure 1.3.1 shows how the removal of duplicates is done.

```
[7]  # read the test data
     data_test_raw_ = pd.read_csv(COLAB_FILEPATH+'data/trip-advisor-comments.csv')
     print(f'Shape of the dataset:{data_test_raw_.shape}')
     #data_test_raw_.head()

     Shape of the dataset:(97190, 5)
```

```
[8]  # remove duplication of entries
     data_test_raw=data_test_raw_.drop_duplicates(subset={"Reviewer\'s Name","Comment"},
                                                  keep='first', inplace=False)
     print(f'Shape of the dataset after removing duplicates:{data_test_raw.shape}')

     Shape of the dataset after removing duplicates:(88042, 5)
```

Figure 1.3.1: How removal of duplicates is done

There are a total of 3688240 and 137499 words and unique words respectively in the crawled corpus. Figure 1.3.2 shows how we get the total number of words and unique words from the crawled corpus.

```
[11] # total number of words
     no_words = 0
     k = list(data_test_raw['Comment'].str.count(' ') + 1)
     for i in k:
         no_words += i
     print(f'Number of words: {no_words}')

     # total number of unique words
     uniqueWords = list(set(" ".join(data_test_raw['Comment'].values).split(" ")))
     count = len(uniqueWords)
     print(f'Number of unique words: {count}')

     data_test_raw.head(3)

     Number of words: 3688240
     Number of unique words: 137499
```

Figure 1.3.2: Obtaining the number of words and the number of unique words

Indexing and Querying

Overview

We have chosen Solr as our indexing system. Solr is an open source enterprise search platform which is written in Java. It provides full-text indexing and searching for structured and unstructured documents. REST-like API with the ability to adjust response format between XML, CSV or JSON makes it easy to integrate with other subsystems.

Client apps can reach Solr by creating HTTP requests, then parsing the corresponding HTTP responses.

A simple user interface was designed to illustrate the results when a query is made. PySimpleGUI is the Python package used to create our GUI. It is solely based on Tkinter.

Question 2

2.1 Solr Introduction

Solr is a ready-to-use application, however some configurations are still required to provide better performances. In our application, we run the Solr server in standalone mode which means we only host on a single machine. We chose a schemaless implementation which Solr will automatically perform analysis on documents whenever the documentation is posted for indexing, necessary fields and field type will be created and managed by Solr. To prevent incorrect classification of data, one can predefine the fields with web admin interface or use schema API to configure schema.

Schema include field and field type definitions. For each field, it can be configured in the field type and properties on how the data will be indexed. For the field type, it provides 3 components to configure which includes analyzer, tokenizer and list of filters which defines the text preprocessing methods. For the analyzer, it allows us to define different sets of rules to be applied for indexing or querying separately. Tokenizer defines a set of rules used to break input sequences into lexical units, for example Solr provides N-Gram Tokenizer to break input sequences into n-gram characters. After the input sequences are tokenized, a set of filters provide the rules for linguistic analysis which includes stopwords elimination, lemmatization, stemming and etc. Figure 2.1.1 shows an example of field type configuration for text_en.

```
<fieldType name="text_en" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EnglishPossessiveFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
    <filter class="solr.PorterStemFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>
    <filter class="solr.StopFilterFactory" words="lang/stopwords_en.txt" ignoreCase="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EnglishPossessiveFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
    <filter class="solr.PorterStemFilterFactory"/>
  </analyzer>
</fieldType>
```

Figure 2.1.1 example of field type configuration for text_en

With reference on field type declaration above, Solr web admin interface provides visual representation on how Solr performs pipelined text pre-processing based for both indexing and query. Figure 2.1.2 and 2.1.3 below shows text pre-processing analysis on field type text_en

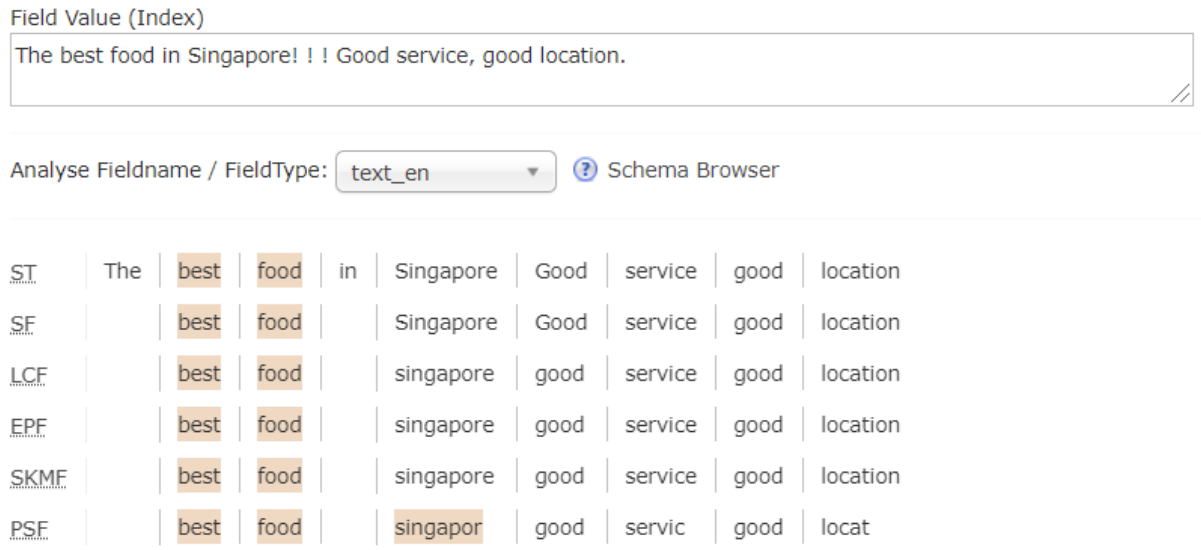


Figure 2.1.2 text analysis for indexing document

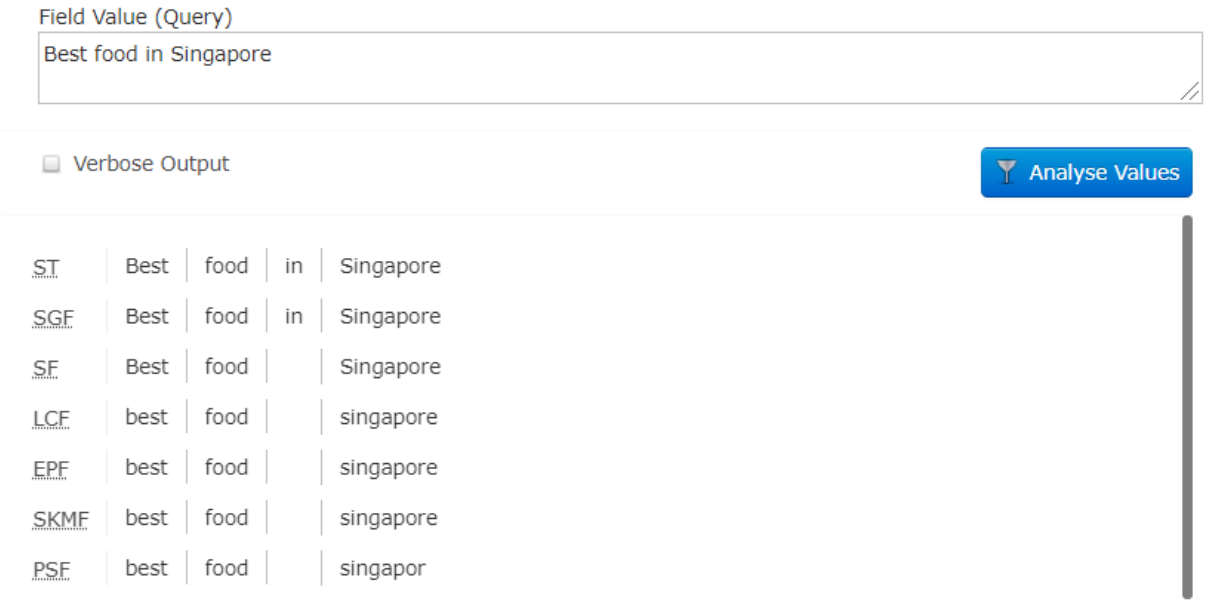


Figure 2.1.3 text analysis for query terms

2.2 Solr Core

Solr Core is an instance of a Lucene index that contains all the Solr configuration files required to use it. We need to create a Solr Core to perform operations like indexing and analyzing. Figure 2.2.1 shows a command instruction to create a new core named comment. After core is created, we may start posting documents to Solr which provides multiple posting tools for posting documents, it also takes in multiple formats for documents ranging from CSV, XML to rich text format like PDF, Word or HTML. Figure 2.2.2 below shows posting of csv document with curl tool.

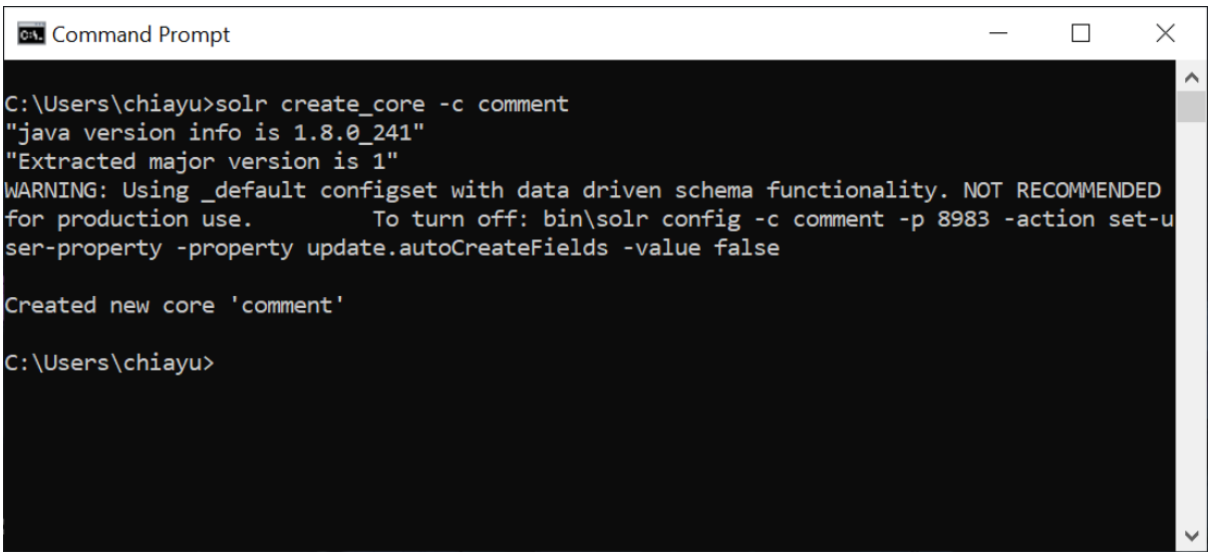


Figure 2.2.1 command to create new core

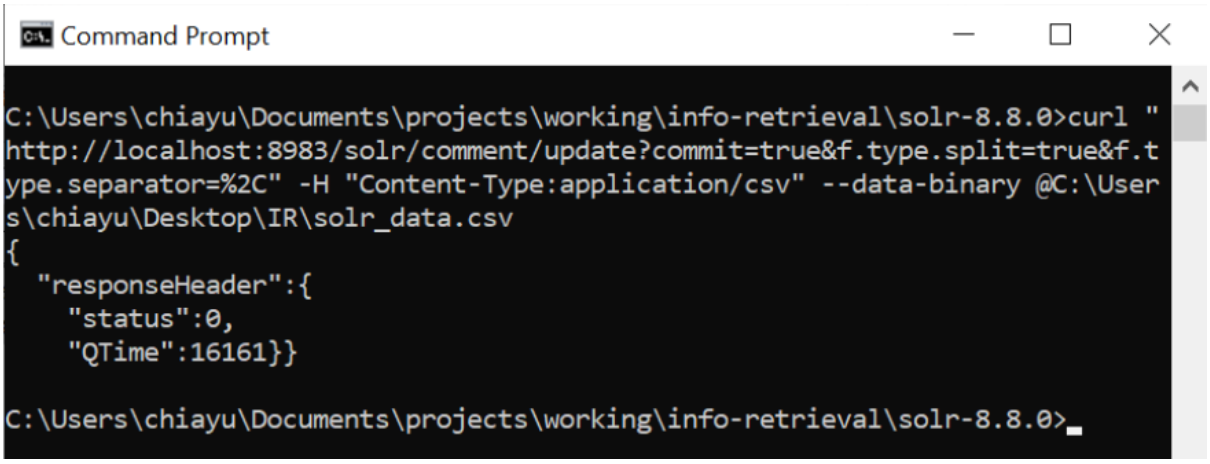


Figure 2.2.2 curl tool to post csv document to solr

After the documents are posted to Solr, we can verify the number of documents posted in core overview like Figure 2.2.3 and also check number of indexed terms for each field like Figure 2.2.4.

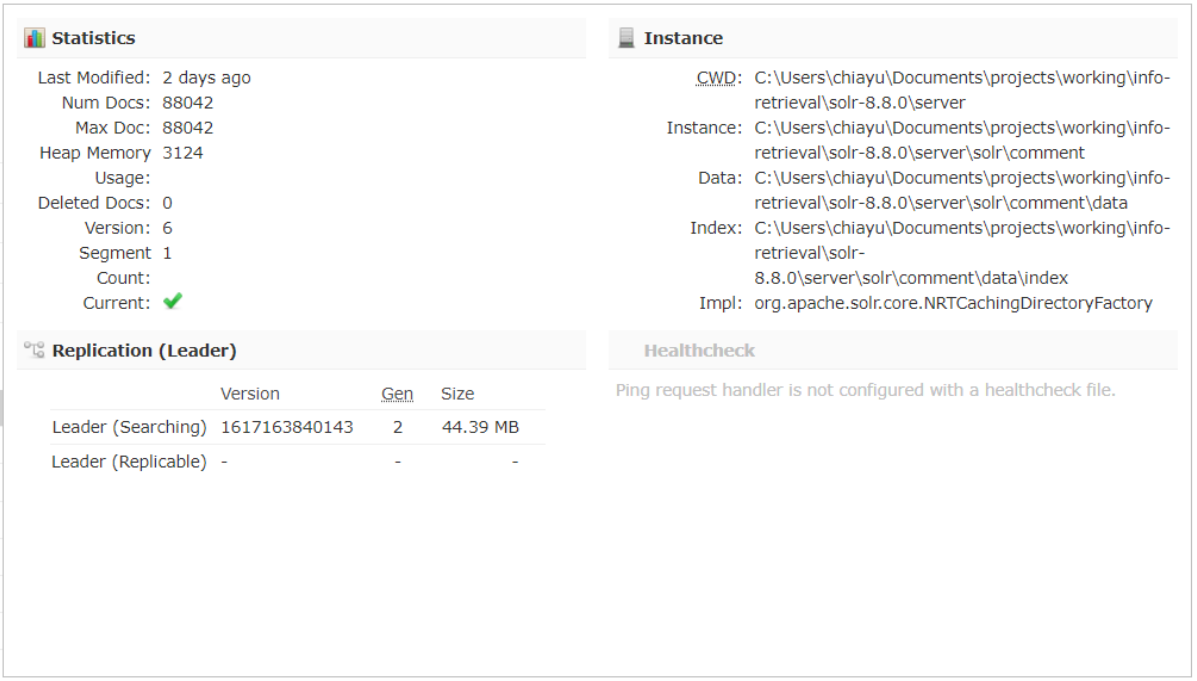


Figure 2.2.3 show number of documents in core

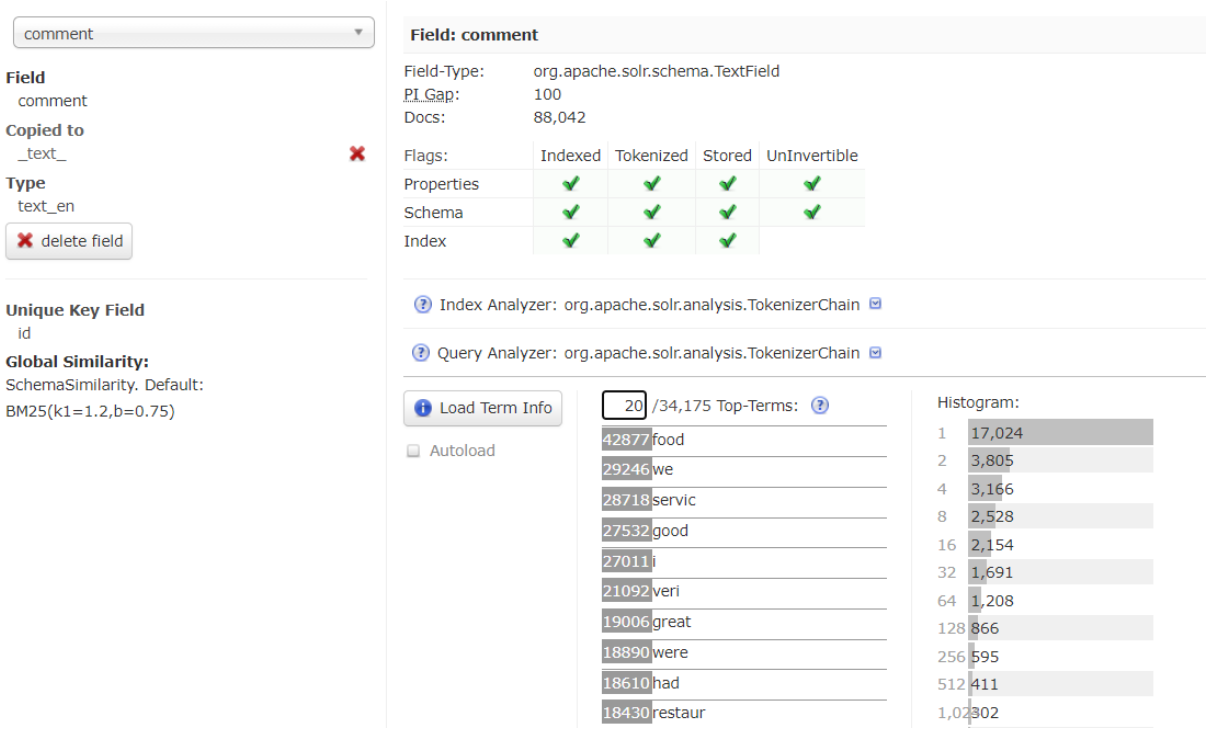


Figure 2.2.4 field definition and term information for field comment

2.3 Solr Query

In Solr web admin interface, it provides a very comprehensive query tool for us to play with querying terms with different parameters to see the effect. The table below shows a set of parameters used in our application and their purpose

