Q
n
A
iTech
Part of the image taken from free download https://thumbs.dreamstime.com/z/q-26343823.jpg

EBA5004 – PLP Practice Module – Group 15

A0100172A Lim Kah Ghi

A0024023A Ryan Chng Yan Hao

Table of Contents

1.0 Executive Summary 3

2.0 Introduction 4

  2.1 What is StackOverflow 4

  2.2 Problem Statement 4

  2.3 Proposed System 4

3.0 Data 5

  3.1 Data Acquisition 5

  3.2 Data Selection and Exploration 6

    3.2.1 Dataset for Classifier – Python 6

    3.2.2 Dataset for Classifier – Tags 6

    3.2.3 Get Questions-Answers 6

  3.3 Data Pre-processing 7

    3.3.1 Classifier – Python 7

    3.3.2 Classifier – Tags 8

    3.3.3 Text Summary 9

4.0 Approaches 10

  4.1 Classifier – Python 10

  4.2 Classifier – Tags 11

  4.3 Extraction of ROOTS and dObj 12

  4.4 Text Summary via Manually Built Transformer 13

5.0 System Architecture / Conversation Flow 15

6.0 Conclusion 17

  6.1 Future work 17

7.0 Reference 18

1.0 Executive Summary

Whenever we have a question to ask, we often hear people say, “Google is your best friend”. Indeed, the internet (or namely Google Search) is a great place to find solutions online. Information online exists in various forms, from forums, to articles, to books. However, to formulate a good query that will potentially return an answer that will solve your problem, one would have to know how and what to ask. When links to articles or even to books, one would also need to know where to extract the answer from.

This process is the same regardless of domain. For this project, we focused on python coding related questions, based on StackOverflow forum. When we Google a coding related query, most of the time top results would be linked to StackOverflow threads, where others have asked a similar question with answers from the community.

However, oftentimes many threads must be explored before the most similar question and hence the most suitable answer can be retrieved. It is our aim, through iTechQnA, to reduce the time spent sieving through these queries.

iTechQnA functions as a simple QA conversation flow focused onto Python-related questions, by classifying if a question is python or not. It then classifies the question into commonly used tags and returns a short summary of the best answer from historical question-answer pairs. iTechQnA is compact and values speed and simplicity to achieve the aims.

2.0 Introduction

2.1 What is StackOverflow

StackOverflow is one of the most popular websites in the world, serving 100 million people every month. StackOverflow is where the community comes together "to find and contribute answers to technical challenges", thereby contributing to the public platform by building a "definitive collection of coding questions and answers" (Stack Exchange, 2021).

When we encounter issues during coding, we would often turn towards searching for solutions online through Google Search. Often, the top results would be returned from StackOverflow, where someone else has encountered a similar issue and even better, somebody has provided a solution to it. StackOverflow thus is an extremely useful platform that hosts various answers and questions on a myriad of coding related topics.

2.2 Problem Statement

On StackOverflow, many similar questions are asked, with many possible answers or same answers provided by the community across multiple posts. These questions are typically reflected when we do a quick Google search. However, there are no ranking to the questions, nor any answers shown from the Google search. As such, users may have to click on several links before finding the solution for the problem.

2.3 Proposed System

Our proposed solution, iTechQnA, aims to provide a summary of answers best suited to answer their query, limited to the context of python topics. iTechQnA is compact and values speed and simplicity to achieve the aims.

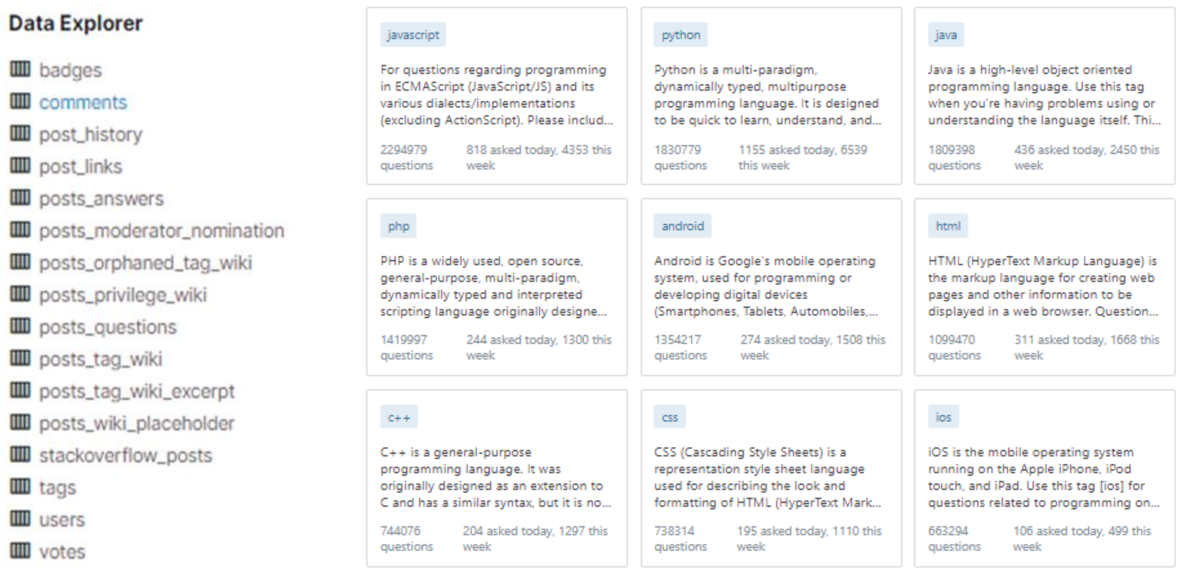
iTechQnA would first get the user input, classify the existing answers from the corpus into high or low quality, returns partial or full best matched solution while developers wait for new replies, through a conversational user interface. There may be specific problems that require new solutions at times, and it is our hope that iTechQnA may still return solutions that may aid the developer, with a chance to resolve their problem faster.