q	The query strings to find matching documents and calculate for similarity score. ● Usually in format of q=field:value ● Wildcard Searches are supported like following ○ q=comment:ama?ing ○ q=comment:te*t ○ q=comment:*est
fq	Filter query, which applies a query to the search results
fl	Field list, which limits the information to be included in a query response. Similarity scores are hidden as default, using field list, we can allow score to be included in return response
df	Default field, when query string does not provides specification on which field to be search, it fall back to default field
start	Number of leading documents to skip. Together with `row` parameter, it allow us to perform pagination on search results

row	Maximum number of documents to be presented on query result
wt	Writer type, which allows Solr to change response format from csv, json, xml or etc.
debug	Allow us to see how similarity score is calculated

REST-like API is convenient to use and integrate with other subsystems, however, the http get request has a characters limit. Hence providing long query parameters can be an issue, these can be solved by modifying solrconfig,xml to set default value for query parameters.

In solrconfig.xml, we found out that all queries have a default field set to _text_ field like figure 2.3.1. Hence to allow searching on multiple fields with a single query string, we create multiple copyfield rules to map multiple fields including restaurant name, type and comment to _text_ like figure 2.3.2.

```
<initParams path="/update/**,/query,/select,/spell">
  <lst name="defaults">
    <str name="df">_text_</str>
  </lst>
</initParams>
```

Figure 2.3.1 default field configuration in solrconfig.xml

```
<copyField source="comment" dest="_text_"/>
<copyField source="restaurant" dest="_text_"/>
<copyField source="type" dest="_text_"/>
```

Figure 2.3.2 copy field rule on manage-schema

Besides that, Solr provides multiple request handlers to allow different sets of rules to be applied for searching results. Figure 2.3.3 and 2.3.4 below shows configuration applied to standard and spell checking request handlers. From Figure 2.3.5, we can see that the spellcheck distance method is using levenshtein edit distance.

```
<requestHandler name="/select" class="solr.SearchHandler">
  <!-- default values for query parameters can be specified, these
       will be overridden by parameters in the request
    -->
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
    <!-- Default search field
       <str name="df">text</str>
      -->
    <!-- Change from JSON to XML format (the default prior to Solr 7.0)
       <str name="wt">xml</str>
      -->
    <str name="fl">restaurant,type,reviewer,rating,comment,true_label,pred_label,score</str>
    <str name="facet.field">pred_label</str>
  </lst>
```

Figure 2.3.3 request handler for standard query

```
<requestHandler name="/spell" class="solr.SearchHandler" startup="lazy">
  <lst name="defaults">
    <!-- Solr will use suggestions from both the 'default' spellchecker
         and from the 'wordbreak' spellchecker and combine them.
         collations (re-written queries) can include a combination of
         corrections from both spellcheckers -->
    <str name="spellcheck.dictionary">default</str>
    <str name="spellcheck">on</str>
    <str name="spellcheck.extendedResults">true</str>
    <str name="spellcheck.count">10</str>
    <str name="spellcheck.alternativeTermCount">5</str>
    <str name="spellcheck.maxResultsForSuggest">5</str>
    <str name="spellcheck.collate">true</str>
    <str name="spellcheck.collateExtendedResults">true</str>
    <str name="spellcheck.maxCollationTries">10</str>
    <str name="spellcheck.maxCollations">5</str>

    <str name="echoParams">explicit</str>
    <str name="fl">restaurant,type,reviewer,rating,comment,true_label,pred_label,score</str>
    <str name="facet.field">pred_label</str>
  </lst>
  <arr name="last-components">
    <str>spellcheck</str>
  </arr>
</requestHandler>
```

Figure 2.3.4 shows request handler for query with spell checking

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
  <str name="queryAnalyzerFieldType">text_general</str>
  <!-- a spellchecker built from a field of the main index -->
  <lst name="spellchecker">
    <str name="name">default</str>
    <str name="field">_text_</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <!-- the spellcheck distance measure used, the default is the internal levenshtein -->
    <str name="distanceMeasure">internal</str>
    <!-- minimum accuracy needed to be considered a valid spellcheck suggestion -->
    <float name="accuracy">0.5</float>
    <!-- the maximum #edits we consider when enumerating terms: can be 1 or 2 -->
    <int name="maxEdits">2</int>
    <!-- the minimum shared prefix when enumerating terms -->
    <int name="minPrefix">1</int>
    <!-- maximum number of inspections per result. -->
    <int name="maxInspections">5</int>
    <!-- minimum length of a query term to be considered for correction -->
    <int name="minQueryLength">4</int>
    <!-- maximum threshold of documents a query term can appear to be considered for correction -->
    <float name="maxQueryFrequency">0.01</float>
    <!-- uncomment this to require suggestions to occur in 1% of the documents
      <float name="thresholdTokenFrequency">.01</float>
    -->
  </lst>
</searchComponent>
```

Figure 2.3.5 shows spellcheck component definition

2.4 Building a simple web interface for the search engine

After importing the PySimpleGUI package, we create a series of widgets, which are called “Element” in PySimpleGUI, for example: Text, InputText and Button.

All the elements created are put into a list called “layout”, after that, a “window” is created. It is the parent Element that contains all the other elements. A while loop is then created and the Window’s read() method is called to extract the events and values the user has set.

A drop down list is created for the user to choose the function between “search”(select) and “spell check”(spell). By default, it is “search”. The selected term will be passed on to the request-handler(qt) in solr. The wt parameter in the query allows users to select an output format for the chosen API, for this assignment, JSON is selected as it is a more robust response format.A query is sent to solr by using http requests(by importing urllib) and parsing the JSON response(by importing simplejson) instead of using a client API.

When using the application, a user can simply type the keywords in the search bar, and select the type of functions they want to perform, namely ‘search’ or ‘spell check’, where a user can use to check if there are any suggestions on the misspelled words. Figure 2.4.1 shows our user interface when it is launched before hitting the ‘search’ button when a query is typed.

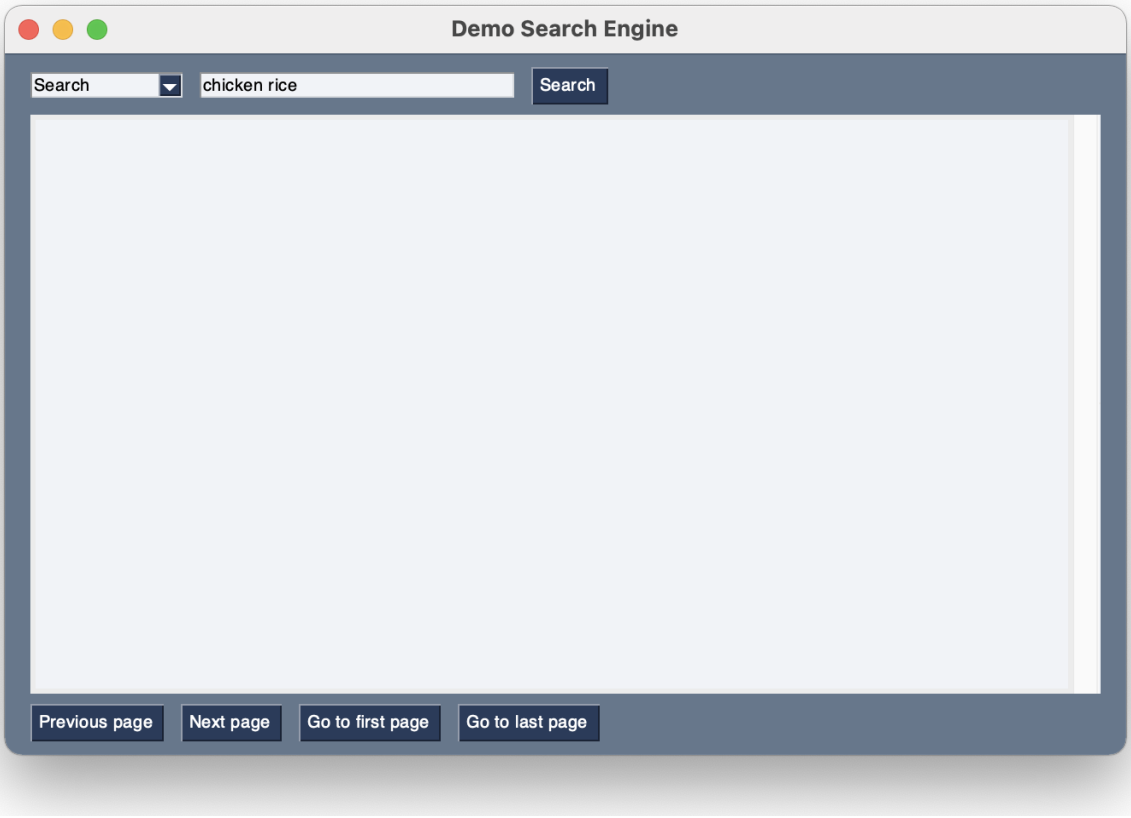


Figure 2.4.1: User Interface before hitting ‘search’

For searching, after hitting the ‘search’ button, the results will be shown in the output box below. It will first show the number of results found in the database and the query time needed for the search. By default, it will show 10 results per page.

For each result shown, it will contain the name of the restaurant, type of the restaurant, name of the reviewer, rating given, comment given, semantic predicted and tf/idf score.

A user can then choose to navigate to the previous or next page, or jump to the first or last page based on their current page. Figure 2.4.2 shows the search result after entering a query.

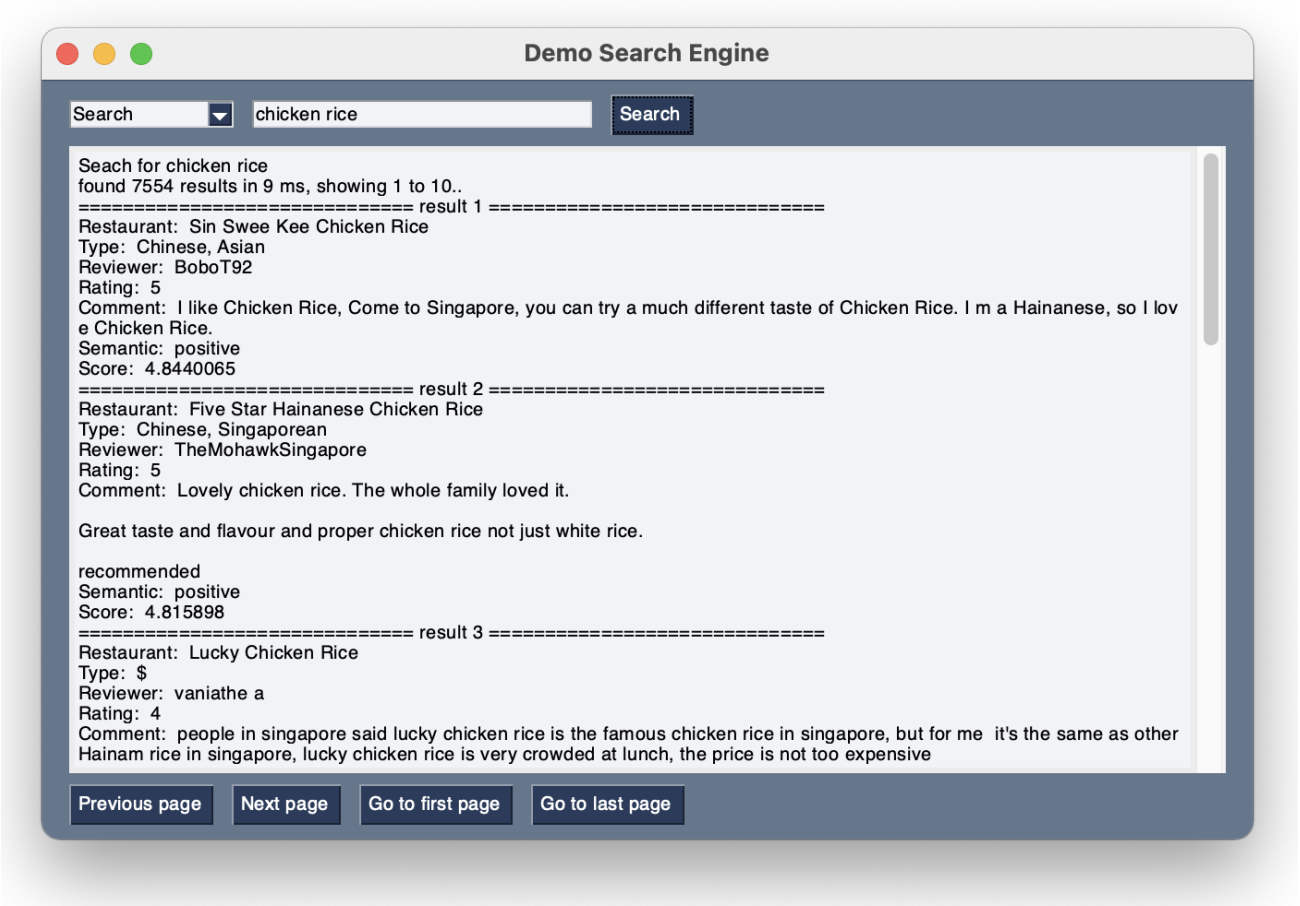


Figure 2.4.2: Search result after a query is entered

For spell check usage, the user has to select ‘spell check’ from the drop down bar, type the word and click ‘search’. It will then print out the number of results found containing this keyword and its query time. Other than that, it will also print out the suggested word for correction if there exists one. Figure 2.4.3 shows how it is done.

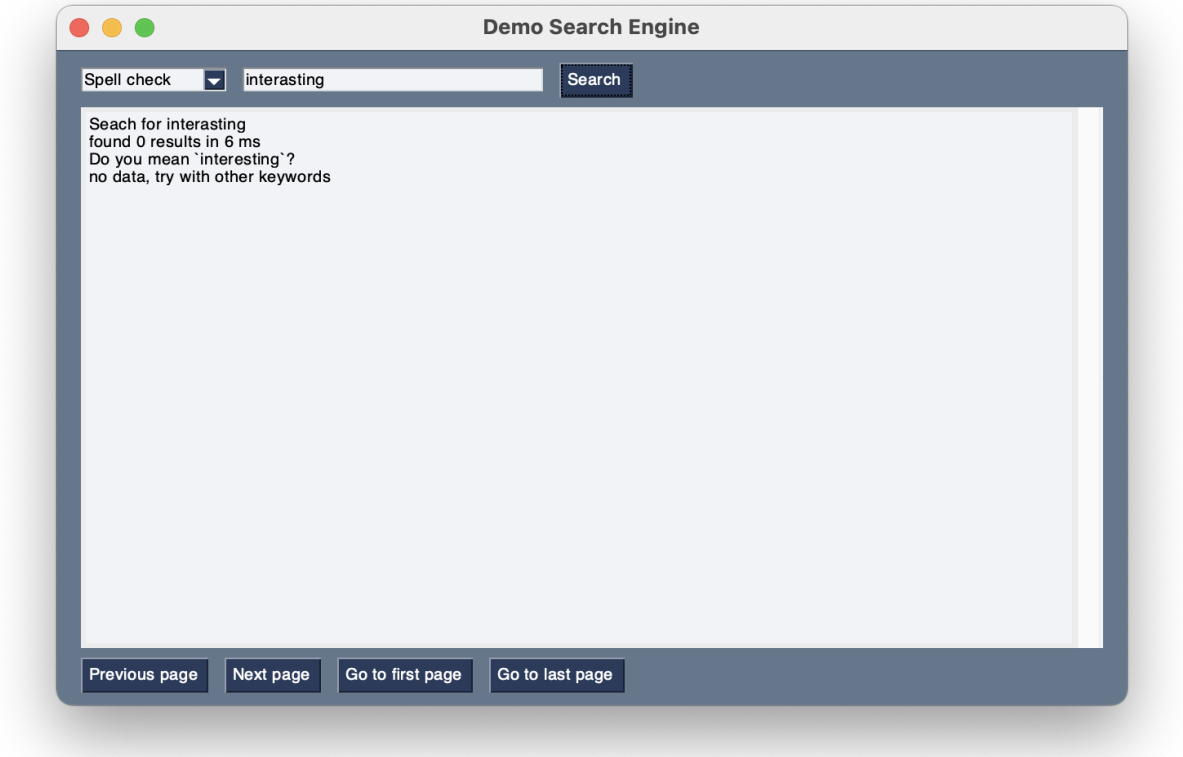


Figure 2.4.3: Spell check function

2.5 Write five queries, get their results, and measure the speed of querying

2.5.1 Multiple keywords → best food

Figure 2.5.1: Multiple keywords

When we search for “best food”, the system will return us all the entries which contain either the term “best”, “food” or both.

2.5.2 Search with quotation to group query terms together → “best food”

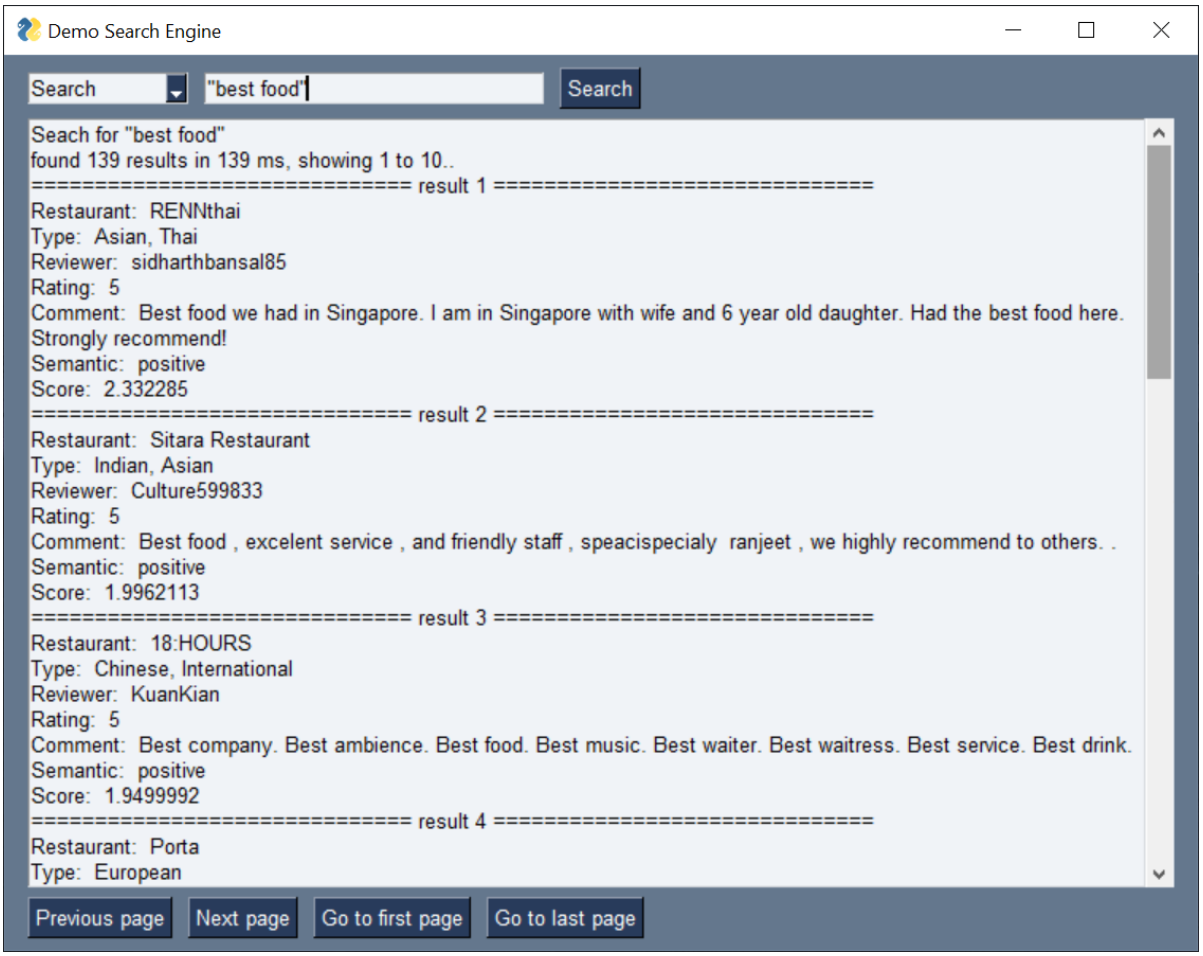


Figure 2.5.2: Multiple keywords

When searching with two or more keywords using quotation marks, solr will search for the exact sequence of the search term, instead of showing all of the entries which contain at least one of the keywords.

2.5.3 Filter keywords → best food, exclude 'chinese'

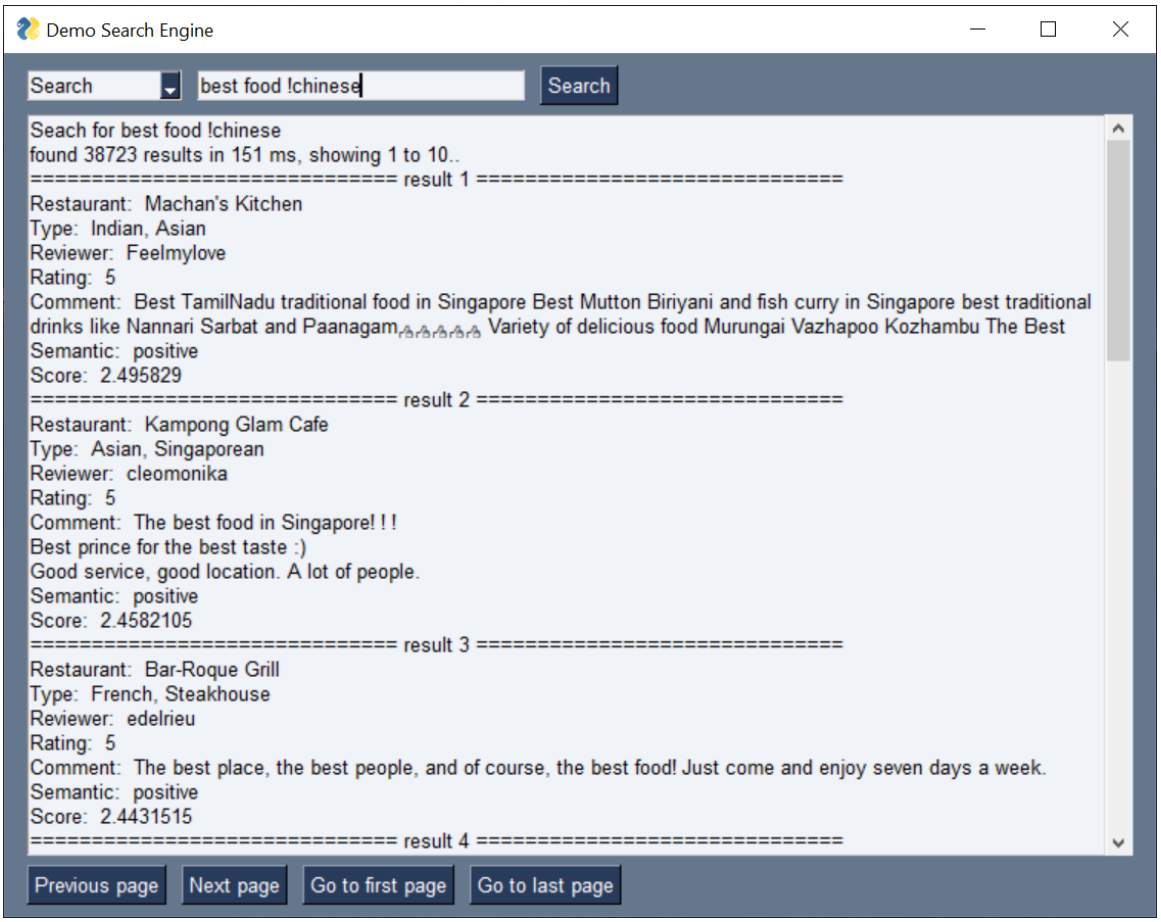


Figure 2.5.3: Filter keywords

This query searches for entries which contain the term "best", "food" or both and filters away those which contain "chinese".

2.5.4 Semantic searching (search based on specific field) → pred_label:positive

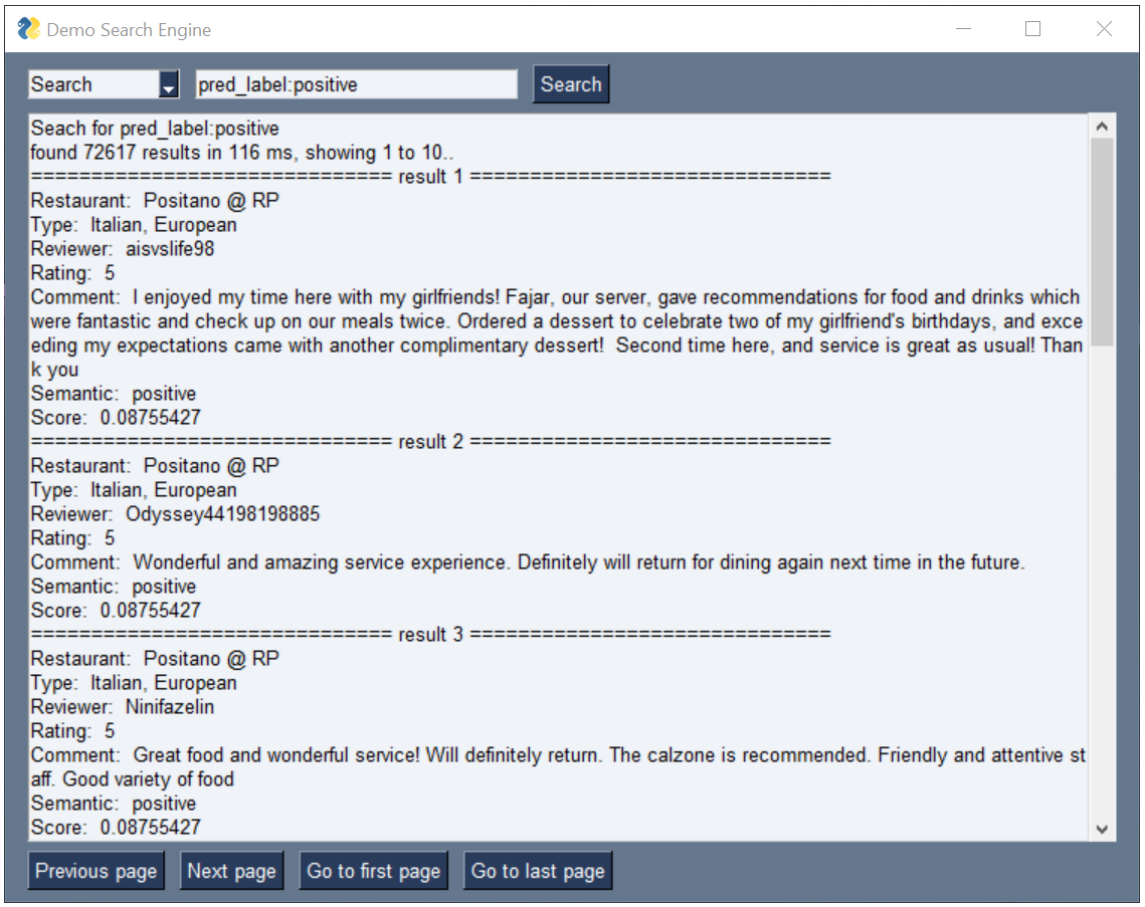


Figure 2.5.4: Semantic searching

We can search for each entry using the predicted label by typing “pred_label:(semantic)”. For this example, it will return all the entries which are predicted as positive for the semantic.

2.5.5 Single term (spell check) → amasing

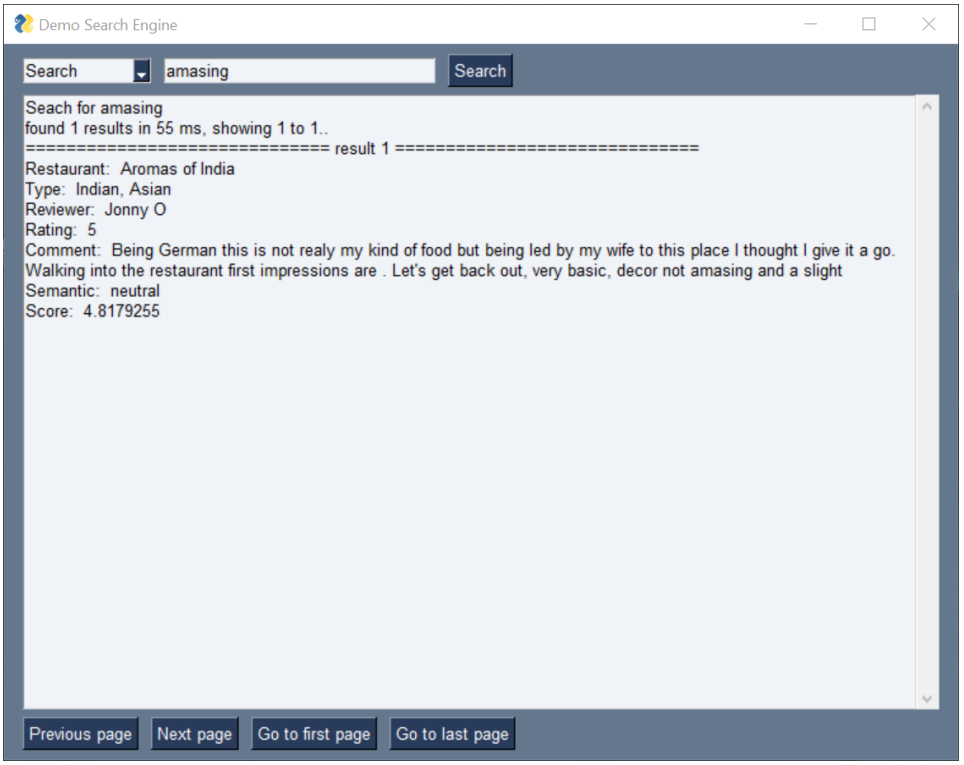


Figure 2.5.5: Single term (spell check)

By selecting the "spell check" from the drop down list, solr will do a spell check for the keyword typed and return its suggested word. At the same time, it will also show if any of the entries contain the keyword user typed. Table 2.5.1 shows the query time and the results of the five queries we made as described above.

Query term	Query time(ms)	Subsequent query time(ms)	Number of results
best food	146	2	46402
"best food"	139	2	139
best food !chinese	151	1	38723
pred_label:positive	116	0	72617
amasing	55	2	1

Table 2.5.1: Query time and results of the five queries

With reference to the table 2.5.1, Solr cache the search results to improve subsequence searches.

Question 3

We have also explored some innovations for enhancing the indexing and ranking system, We explored:

1. SolrCloud
2. Multilingual Search

3.1 SolrCloud

Solr provides alternative hosting mode namely SolrCloud, which is a distributed implementation to allow multiple machines running Solr servers as a cluster to provide high fault tolerance and high availability. Figure 3.1.1 shows the Architecture of Solr Cloud.

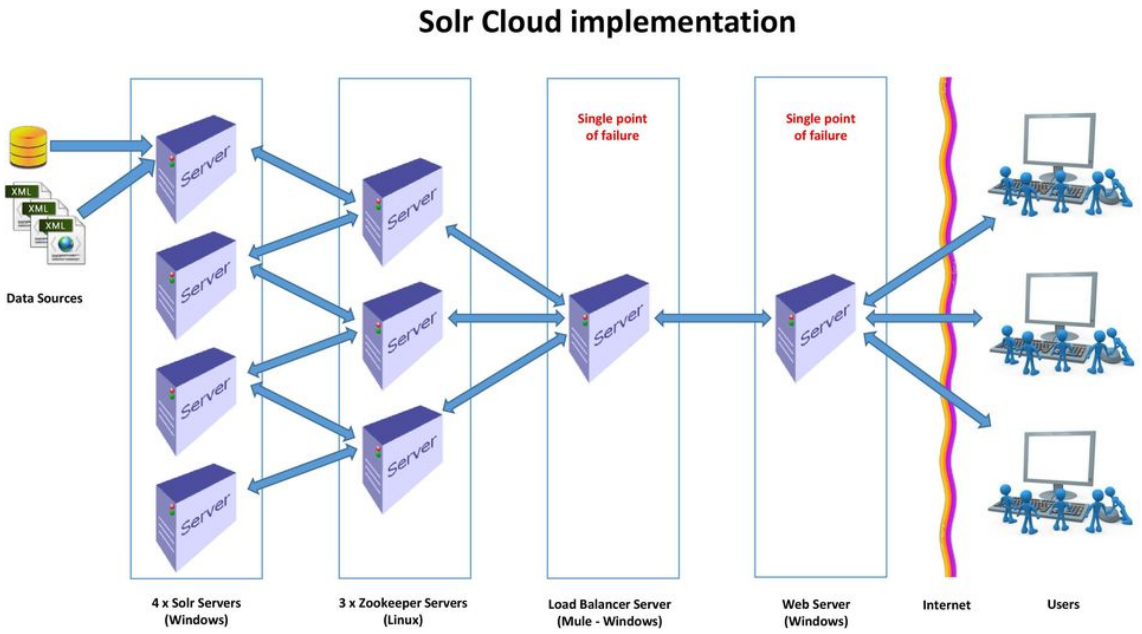


Figure 3.1.1: Solr Cloud Architecture

3.2 Multilingual search

Solr is able to perform language detection on indexing documents, and automatically adds an additional label for detected language of document, which allows us to search based on different languages as a filter as shown in Figure 3.2.1.

```
<processor class="org.apache.solr.update.processor.TikaLanguageIdentifierUpdateProcessorFactory">
  <lst name="defaults">
    <str name="langid.fl">title,subject,text,keywords</str>
    <str name="langid.langField">language_s</str>
  </lst>
</processor>
```

Figure 3.2.1: Processor configuration to be added during indexing

Classification

Overview

Sometimes, a particular user may be interested in looking at other users' opinions at a particular restaurant or food in order to make an informed decision. As such, sentiment classification of the comments is essential so that the user can quickly decide whether the particular restaurant is of his/her choice based on the positive, neutral and negative comments.

Question 4

4.1 Information Extraction on crawled corpus

Sentiment analysis is actually a complex task even if we are just dealing with a binary class (positive vs negative comments) because it requires tackling many other subtasks. In this section, we will explore and tackle two subtasks by performing information extraction on the data that we crawled as required in question 1. We will look at:

1. Text Summarization using Deep Learning
2. Sentence Segmentation using POS Tagging

4.1.1 Text Summarization using Deep Learning

Sometimes, we may encounter texts that are very lengthy and we only have a limited amount of time to skim through an article or a webpage. Likewise in our context, users may only be interested in the gist of the content when they seek for restaurant comments made by other users. As such, text summarization is needed in order to keep long descriptions short and sweet.

Text Summarization is the technique for generating a concise and precise summary of large texts while focusing on the sections that convey useful information and retaining its overall meaning. There are two approaches towards Text Summarization, Extractive Summarization and Abstractive Summarization. Extractive Summarization is an approach that identifies the important sentences or phrases from the original text and extracts only those from the text. There will not be new sentences formed. Abstractive Summarization on the other hand generates new sentences from the original texts. The new sentences may not be present in the original text.

In our context, we adopted the Abstractive Text Summarization method to summarize our crawled text using the Sequence-to-Sequence (Seq2Seq) model. We used the Abstractive Text Summarization method because it produces more human-like summaries as compared to Extractive Text Summarization. Seq2Seq consists of the Encoder-Decoder Architecture, which is used to solve the Seq2Seq problems where the input and the output sequences are of different lengths. Our

Encoder-Decoder Architecture consists of 2 phases, the training phase and the inference phase.

Training Phase

Using Long Short-Term Memory (LSTM), the encoder reads the entire input sequence of text comments in the train data. At each timestep, a word is fed into the encoder. For every timestep, it will process and capture the contextual information present in the input text comments. Figure 4.1.1 illustrates an encoder and the above descriptions.

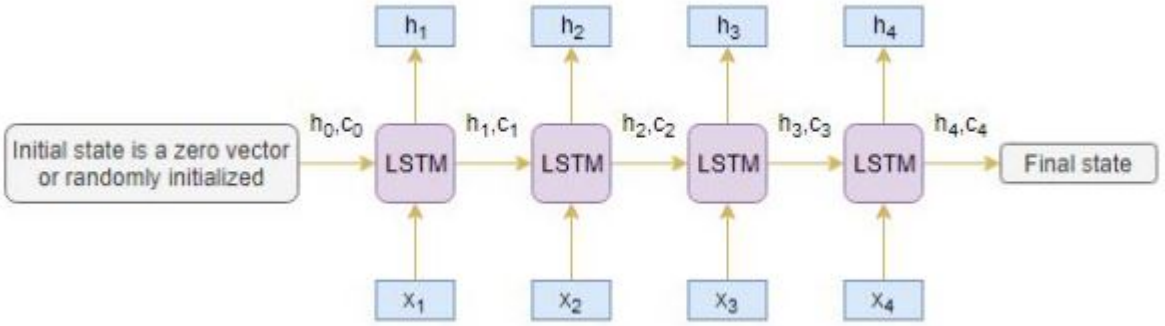


Figure 4.1.1: An Encoder

The decoder is also an LSTM network that takes in the target sequence one word at a time and predicts the same sequence offset by one timestep. It will then be trained to predict the next word in the sequence given the previous word. Figure 4.1.2 illustrates a decoder and the above descriptions.

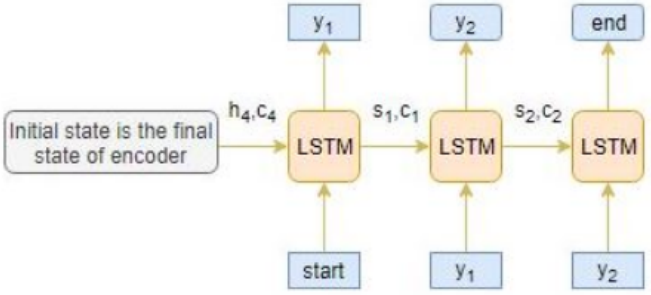


Figure 4.1.2: A Decoder

We will then apply Attention Mechanism into our model to overcome the difficulty for the encoder to memorize long sequences in the fixed length vector. Attention mechanism looks at the specific parts of the sequence that are more important rather than taking into account the whole sequence.

Inference Phase

After the training phase, the model is tested on the crawled data but the summarized sequence is unknown, hence the inference architecture is needed to decode the crawled data.

Our Methodology

We use the ‘Amazon Fine Food Reviews’ dataset as our training data to build our Encoder-Decoder model. Then, we use our crawled data as our test set. Figure 4.1.3 shows the kaggle introduction page of the Amazon Fine Food reviews.

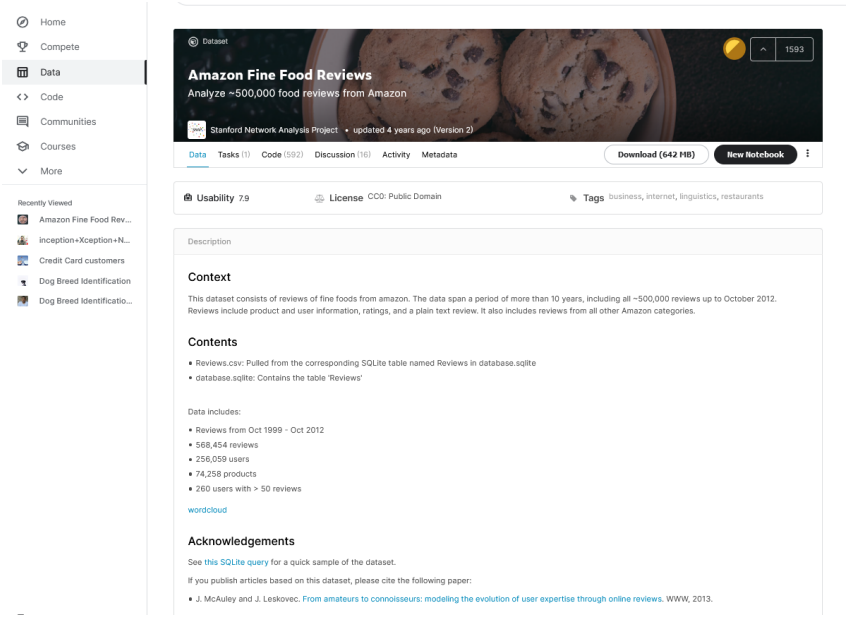


Figure 4.1.3: Kaggle introduction page

We use the ‘Amazon Fine Food Reviews’ dataset as our training data because the dataset consists of a column named ‘Summary’, which is the summarized text for the actual comment in the ‘Text’ column. As such, we make this into a supervised learning task by treating the full text comments as our input and the summary texts as our ‘ground truth’ output to train the model. We will then fit it into our crawled data to get the desired summarized text. Figure 4.1.4 shows the first three entries in the dataset.

```
[ ] # using SQLite Table to read data.
    con = sqlite3.connect(COLAB_FILEPATH + 'data/database.sqlite')
    data_train_raw_ = pd.read_sql_query(""" SELECT * FROM Reviews """, con)
    print(data_train_raw_.shape)
    data_train_raw_.head(3)
```

```
(568454, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...

Figure 4.1.4: Some entries of the ‘Amazon Fine Food Reviews’ dataset

Results

Table 4.1.1 shows five entries of the inference for the summarized text as compared to the ‘ground truth’ from the validation data as well as the cleaned full text.

Full Text: disappointing salty flavor shrimp made wheat rice cracker **Summarized Text:** not shrimp chips **Predicted Summarized Text:** not what expected
Full Text: first bought knowing expect love brown rice since whole grains nutrition excellent flavor wonderful blend basic way use chicken stock broth tbsp butter cooking wonderful highly recommended difficulty finding locally glad find online even larger quantities **Summarized Text:** excellent product **Predicted Summarized Text:** great rice
Full Text: two cats say paws think good also feed wet still seek kibble fix **Summarized Text:** cats love it **Predicted Summarized Text:** my cats love this
Full Text: used make large loaf white bread dinner accompaniment bread tasty even kids know good posts happy pay homemade loaf bread get fresh baked bread smell home gave stars truly think convenience include yeast packets even though would make product dates difficult **Summarized Text:** very easy bread machine mix **Predicted Summarized Text:** great bread
Full Text: stash premium coconut mango wuyi oolong tea right flavored teas flavor extremely disappointed purchased count box even doubling tea bags little flavor spent different flavors stash like throwing money away **Summarized Text:** overall disappointment with stash premium tea **Predicted Summarized Text:** not what expected

Table 4.1.1: Inference of the summarized text from the validation data with comparison to the ‘ground truth’ summarized text

From Table 4.1.1, we can see that although the exact texts might not be the same when comparing the summarized text and the predicted summarized text, the general sentiment polarity of the summary pair is somewhat similar, with the predicted summarized text being more generic and shorter than the given summarized texts. We then move on to fit our crawled text data comment to get an inference for the summarized text. Table 4.1.2 shows five entries of the inference for the summarized text of the crawled data as well as the cleaned full text.

Full Text: tomato maple syrup honestly best thing ever tasted definitely coming back **Predicted Summarized Text:** best syrup ever
Full Text: colleagues client catch lunch well served william tan team good recommendations food items suit everyone taste preference **Predicted Summarized Text:** great for making lunches
Full Text: excellent customer service service recovery chris friendly ensured great time celebrate dad birthday thank **Predicted Summarized Text:** great service
Full Text: food extremely delicious meat tender salmon good excellent service several stuff friendly one stuff nice knew menu well good recommendations would definitely come back **Predicted Summarized Text:** great for making salmon
Full Text: valentines dinner chef clearly given excellent unfortunately one small serve entree dessert persons share seem make good sense given ordered set meal would better **Predicted Summarized Text:** good but not great

Table 4.1.2: Inference of the summarized text from the crawled data

Summary of the Text Summarization steps

Figure 4.1.5 shows the steps we took to implement abstractive text summarization on our crawled corpus.

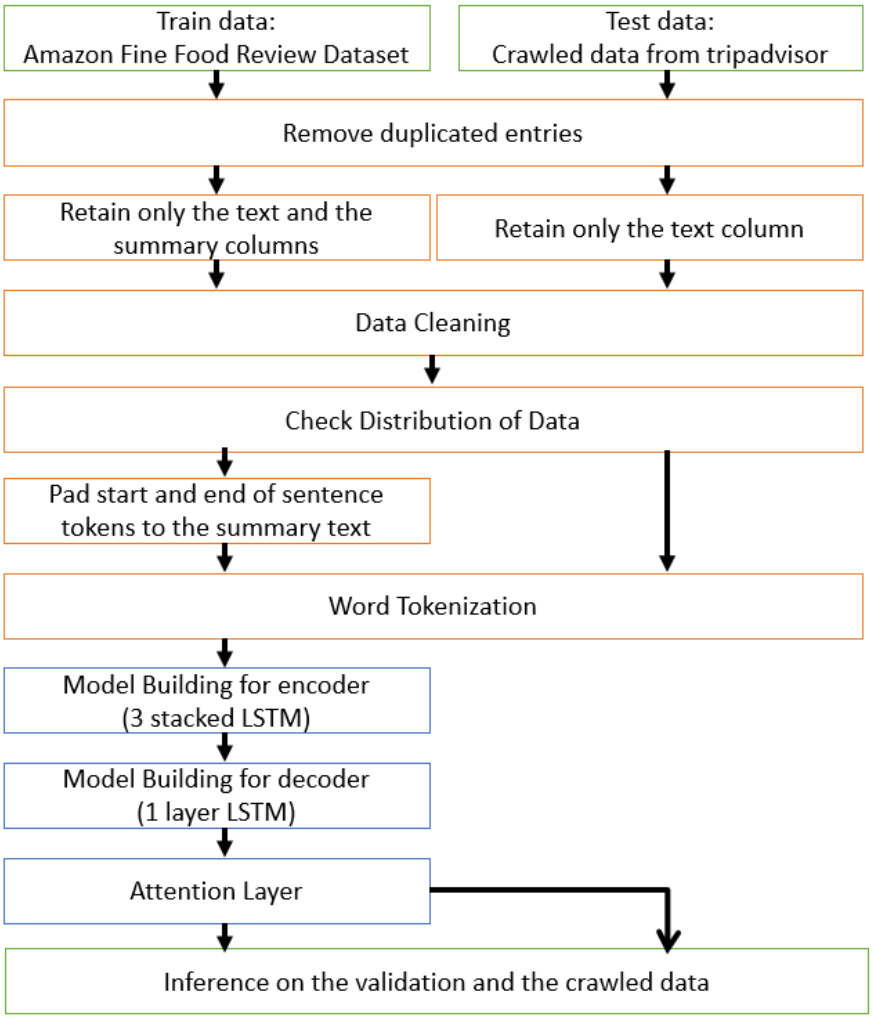


Figure 4.1.5: Steps to implement abstractive text summarization

4.1.2 Sentence Segmentation using POS Tagging

Another technique for information extraction is sentence segmentation. Sentence segmentation is the problem of dividing a string of written language into its component sentences. Usually the sentences are segmented by punctuations. However, in our case, we are interested in extracting the noun, noun phrase, verb and verb phrase from the crawled data using Parts-of-Speech (POS) Tagging.

Results

Table 4.1.3 shows five different entries from the crawled data and the segmented texts.

Full Text: Place has great food, great ambience and the staff are very friendly. Shout out to Naufal and Ain for their pleasant service. **Text Segments:** ['place great food great ambience staff', 'friendly shout', 'naufal pleasant service']
Full Text: the food were really delicious, but the staff made the dining experience even better. really good service and friendly :) **Text Segments:** ['food', 'delicious worth', 'penny', 'definitely recommend', 'friend family']
Full Text: Love their risotto. This is the third time coming here. It won't be the last. Highly recommended if you love Risotto. Truffle Mushroom Risotto is a must try here. **Text Segments:** ['love', 'risotto third time', 'come', 'highly recommend', 'love risotto truffle mushroom risotto', 'try']
Full Text: From food to service to ambience. Everything is great! Must try their pizza. Unique and delicious. 👍🏽👍🏽 **Text Segments:** ['food service ambience everything', 'try', 'pizza']
Full Text: Great food . Love the pasta , milk shake . Staff was attentive and prompt . We had the truffle fries , seafood linguine and the roche milk shake . Would recommend! **Text Segments:** ['great food love pasta milk', 'shake', 'staff attentive prompt truffle fry', 'seafood', 'linguine roche milk shake', 'recommend']

Table 4.1.3: Sentence Segmentation

Summary of the Sentence Segmentation steps

Figure 4.1.6 shows the steps we took to implement sentence segmentation on our crawled corpus.

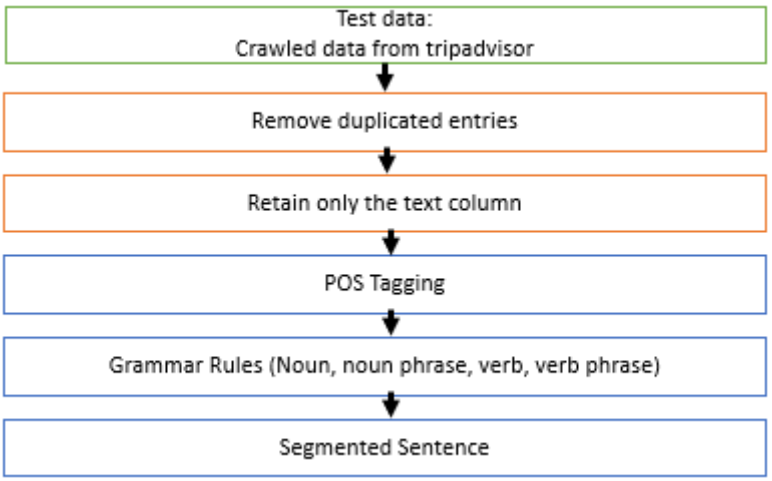


Figure 4.1.6: Steps to implement sentence segmentation

4.2 Motivate the choice of the classification approach in relation with the state of the art

We adopted deep learning approaches for this sentiment classification task. Deep learning learns the information from the data and the representation across multiple levels, in which the lower level corresponds to more general information that can be leveraged by other areas directly or after fine-tuning. Specifically, we will be looking at two types of deep learning architecture in this section, the sequence model architecture and the transformer architecture. We believe that training our models using these 2 architectures will enable us to classify the sentiments of our comments with high accuracy.

4.2.1 Sequence Model Architecture - Long Short-Term Memory (LSTM)

With recurrent neural networks (RNN), deep learning is able to capture the sequence information of the comments we passed as inputs in a much better sense. Long Short-Term Memory (LSTM) is one of the recurrent neural networks that overcomes the long-term dependency problem due to vanishing gradient problem. LSTM is dependent on three things, the current long-term memory of the network (cell state), the output at the previous point of time (hidden state) as well as the input data at the current time step. LSTM uses a series of gates to control the flow of the information. Figure 4.2.1 shows a cell of the LSTM.

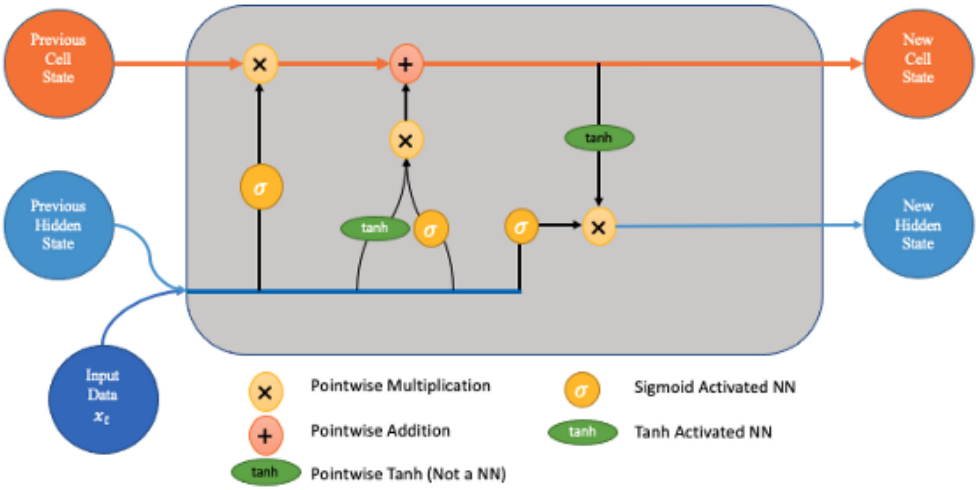


Figure 4.2.1: A cell in LSTM

Putting the brief explanation of LSTM above into our context, the comments we crawled have entries that are about hundred words long. This means that the words may have dependencies between each other in a sentence or a paragraph. This makes the LSTM architectures suitable candidates to build our prediction model for our sentiment analysis task.

Another variant of the LSTM model is the Bi-directional LSTM model. Figure 4.2.2 shows the difference in the outline of the 2 variants of the LSTM Architectures.

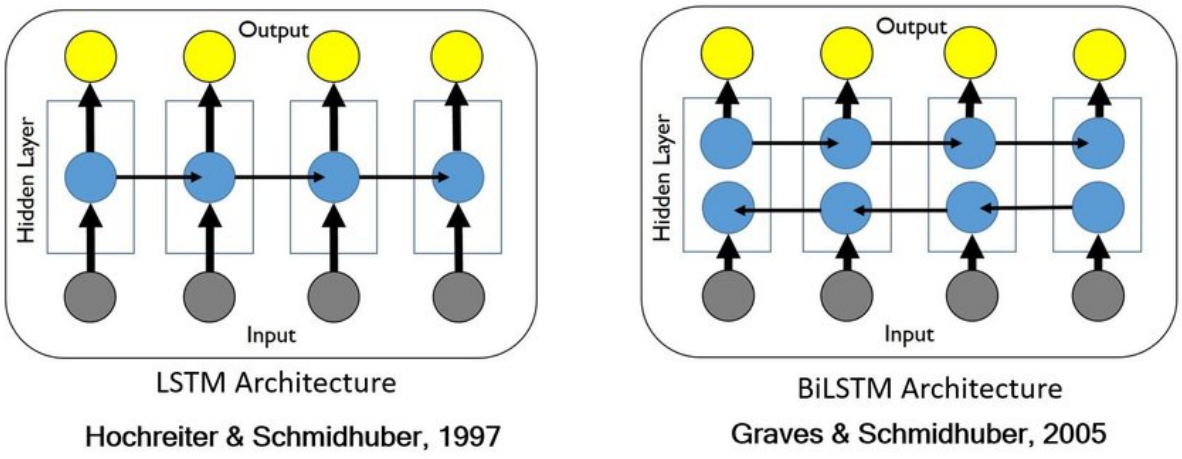


Figure 4.2.2: Unidirectional LSTM versus the Bi-directional LSTM architecture

In contrast to the unidirectional LSTM architecture, the bi-directional LSTM architecture is capable of taking into account not just the words in the past but also words in the future in a sentence for the current prediction. Bi-directional LSTM trains two LSTMs on the input sequence. The first on the input sequence just like the unidirectional LSTM and the second on a reversed copy of the input sequence. This can provide additional context to the neural network and result in a better learning

model on the classification task. As food reviews usually are descriptive in terms of the food, the services and the locations, bi-directional LSTM is capable of providing context for each of the words in the food review comments based on the past and future words, resulting in a better learned model in the contextual point of view.

4.2.2 Transformer Architecture - Bidirectional Encoder Representations from Transformers (BERT)

Introduction to the Transformer Architecture

With LSTM models, dealing with long-range dependencies is still challenging and the sequential nature of the model architecture prevents parallelization. As such we will look at the Transformer architecture to solve these issues.

Transformer aims to solve sequence-to-sequence tasks even with long-range dependencies. It is the model that relies on a self-attention mechanism to convert the input sequences into the output sequences without the need to be sequence aligned such as the LSTM. Self-attention is the mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Figure 4.2.3 shows the Transformer's model architecture.

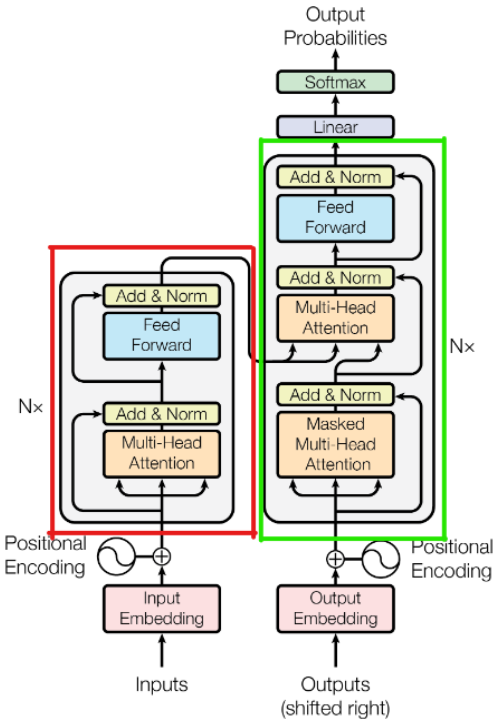


Figure 4.2.3: Transformer's model architecture

From Figure 4.2.3, the encoder portion is represented by the red box and the decoder portion is represented by the green box. The encoder and decoder blocks are actually multiple identical encoders and decoders stacked on top of each other and both have the same number of units. How the whole mechanism works is that the word embeddings of the input sequence are first passed to the encoder and then

propagate to the next encoder. The output of the last encoder stack is then passed to all the decoders in the decoder stack for decoding.

Why BERT?

In relation to the state-of-the-art model in Natural Language Processing (NLP), BERT combines two powerful technologies, it is based on a deep transformer network that is able to read in long texts by using attention and is also bi-directional, which takes into account the whole text passage to understand the meaning of each word. This is especially useful in our context here as we predict the sentiment of the comments from the reviewers because most of the comments are long and descriptive. Figure 4.2.4 shows the architecture of BERT.

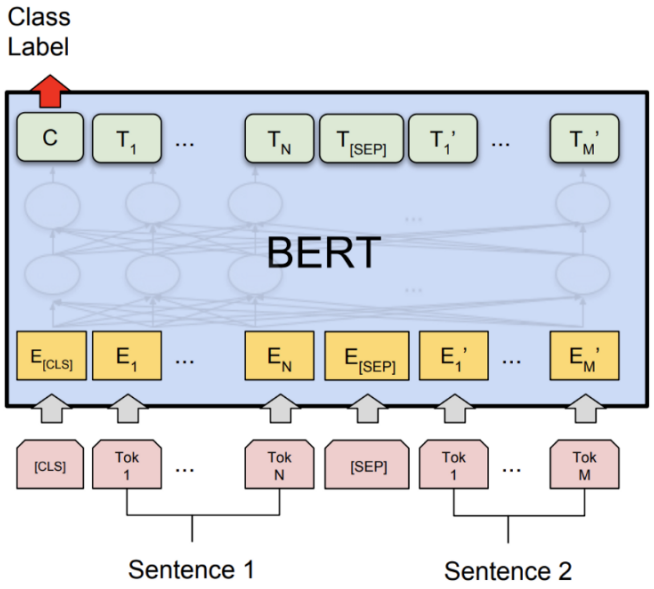


Figure 4.2.4: BERT's architecture

4.2.3 Models used in the classification task

The above analyses (Section 4.2.1 and 4.2.2) gave us the motivation that using the sequence and the transformer architecture will work well for this sentiment analysis task. As such, we trained the following three models and their specifications to make our predictions on the comments crawled. These models are generally trained on the optimal hyperparameters:

1. **Recurrent Neural Network: Unidirectional LSTM**
    a. Embedding vector length: 32
    b. Number of epochs: 5
    c. Learning rate: 0.001
    d. Number of hidden layer neurons: 128
    e. Number of hidden layers: 1
    f. Dropout: 0.2 per layer
    g. Batch size: 512

2. **Recurrent Neural Network: Bidirectional LSTM**
    a. Embedding vector length: 32
    b. Number of epochs: 5
    c. Learning rate: 0.001
    d. Number of hidden layer neurons: 128
    e. Number of hidden layers: 1
    f. Dropout: 0.2 per layer
    g. Batch size: 512
3. **Transformer Architecture: BERT**
    a. Embedding vector length: 100
    b. Learning rate: 2e-5
    c. Epoch: 1
    d. Batch Size: 6

4.3 How we use our data to train our prediction model

Similar to our Text Summarization subtask, we will use the 'Amazon Fine Food Reviews' dataset as our training data to build our models. Then, we will use our crawled data as our test set.

Although our crawled data also have ratings which can be treated as the ground truth, we do not use it for training because the crawled data is much smaller in size as compared to the Amazon dataset. The Amazon dataset has 568,454 reviews which is considerably large as compared to our crawled data with only 97190 reviews. Also, both datasets have the same context which is about food comments and hence, this justifies the choice of choosing Amazon dataset as our training set. The test set will be the data that we had crawled. The aim is to build a model using the Amazon dataset and use that to predict the sentiments of the comments in the crawled data, then use this prediction to compare with the ground truth rating given by the users in the crawled data. We hope to achieve high test accuracy with the models trained on either of the architectures stated in section 4.2.3.

4.4 Pre-processing of the data and the reasons for doing so

We need to pre-process our training data so that our models can be trained in the most efficient way. We will also need to pre-process the test data a little so that it could fit the models trained by the training set.

4.4.1 Importing data to Google Colaboratory

We will first look at the training set which is the Amazon Fine Food Reviews dataset. The dataset consists of 10 unique columns. Figure 4.4.1 shows the first three entries of the dataset and also the column headers.

```
[ ] # using SQLite Table to read data.
    con = sqlite3.connect(COLAB_FILEPATH + 'data/database.sqlite')
    data_train_raw_ = pd.read_sql_query(""" SELECT * FROM Reviews """, con)
    print(data_train_raw_.shape)
    data_train_raw_.head(3)
```

(568454, 10)

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...

Figure 4.4.1: First three entries of the Amazon Fine Food Review dataset

4.4.2 Removal of duplicates

We will first remove all the duplicated entries. Surprisingly after the removal, we are left with only 393933 data entries in the dataset, which is only 69.3% of the original dataset. Removal of duplicates are essential because we do not want our model to put more weights onto a particular duplicated sentence as compared to other non-duplicated sentences, which may affect our model performance. We dropped the duplicates based on the columns UserId, ProfileName, Time and Text as we believe that these columns will determine whether the particular comments are duplicated or not.

4.4.3 Drop the columns that will not be used in the training

We then create a new dataframe that only keeps the Text and the Score column from the non-duplicated dataframe. We rename the column ‘Text’ and ‘Score’ into ‘comments’ and ratings’ respectively. Additionally, we appended a new column ‘type’ into our new dataframe to signify that this dataset is a training data, this column will be used in the later part of the pre-processing. Figure 4.4.2 shows a code snippet of the process of creating the new dataframe with only required data columns.

```
[ ] # create an empty dataframe
    data_train = pd.DataFrame()

    # to store only the required columns into the new dataframe
    data_train['comments'] = data_train_raw['Text']
    data_train['ratings'] = data_train_raw['Score']
    data_train['type'] = "train"
    print(data_train.shape)
    print(data_train.dtypes)
    data_train.head()
```

```
(393931, 3)
comments    object
ratings      int64
type        object
dtype: object
```

	comments	ratings	type
150528	In June I saw a charming group of ro...	4	train
150506	This is a fun way for children to learn their ...	4	train
150505	I grew up reading these Sendak books, and watc...	4	train
150504	Get the movie or sound track and sing along wi...	4	train
150503	A very entertaining rhyming story--cleaver and...	4	train

Figure 4.4.2: New data frame for training with only the required data

Likewise, we did a duplication check and appended a ‘type’ column, but we retained our data entries for the test dataset which is our crawled data at this point of time. This is because some columns might still be needed by the Solr indexing and querying system as explained in section 2 of this report. After removing the duplicates, only 90.6% of the test data are left. Figure 4.4.3 shows a code snippet of the filtered test dataset.

```
[ ]  #data_test_raw = pd.read_csv(COLAB_FILEPATH + 'data/trip-advisor-comments-filtered.csv')
     # create an empty dataframe
     data_test = pd.DataFrame()

     # to store only the required columns into the new dataframe
     data_test['Restaurant Name'] = data_test_raw['Restaurant Name']
     data_test['Restaurant Type'] = data_test_raw['Restaurant Type']
     data_test['Reviewer\'s Name'] = data_test_raw['Reviewer\'s Name']
     data_test['comments'] = data_test_raw['Comment']
     data_test['ratings'] = data_test_raw['Rating']
     data_test['type'] = "test"
     print(data_test.shape)
     print(data_test.dtypes)
     data_test.head()
```

```
(88042, 6)
Restaurant Name    object
Restaurant Type    object
Reviewer's Name    object
comments           object
ratings             int64
type               object
dtype: object
```

	Restaurant Name	Restaurant Type	Reviewer's Name	comments	ratings	type
0	Positano @ RP	Italian, European	aisvslife98	I enjoyed my time here with my girlfriends! Fa...	5	test
1	Positano @ RP	Italian, European	Odyssey44198198885	Wonderful and amazing service experience. Defi...	5	test
2	Positano @ RP	Italian, European	Ninifazelin	Great food and wonderful service! Will definit...	5	test
3	Positano @ RP	Italian, European	Amaliamazlan	Not my first time in Positano and definitely w...	5	test
4	Positano @ RP	Italian, European	Shahzanstim	Excellent service from the staff. The beef was...	5	test

Figure 4.4.3: New data frame for the test set

4.4.4 Partition the ratings into three classes

As our goal in this sentiment classification task is to predict if a particular comment is positive, negative or neutral, we will need to partition our ‘ratings’ column in both of our train and test dataset into just three classes. We will then create a new column in both datasets named ‘ratings_class’. Here is the mapping from ratings to ratings_class:

- If the rating is 1 or 2, ratings_class will be ‘-1’, implying a negative sentiment
- If the rating is 3, ratings_class will be ‘0’, implying a neutral sentiment
- If the rating is 4 or 5, ratings_class will be ‘1’, implying a positive sentiment

Figure 4.4.4 shows the code snippet of how we partitioned our data and the appended ratings_class column in both of our dataframes.

```
[ ] def partition(x):
        if x < 3:
            return -1
        elif x == 3:
            return 0
        else:
            return 1
```

```
[ ] # append partitioned data to the train set
    actualScore = data_train['ratings']
    class_ = actualScore.map(partition)
    data_train['ratings_class'] = class_
    print("Number of data points in train data", data_train.shape)
    data_train.head(3)
```

```
Number of data points in train data (393931, 4)
```

	comments	ratings	type	ratings_class
150528	In June I saw a charming group of ro...	4	train	1
150506	This is a fun way for children to learn their ...	4	train	1
150505	I grew up reading these Sendak books, and watc...	4	train	1

```
[ ] # append partitioned data to the test set
    actualScore = data_test['ratings']
    class_ = actualScore.map(partition)
    data_test['ratings_class'] = class_
    print("Number of data points in test data", data_test.shape)
    data_test.head(3)
```

```
Number of data points in test data (88042, 7)
```

	Restaurant Name	Restaurant Type	Reviewer's Name	comments	ratings	type	ratings_class
0	Positano @ RP	Italian, European	aisvslife98	I enjoyed my time here with my girlfriends! Fa...	5	test	1
1	Positano @ RP	Italian, European	Odyssey44198198885	Wonderful and amazing service experience. Defi...	5	test	1
2	Positano @ RP	Italian, European	Ninifazelin	Great food and wonderful service! Will definit...	5	test	1

Figure 4.4.4: Dataframes after partitioning

4.4.5 Separation of the neutral data from opinionated data

From both of our train and test datasets, we found out that there are neutral data. As such we separate the neutral data from the opinionated ones, simply by splitting a dataframe into two based on the ratings_class column. A dataframe with ratings_class '0' and another with ratings_class '1' or '-1' will be produced for both the train and test data (a total of 4 dataframes). As there may be bias in the ratings by the users as according to their comments given, we look at a few of the entries in both the 'neutral' dataframe and decide whether the data is really neutral or not based on inter-annotator agreements. After the separation, we noted that there are 29772 and 8837 neutral data entries from the train and the test set respectively. Below shows some of the comments that are marked as 'neutral' according to the ratings_class:

Neutral train data

- This book was purchased to be used in a classroom setting for read-a-loud purposes. The small size of the book(4X6)makes that almost impossible. The book itself is very cute and flows nicely, but is best shared with only one child at a time.
- My dog enjoys these treats in moderation. Because they are dried, he seems to enjoy them less the more he eats them. With water, the treats may be much more flavorful. After having a fair amount, he starts to lose interest in them until he can drink.

This brings me to a question regarding serving size. The container recommends 2-3 cubes per day for a medium breed. The issue I'm having is that the size of the cubes is so very inconsistent.

Some bits are the size of a pea while others of a grape. The rest are a variety of sizes in between. Some consistency would be preferred.

Also, this product comes highly recommended for training because it is low in fat. My dog certainly won't learn anything with just 2-3 of these per day. Like most dogs, he is highly motivated by food so 20-30 treats is more accurate. To make this possible, I have to dice them up into smaller pieces or shift through to find the little bits.

At least these treats are natural and thus healthier than some of the other options. I just didn't expect to have to prepare them first.

- I bought this set because my niece was obsessed with having a black Easter egg (kids!) long story short, even though I used an entire bottle the coloring of the eggs were not vibrant. Not sure how they would work for baking so that's why they are still getting 3 stars.

Neutral test data

- We visited Entre-Nous creperie just now, not entirely crowded with decent crowd but we waited almost 40 minutes for our crepes, one banana and another salted butter caramel to be served. I counted 3 chefs in the kitchen and one lady waitress serving.
- Conclusion, the Food was not bad and service was good. However, there was a birthday function held at the general dinning area (not in a private room as there were many tables) and it was very noisy. Would have given a 4 star if not of this .
- I was under the illusion that there is no need to make reservationsoo when eating out because of covid 19 restrictions.I was very wrong. On thu I tried to make reservations at Koma for two for dinner the next day. First available time is 9.45pm .

From the above comments, we can see that the data entries are generally of neutral sentiments.

4.4.6 Specification of the training model

For our model, we exclude the neutral data during training and only train on the opinionated data. Likewise, we remove the neutral data from the crawled dataset as well. As such, this will be a binary classification problem with positive and negative (‘1’ and ‘-1’) as the class labels for polarity detection. We will move on to look at the opinionated data.

4.4.7 Data imbalance issue in train data and using downsampling to solve it

Using value_counts() on the ratings_class column, we can see how many entries are present in each of the ratings_class in the train data:

- 307052 entries with ratings_class labelled as ‘1’
- 57107 entries with ratings_class labelled as ‘-1’
- 29772 entries with ratings_class labelled as ‘0’

From the above figure, we can see that the proportion of the number of entries with ratings_class ‘-1’ as compared to ratings_class ‘1’ is only 0.18598. This is a serious data imbalance issue. If we were to build our model with all of these opinionated data, our classifier will simply be lopsided towards the positive (‘1’) class since the positive data is 5.38 times more than the negative (‘-1’) data. When we fit our test data into our model, the model will not be generalised and hence causing inaccurate results on the test set.

As such, we downsampled the data that has the ratings_class of ‘1’ to the number of data that has the ratings_class of ‘-1’. That is, to sample 57107 entries out of the

307052 entries of the ratings_class ‘1’ so that the number of data in both classes are equal. We will first retrieve the entries from the main dataframe that contains the ratings_class ‘1’ and ‘-1’ separately. Then, we randomly sampled 57107 entries from the train dataset. Figure 4.4.5 shows the code snippet of how it is done.

```
[ ]  # data with class -1
     data_negative_train = data_train[data_train['ratings_class'] == -1]
     print(data_negative_train.shape)
     data_negative_train.head(3)
```

(57107, 4)

	comments	ratings	type	ratings_class
150496	I give five stars to the Maurice Sendak story....	1	train	-1
150525	This is one of the best children's books ever ...	1	train	-1
24749	My dogs loves this chicken but its a product f...	2	train	-1

```
[ ]  # data with class 1, sample with the total number of entries in the negative class (downsampling)
     data_positive_train = data_train[data_train['ratings_class'] == 1].sample(len(data_negative_train), replace=False)
     print(data_positive_train.shape)
     data_positive_train.head(3)
```

(57107, 4)

	comments	ratings	type	ratings_class
198912	I was looking for Mag Citrate in a fast-acting...	5	train	1
231943	My daughter is egg, soy, peanut, tree nut, ses...	5	train	1
415336	This drink ***SLURP*** is soo good! I am drink...	5	train	1

Figure 4.4.5: Downsampling the positive class entries from the train dataset

After the downsampling of the positive class entries, we proceed to concatenate the positive entries and the negative entries. Then, we randomised the order of the dataframe to produce our final dataframe for the next phase of data pre-processing. The randomisation process is necessary because we want the model to be more robust during the training process. Figure 4.4.6 shows the code snippet of how the final dataframe for the next phase of data preprocessing is done.

```
[ ]  # concatenate the positive and negative data into a new dataframe
     # this dataframe will be the training set (amazon dataset) for our test set (crawled corpus)
     data_train_undersampled_ = pd.concat([data_negative_train, data_positive_train])
     # randomise the order of the sampled dataframe
     data_train_undersampled = data_train_undersampled_.sample(frac=1)
     print(data_train_undersampled.shape)
     data_train_undersampled.head()
```

(114214, 4)

	comments	ratings	type	ratings_class
214595	This maple syrup was very good. I was very di...	2	train	-1
97297	I cant believe my favorite type of chocacolate...	5	train	1
46253	The noodles were better than I thought instant...	2	train	-1
228596	I think I ordered the wrong brand. Peter Pan c...	2	train	-1
102453	My bulldog does so well on this food, I doubt ...	5	train	1

Figure 4.4.6: Forming the final dataframe

Likewise for the test set, we separated the neutral data from the opinionated ones and only use the opinionated one to get the prediction after the model is being trained.

4.4.8 Merging the data

For the ease of data cleaning and tokenization of data, we merge the preprocessed train and test data into one single dataframe (named 'data_overall') first. We will later separate it again before fitting it into our model. Figure 4.4.7 shows a code snippet of how we merge our dataframes and how we find the splitting point between the train and the test dataset.

```
[ ]  # merge the 2 dataframes together to perform tokenization
     frames = [data_train_undersampled,data_opinionated_test]
     data_overall = pd.concat(frames)
     # check the dimension of the merged dataframe
     print(data_overall.shape)
     data_overall.head(3)
```

```
(193419, 4)
```

	comments	ratings	type	ratings_class
214595	This maple syrup was very good. I was very di...	2	train	-1
97297	I cant believe my favorite type of chocacolate...	5	train	1
46253	The noodles were better than I thought instant...	2	train	-1

```
[ ]  # check the range where the data changes from train set to test set
     # debug
     #data_overall.iloc[:,:]
     # finding the split
     data_overall.iloc[train_data_length-2:train_data_length+2,:]
```

	comments	ratings	type	ratings_class
534782	I only ordered this MFG brand of lingonberry b...	1	train	-1
48580	I got a sample of this in my Summer VoxBox and...	5	train	1
0	I enjoyed my time here with my girlfriends! Fa...	5	test	1
1	Wonderful and amazing service experience. Defi...	5	test	1

Figure 4.4.7: Merging the dataframes to form data_overall

4.4.9 Data cleaning and Lemmatization

The data that we used might contain characters, punctuations or stop words that are not useful in our sentiment classification tasks. As such, we can use regular expressions (regex) and the python library nltk to clean our merged dataframe. Below shows the steps that we imposed to clean our data.

1. Remove contractions
2. Remove websites
3. Remove tags (e.g
)
4. Remove all the escape characters (e.g newline \n)
5. Make all the text into lowercase

In addition to that, we then apply lemmatization to the 'comments' columns. Lemmatization in NLP refers to doing things properly with the use of a vocabulary

and morphological analysis of words, normally aiming to remove inflectional endings only to return the base or dictionary form of the word known as lemma. We used Lemmatization as compared to Stemming because Lemmatization always returns the root form of the word after running some algorithms, whereas Stemming just stems the last few characters of a word, which may sometimes lose the root form of the word. The root form of the word is of importance in this sentiment classification task because if the word is stemmed wrongly, it may affect the model's prediction. Therefore, using Lemmatization is a safer option here even though the processing time is longer than Stemming. We use the nltk library to do Lemmatization in this task. Table 4.1.1 shows some entries of the comments before and after applying data cleaning, and Lemmatization.

Before	After	Type of data
This maple syrup was very good. I was very disheartened to find mold in the bottom of the bottle on my last serving. I didn't think mold formed in maple syrup. It was gross	maple syrup good disheartened find mold bottom bottle last serving did not think mold formed maple syrup gross	Train
I cant believe my favorite type of chocacolate is being sold at such a low price. I've checked other websites and this website and 4 aero bars cost 34.95$ and i knew that i could find it for a lower price. This is an amazing deal for one of the best chocolate bars around. I highly recommend that you order at least 5 of these NOW!!!!!!!	cant believe favorite type chocacolate sold low price ihave checked website website aero bar cost knew could find lower price amazing deal one best chocolate bar around highly recommend order least	Train
Nice and cozy atmosphere. Main menu could be more extensive but the food was excellent and so were the drinks	nice cozy atmosphere main menu could extensive food excellent drink	Test
Everything was PERFECT! The cocktails were incredible, the food was beautiful, the flavours were marvelous. The food is really tasty and the staff is adorable. Donâ€™t miss it, youâ€™ll be in for some great experience!	everything perfect cocktail incredible food beautiful flavour marvelous food really tasty staff adorable miss great experience	Test

Table 4.1.1: Before and after applying data cleaning and Lemmatization

4.4.10 Tokenization of data - only for LSTM and Bidirectional LSTM model

As the computer works better with numerical data, we proceed to tokenize the words in the comments column into an array of numbers. Figure 4.4.8 shows an example of how the comment is being represented as a list of numbers.

```
[ ]  # check the tokenized representation
     print(data[train_count-1], end= ' ')

     [75, 978, 1151, 9336, 1471, 709, 14, 1731, 1355, 3685, 970, 265, 346, 413, 303, 94, 180, 201, 1677]
```

Figure 4.4.8: Tokens of a sentence

The tokenization step is not needed if the model is trained using BERT because BERT will automatically deal with the word tokenization on its end.

4.4.11 Split the data into train and test set again

After tokenization of the data, we split the dataset_overall into train and test set again, by indexing based on the training dataset count. Figure 4.4.9 shows how it is being done. '_oh' in 'Y_train_oh' and 'Y_test_oh' stands for the one hot representation of the ratings_class.

```
[ ]  # splitting of data
     X_train, X_test = data[:train_count], data[train_count:]
     Y_train, Y_test = y[:train_count], y[train_count:]
     Y_train_oh , Y_test_oh = y_oh[:train_count], y_oh[train_count:]
```

Figure 4.4.9: Split the data into train and test set

4.4.12 Padding of the input sequences for LSTM and Bidirectional LSTM

To train the LSTM models, the input must be of exact length. To consider between the comments with a lot of words and the resource limitations on the RAM, we set the max review length of the comments in both datasets to be 100 tokens. Any comments less than 100 tokens will be padded with zeros. Any comments more than 100 words will be truncated (rarely will have comments entries having more than 100 tokens). Figure 4.4.10 shows how the padding is being done on one of the entries.

```
[ ]  # padding input sequences
     max_review_length = 100
     X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
     X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

     print(X_train.shape)
     print(X_train[2000])
```

```
(114214, 100)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0  195  211  169  164  532  547  862 1890   89 1951
 1296  479   90   56    9  185  466  238   28    1  965   77  724   68
    9 4089]
```

Figure 4.4.10: Padding of a comment entry with less than 100 tokens

4.4.13 Summary of the data pre-processing steps

Figure 4.4.11 summarizes the steps we took to pre-process the data as inputs to the LSTM, Bidirectional LSTM and the BERT model.

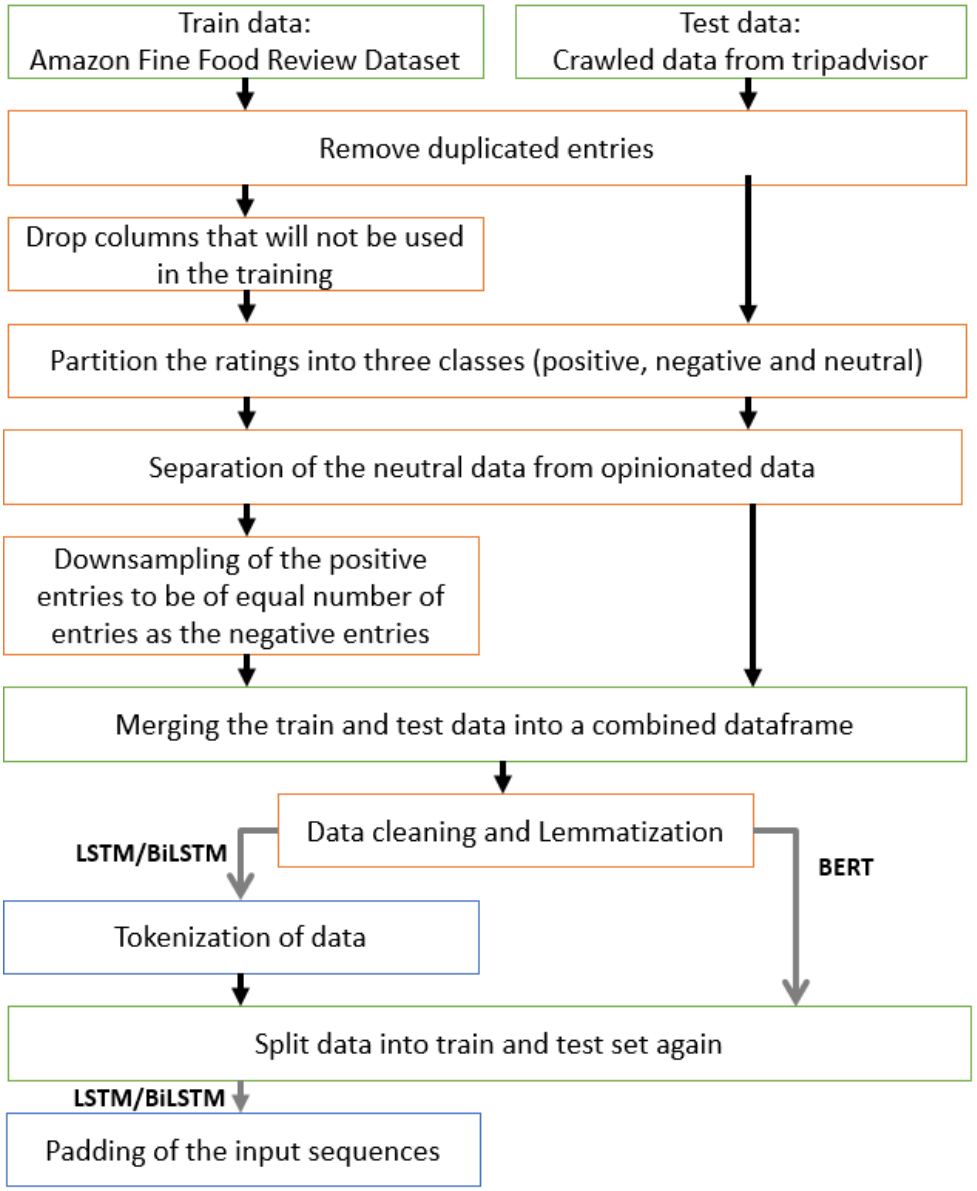


Figure 4.4.11: Steps to pre-process the data as inputs right before training the models

4.5 Evaluation metrics

4.5.1 Evaluation on the validation data

We split our train data into training and validation sets with a ratio of 80:20 and start training the models on three different variations. Table 4.5.1 shows the training loss, training accuracy, validation loss and validation accuracy of the 3 different variations.

Model Architecture	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
Unidirectional LSTM	0.1598	0.9403	0.3027	0.8850
Bidirectional LSTM	0.1630	0.9394	0.2907	0.8817
BERT	0.3221	0.8561	0.1857	0.9278

Table 4.5.1: Models' performances on the validation set

Clearly, we can see that the inputs worked best with BERT as the validation loss is the lowest and the validation accuracy is the highest among all three models trained.

4.5.2 Definitions on some of the evaluation metrics

Since we have the 'ground truth' of the crawled data (test set), we will perform more analysis, such as the test accuracy, F1-score, confusion matrix and AUC-ROC, on the crawled data shown in the next section.

Test accuracy refers to how much of the entries are predicted correctly against the ground truth labels in the test dataset after passing it into the trained model. F1-score is a measure of a model's accuracy by combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall. In our context, the precision is defined as that of all the reviewers' comments that were labelled as 'positive', how many were actually 'positive'? The recall is defined as that of all the reviewers' comments that are actually 'positive', how many did we labelled as 'positive'. Similar definitions are drawn on the 'negative' comments as well. The F1-score is defined as:

$$F1 = (2*Precision*Recall)/(Precision+Recall)$$

Confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known. AUC-ROC (Area Under the Curve - Receiver Operating Characteristics) is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree of separability. It tells how much the model is capable of distinguishing between classes. In our context, the higher the AUC, the better the model is predicting the comments are positive when the ground truth is actually positive and predicting the comments are negative when the ground truth is actually negative.

4.5.3 Evaluation on the crawled data (test data)

Test Accuracy, F1-score and AUC

We sent the pre-processed crawled data for prediction using the models that we have trained. Table 4.5.2 shows the test accuracy of the crawled data, F1-score and the Area Under Curve (AUC) on the three trained models.

Model Architecture	Test Accuracy	F1-score (Macro Average)	F1-score (Weighted Average)	AUC
Unidirectional LSTM	0.9186	0.77	0.92	0.7464
Bidirectional LSTM	0.9199	0.77	0.92	0.7496
BERT	0.9492	0.85	0.95	0.8341

Table 4.5.2: Models' performances on the test set

As seen from Table 4.5.2, the F1-score has two variations, the macro average and the weighted average. In our context, macro average computes the F1-score for each label (positive and negative) and returns the average without considering the proportion for each label in the dataset. Weighted average computes the F1-score for each label and returns the average considering the proportion for each label in the dataset. Since the proportion of the comment entries labelled as 'positive' is a lot larger than the comment entries labelled as 'negative', it will be more appropriate to look at the F1-score for the weighted average to make our comparison.

Confusion Matrix

Table 4.5.3, 4.5.4 and 4.5.5 shows the confusion matrix when the trained model is being predicted on the test set using Uni-directional LSTM, Bidirectional LSTM and BERT respectively.

Actual \ Predicted	Negative '-1'	Positive '1'
Negative '-1'	4406	2519
Positive '1'	3932	68348

Table 4.5.3: Confusion Matrix when model is trained using Uni-directional LSTM on the test set

Actual \ Predicted	Negative '-1'	Positive '1'
Negative '-1'	4292	2633
Positive '1'	3711	68569

Table 4.5.4: Confusion Matrix when model is trained using Bi-directional LSTM on the test set

Actual \ Predicted	Negative '-1'	Positive '1'
Negative '-1'	5242	1683
Positive '1'	2337	69943

Table 4.5.5: Confusion Matrix when model is trained using BERT on the test set

AUC-ROC plot

Figure 4.5.1 shows the AUC-ROC plot of all the three models.

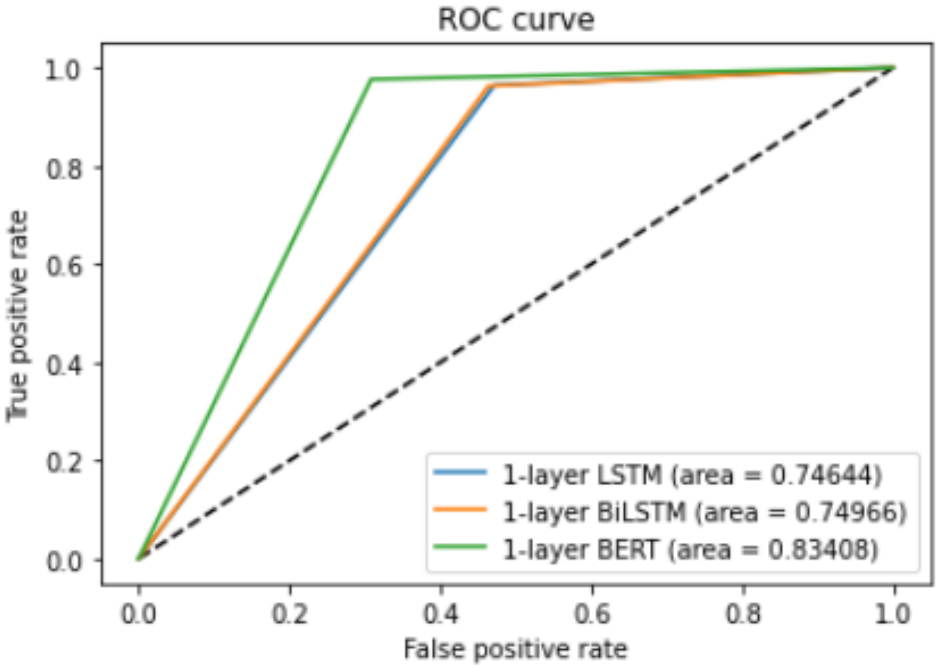


Figure 4.5.1: AUC-ROC Plot for the three different models

4.5.4 Conclusion on the models

Having analysed all the metrics used in the three variations of models trained, we can see that in general, the state-of-the-art NLP model, BERT, did the best in predicting the sentiment class ('positive' or 'negative') that the comments belong to as against the ground truth label in the crawled data.

4.6 Building an evaluation dataset by manually labelling 1000 records and determine its inter-annotator agreement percentage

There may be some biases when it comes to rating a comment. Some comments that appear to be positive to one may appear neutral or negative to another and vice versa. Depending on how each user reads the comments and how he/she detects the indirectness in it such as sarcasm, there are bound to be disagreements in giving the rating.

In this section, we want to find out how two annotators will annotate a 1000 entries of data sampled from the crawled data. The labels will be either positive (‘1’), neutral (‘0’), or negative (‘1’). We aimed to have an inter-annotator agreement of at least 80% as measured by the cohen-kappa score metric. Inter-annotator agreement is a measure of how well two annotators can make the same annotation decision for the three labels as described (‘1’, ‘0’ or ‘-1’). Cohen-kappa is a statistic that is used to measure the inter-annotator agreement.

4.6.1 How we sampled the 1000 entries from the crawled data

In the section on data pre-processing, where we remove duplicates, separate our data as neutral and opinionated and partitioning of data, we will do likewise here. As such, we look at the ratings_class, and randomly sampled 500 entries belonging to the ‘0’ (neutral) class, 250 entries belonging to the ‘1’ (positive) class and another 250 entries belonging to the ‘-1’ (negative) class to make a total of 1000 samples. Figure 4.6.1 shows a code snippet of how we sampled our data.

```
# sample 500 neutral comments
data_zero = data[data['rating_class_main'] == 0].sample(500, replace=False)
print(data_zero.shape)
data_zero.head(3)
```

```
(500, 6)
```

	Restaurant Name	Restaurant Type	Reviewer's Name	Rating	Comment	rating_class_main
44414	Salt tapas & bar	Bar, Mediterranean	MelbourneFrog69	3	We went there for a quick bite and a drink. Th...	0
80817	Gudetama Cafe	Cafe	Wiwan	3	This is our 2nd visit. We came not long after ...	0
62612	The Coastal Settlement	Bar, Diner	AlvinSohYK	3	Nice relax ambience tucked away in a quiet par...	0

```
# sample 250 negative comments
data_negative = data[data['rating_class_main'] == -1].sample(250, replace=False)
print(data_negative.shape)
data_negative.head(3)
```

```
(250, 6)
```

	Restaurant Name	Restaurant Type	Reviewer's Name	Rating	Comment	rating_class_main
57955	Raffles Courtyard	Bar, European	Mgn12	1	Major disappointment. I booked for 7 pm. The s...	-1
68342	Bee Cheng Hiang	Asian, Singaporean	Aw1009	1	I bought 500g minced pork bakkwa from your Nor...	-1
81583	Dim Sum Haus	Chinese, Asian	anastasiabY320YC	1	A restaurant that does not treat their custome...	-1

```
# sample 250 positive comments
data_positive = data[data['rating_class_main'] == 1].sample(250, replace=False)
print(data_positive.shape)
data_positive.head(3)
```

```
(250, 6)
```

	Restaurant Name	Restaurant Type	Reviewer's Name	Rating	Comment	rating_class_main
59401	Gokul Vegetarian	Indian, Asian	Lucy J	4	I had masala dosa here, which was so tasty.a s...	1
57603	Southbridge	Bar, Fusion	Lars R	5	Good food and superb drinks in a small hideawa...	1
18016	Garibaldi Italian Restaurant & Bar	Italian, European	shirley2h	5	These were the 2nd time i visited the restaura...	1

Figure 4.6.1: Sampling of data from the crawled dataset

After sampling, we proceed to concatenate the three dataframes into one, then shuffle the rows of the final dataframe such that the entries are evenly distributed over every index. Figure 4.6.2 shows a code snippet of how it is done.

```
# fit new dataframe to form the final sample
data_sample_ = pd.concat([data_zero, data_negative, data_positive])
print(data_sample_.shape)
data_sample_.head()
```

```
(1000, 6)
```

	Restaurant Name	Restaurant Type	Reviewer's Name	Rating	Comment	rating_class_main
45485	The Refinery Restaurant & Bar	Asian, Bar	jadeylf	3	The Refinery is located in a fun and relaxing ...	0
85514	The Soup Spoon	$	PKA3300	3	Avoidable food. Avoidable service. \n\nWe went...	0
23993	Bread Street Kitchen	$	OSwander	3	We visited in January 2020 after reading about...	0
24900	Lagnaa...barefoot dining	Indian, Asian	timothy k	3	Dinner for four, we all considered the food to...	0
13563	The Song of India	Indian, Asian	pradeepc406	3	Hi \n\nWe were a group of 7 people and I was c...	0

```
# randomise the order of the sample dataframe
data_sample = data_sample_.sample(frac=1)
data_sample.head()
```

	Restaurant Name	Restaurant Type	Reviewer's Name	Rating	Comment	rating_class_main
11970	CE LA VI	Asian, Fusion	mmuazhiim	5	Situated on the top of Marina Bay sands this r...	1
39026	I am...	Dutch, European	PCSim	5	Only learnt of this restaurant from my Friend....	1
16109	The National Kitchen by Violet Oon at the Nati...	Asian, Singaporean	otnielaldi	5	Great restaurant ambience, with much attention...	1
72948	Ginger Thai	Thai, Bar	Sharijer	3	Giving it this rating, because there was no di...	0
51623	Brotzeit (Raffles City)	German, Bar	Mglap	2	Visited the place as we need to eat some weste...	-1

Figure 4.6.2: Sampling of data from the crawled dataset

4.6.2 Cohen-kappa Score

After annotating the data we achieved an inter-annotator agreement score (cohen-kappa score) of 83.50%. This implies that our two annotators can make quite similar annotation decisions for the three labels as described.

4.6.3 Confusion Matrix

We are also curious about the distribution where the remaining 16.50% that the annotators have disagreements. Hence, we plot the confusion matrix for analysis. Table 4.6.1 shows the confusion matrix of the two annotators.

Annotator 1 \ Annotator 2	Negative (‘-1’)	Neutral (‘0’)	Positive (‘1’)
Negative (‘-1’)	347	22	0
Neutral (‘0’)	23	308	45
Positive (‘1’)	0	19	236

Table 4.6.1: Confusion Matrix of the two annotators to see their agreement/disagreement in their annotations

From Table 4.6.1, we can see that there are quite a fair amount of disagreements in each of the three different classes. We can see that the most number of disagreements in a particular possible combination of class labels is 45, where Annotator 1 thinks that the comments are neutral but Annotator 2 thinks that the comments are positive.

4.7 Discuss performance metrics

Table 4.7.1 shows how long the models take to load, how long the records are classified per record and how many records classified per second.

Model	Load trained model (s)	Time taken to classify per record (s)	Records classified per second (s)
LSTM	0.4349	1.068*10^-4	9362
BiLSTM	1.308	1.717*10^-4	5825
BERT	36.47	7.467*10^-3	134

Table 4.7.1: Performance metrics

There are many other ways to improve the performance of our models. Here are some of the ways:

1. Get more data
2. Rescale data
3. Feature Selection

Question 5

5.1 Explore some innovations for enhancing classification

We attempted a few innovations to enhance our classification model. We explored:

1. Aspect-based Sentiment Analysis (ABSA)
2. Name Entity Recognition (NER)

5.1.1 Aspect-based Sentiment Analysis (ABSA)

ABSA is a text analysis technique that categorizes data by aspect and identifies the sentiment attributed to each one. In our context, ABSA can be used to analyze the comments of the users by associating some targeted sentiments with different aspects of the food or the service. We can enhance classification here as there may be instances where there are two differing opinions from a single user on two different aspects, such as he/she may find the food quality is good (positive) but the customer service is bad (negative) as a single entry. We aim to note down the different aspects given by the users and its corresponding sentiment for that particular aspect.

Table 5.1.1 shows three examples of the full text entries and how ABSA is applied to these texts to get an Aspect-Sentiment Relationship.

Full Text: Food is nice and service is great My family enjoyed the dinner and the services rendered We have ordered Peking duck crispy roasted chicken prawns vegetables and other special recommendations by the staff Food served are fresh and most importantly no MSG. **Aspect-Sentiment Relationship:** [['food', ['nice']], ['service', ['great']], ['family', ['enjoyed']], ['dinner', ['enjoyed']], ['peking', ['vegetables']], ['duckcrispy', ['vegetables']]]
Full Text: Food quality is very good for both dishes and dim sum Staff are warm and provide excellent service **Aspect-Sentiment Relationship:** [['foodquality', ['good']], ['staff', ['sum', 'warm']], ['provide', ['service']], ['service', ['excellent', 'provide']]]
Full Text: Went for lunch to celebrate my momis birthday Good food nice ambience excellent service by Manager Ruel. **Aspect-Sentiment Relationship:** [['lunch', []], ['celebrate', ['birthday', 'ambience']],

['momis', ['birthday']],
['birthday', ['momis', 'celebrate']],
['food', ['good', 'ambience']],
['ambience', ['food', 'nice', 'celebrate']],
['service', ['excellent']],
['managerruel', []],
['team', []]]

Table 5.1.1: Applying ABSA on three of the sampled crawled data

Summary of the steps taken for ABSA

Figure 5.1.1 summarizes the steps we took to implement ABSA.

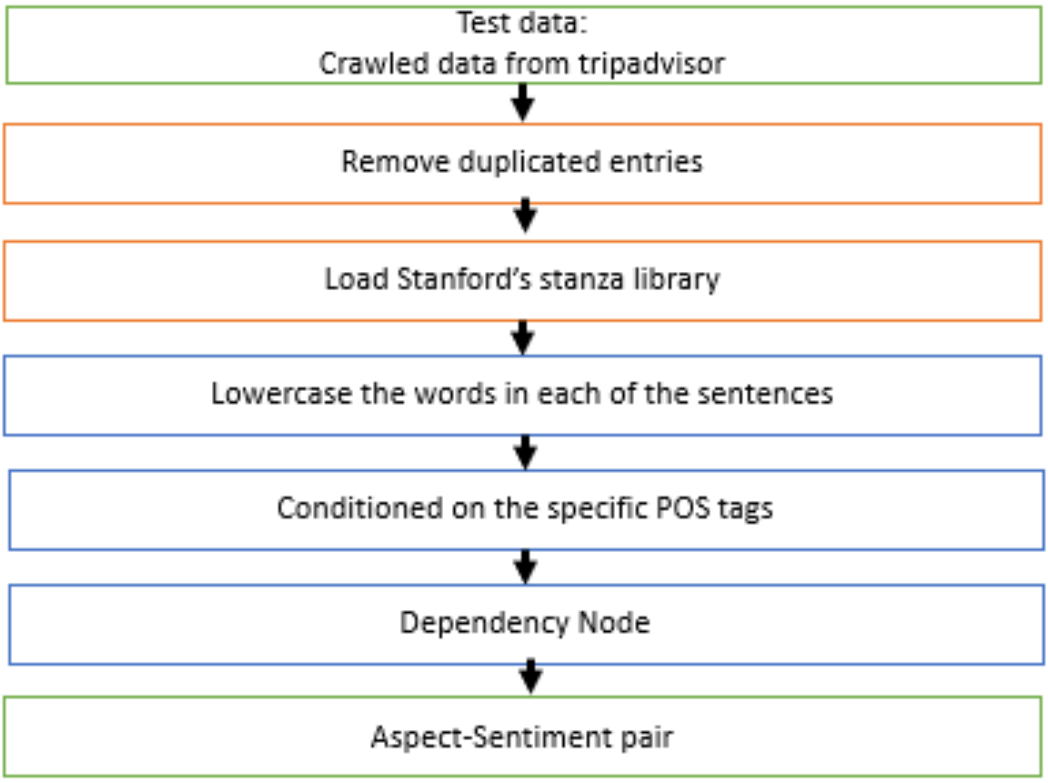


Figure 5.1.1: Steps taken to implement ABSA

5.1.2 Named Entity Recognition (NER)

NER is a technique that seeks to detect and classify named entities in text into categories such as the name of a person, locations, time etc. In our context, we will build our NER system with SpaCy. We can enhance classification here because we are able to detect entities in the crawled corpus.

We also did Inside-outside-beginning (IOB) tagging for the words in our crawled data entries. IOB tagging is a common chunking format. These tags are similar to parts-of-speech tagging (POS tagging) but it can also denote the inside, outside or beginning of a chunk. Multiple chunk phrase types are allowed. Figure 5.1.2 shows a particular entry from the crawled data that has been IOB-tagged.

```
[('Place', 'NN', 'B-NP'),
 ('has', 'VBZ', 'O'),
 ('great', 'JJ', 'B-NP'),
 ('food', 'NN', 'I-NP'),
 (',', ',', 'O'),
 ('great', 'JJ', 'B-NP'),
 ('ambience', 'NN', 'I-NP'),
 ('and', 'CC', 'O'),
 ('the', 'DT', 'B-NP'),
 ('staff', 'NN', 'I-NP'),
 ('are', 'VBP', 'O'),
 ('very', 'RB', 'O'),
 ('friendly', 'RB', 'O'),
 ('.', '.', 'O'),
 ('Shout', 'NN', 'B-NP'),
 ('out', 'IN', 'O'),
 ('to', 'TO', 'O'),
 ('Naufal', 'NNP', 'O'),
 ('and', 'CC', 'O'),
 ('Ain', 'NNP', 'O'),
 ('for', 'IN', 'O'),
 ('their', 'PRP$', 'O'),
 ('pleasant', 'JJ', 'B-NP'),
 ('service', 'NN', 'I-NP'),
 ('.', '.', 'O')]
```

Figure 5.1.2: An example from the crawled data that has been IOB-tagged

Then, we use spaCy to render the named-entity on our crawled text. Figure 5.1.3a, b, c, d and e illustrates five examples from the crawled data on how NER is done.

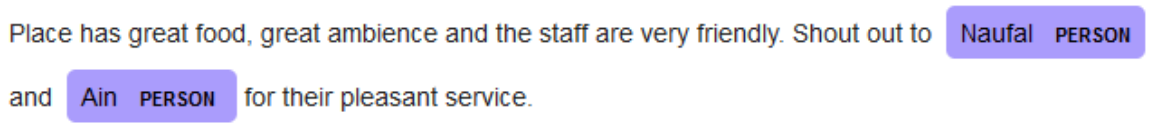


Figure 5.1.3a

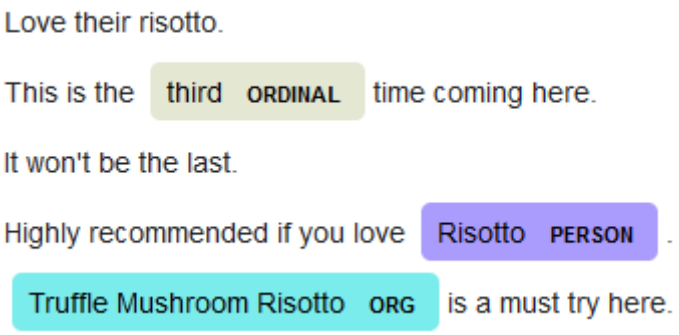


Figure 5.1.3b

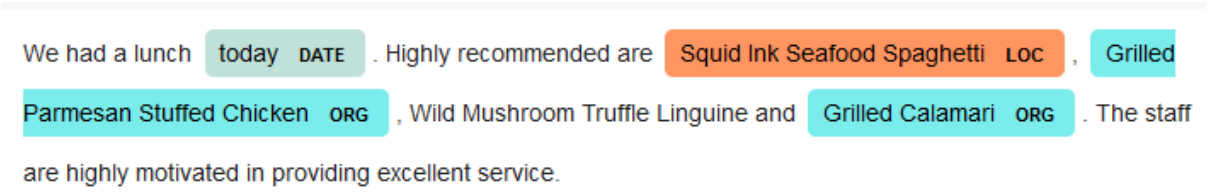


Figure 5.1.3c

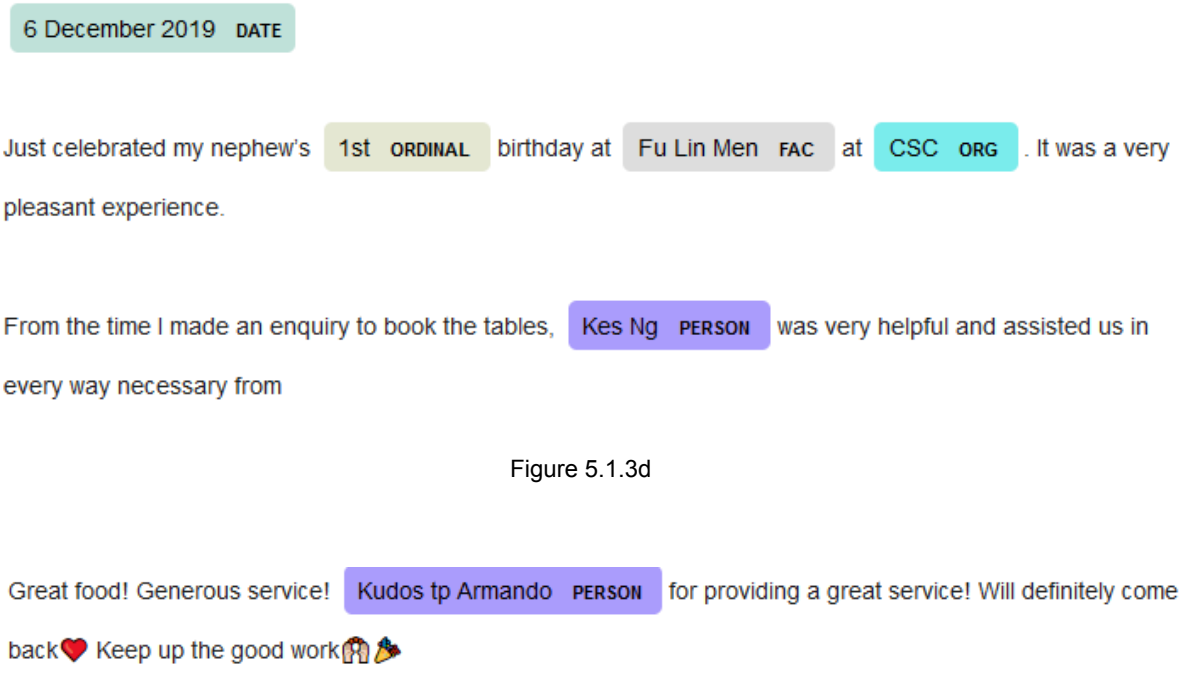


Figure 5.1.3d

Figure 5.1.3e

From the above five examples, we can see that most of the time the NER tagged it correctly. However, sometimes NER is unable to detect minor spelling errors as seen in Figure 5.1.3e, where 'tp' was meant to be 'to', and hence it takes 'Kudos tp Armando' as a PERSON entity instead of just 'Armando'. Also, NER takes all the phrases of food as either an ORG (organization) or LOC (location) as seen in Figure 5.1.3b and Figure 5.1.3c.

Summary of NER steps

Figure 5.1.4 summarizes the steps we took to implement NER into our project

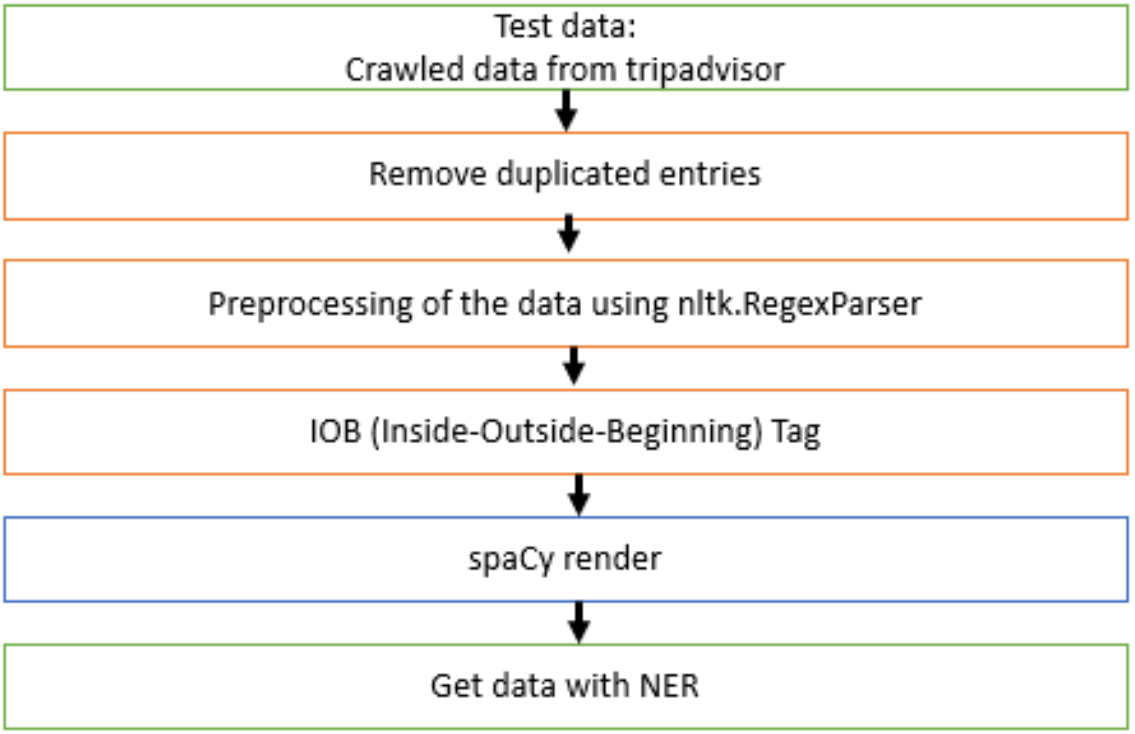


Figure 5.1.4: Steps taken to implement NER

Conclusion

Having gone through the three stages in this assignment, crawling, indexing & querying and also classification, we aim to accurately classify the comments given by the users who reviewed a particular restaurant food. From our application's perspective, we aim to impact other users to make informed decisions in selecting their choices of food for their next meal through our simple user interface backed by the indexing and querying system as well as the deep analysis in the classification of tasks.

References

http://docplayer.net/889523-Solr-cloud-vs-replication.html

https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/

https://www.javaer101.com/en/article/18187799.html

https://www.kdnuggets.com/2018/04/why-deep-learning-perfect-nlp-natural-language-processing.html

https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

https://www.researchgate.net/figure/LSTM-and-BiLSTM-Architectures_fig2_324769532

https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/

https://arxiv.org/pdf/1706.03762.pdf

https://peltarion.com/knowledge-center/documentation/tutorials/movie-review-feelings

https://www.kaggle.com/snap/amazon-fine-food-reviews

https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

https://deepai.org/machine-learning-glossary-and-terms/f-score

https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

https://stackoverflow.com/questions/55740220/macro-vs-micro-vs-weighted-vs-samples-f1-score

https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/

https://github.com/jayaaiyappan/Sentence-Segmentation/blob/master/Sentence%20Segmentation.ipynb

https://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/

https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63

https://machinelearningmastery.com/improve-deep-learning-performance/

https://medium.com/analytics-vidhya/aspect-based-sentiment-analysis-a-practical-approach-8f51029bbc4a

https://github.com/susanli2016/NLP-with-Python/blob/master/NER_NLTK_Spacy.ipynb

https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da

https://www.geeksforgeeks.org/nlp-iob-tags/