3.0 Data

3.1 Data Acquisition

To form our dataset, we utilised StackAPI (Wegner, 2016) to download data from January 2018 to June 2021, with limited calls of 10,000 calls per registered user. Around 500,000 to 1 million rows of data were retrieved over a period of 1 to 2 weeks and PySpark was utilised for data exploration and wrangling of the large dataset.

There are various data available to call using StackAPI, such as badges, comments, post_history, post_links, post_answers and more (Stack Overflow, 2019), as shown in Figure 1. Data that we selected for the scope of our project includes questions, answers, tags and score. The score is computed by subtracting the downvotes from upvotes by StackOverflow.



Data Explorer

- badges
- comments
- post_history
- post_links
- posts_answers
- posts_moderator_nomination
- posts_orphaned_tag_wiki
- posts_privilege_wiki
- posts_questions
- posts_tag_wiki
- posts_tag_wiki_excerpt
- posts_wiki_placeholder
- stackoverflow_posts
- tags
- users
- votes

javascript
For questions regarding programming in ECMAScript (JavaScript/JS) and its various dialects/implementations (excluding ActionScript). Please includ...
2294979 questions — 818 asked today, 4353 this week

python
Python is a multi-paradigm, dynamically typed, multipurpose programming language. It is designed to be quick to learn, understand, and...
1830779 questions — 1155 asked today, 6539 this week

java
Java is a high-level object oriented programming language. Use this tag when you're having problems using or understanding the language itself. Thi...
1809398 questions — 436 asked today, 2450 this week

php
PHP is a widely used, open source, general-purpose, multi-paradigm, dynamically typed and interpreted scripting language originally designe...
1419997 questions — 244 asked today, 1300 this week

android
Android is Google's mobile operating system, used for programming or developing digital devices (Smartphones, Tablets, Automobiles,...
1354217 questions — 274 asked today, 1508 this week

html
HTML (HyperText Markup Language) is the markup language for creating web pages and other information to be displayed in a web browser. Question...
1099470 questions — 311 asked today, 1668 this week

c++
C++ is a general-purpose programming language. It was originally designed as an extension to C and has a similar syntax, but it is no...
744076 questions — 204 asked today, 1297 this week

css
CSS (Cascading Style Sheets) is a representation style sheet language used for describing the look and formatting of HTML (HyperText Mark...
738314 questions — 195 asked today, 1110 this week

ios
iOS is the mobile operating system running on the Apple iPhone, iPod touch, and iPad. Use this tag [ios] for questions related to programming on...
663294 questions — 106 asked today, 499 this week

Figure 1: data available from StackAPI and popular tags

To ensure that our dataset is focused on python related questions, we relied on the tags during the StackAPI call. All python-tagged questions and answers were retrieved and to form the non-python related data, we retrieved and queried some of the most popular non-python tags such as javascript, android etc. as shown in Figure 1. A total of 3.93GB of data was downloaded. The data set comprise of around 880,000 python related questions, 1 million answers due to multiple answers available for the same question, and around 295,000 non-python related records.

3.2 Data Selection and Exploration

To live up to iTechQnA's purpose of being compact and efficient, 3.93GB data is clearly too much for our system to handle. As such, sampling from the downloaded data had to be done, based on some light inference and experimentation.

3.2.1 Dataset for Classifier – Python

Our first classifier will predict the query into python or non-python, in order to filter questions that iTechQnA can handle. As such, a dataset containing questions labelled 'python' and 'others' had to be obtained from our initial download. To achieve a minimum accuracy of 90% for this classifier, a balanced dataset of python and non-python examples had to be sampled. From the python related questions, we sampled 29,613 (3% of 880,000) python related records, and 29,918 (10% of 295,000) non-python related questions, giving a total of 59,531 records.

3.2.2 Dataset for Classifier – Tags

Data with tag counts greater or equal to 3 and with tag occurrence of at least 100 were chosen. This was because one of the tags has been taken up by 'python' and training on individual sub tags did not yield good models by experiment. Training on low tag occurrence does not help much in predicting the tags as well.

From the initial models, a manually calculated similarity accuracy of around 85% can be achieved when training with tag occurrence of at least 200. However, there are too little (17) unique tags to use later. When training with tag occurrence of at least 100, there was a slight decrease in accuracy to 78% but there are more usable unique tags now (34). Detailed results can be found in Section 4.2.

Further reduction of tag occurrence to at least 50 further reduced the accuracy to 73% but increased the number of unique tags (67). However, the smaller occurring tag sequences were not correctly predicted. As such, tag occurrence of at least 100 was utilised.

3.2.3 Dataset for Questions-Answers

For the actual question-answer system of iTechQnA, the question-answer dataset was obtained by assessing the score of the question. A score was defined based on the number of upvotes minus the number of downvotes, where the votes are decided by StackOverflow. Questions with score greater or equal to 2 were sampled for a particular question. Reason for chose of score:

i. Upvoting – downvoting >= 2 indicated more users agree / interested in the question

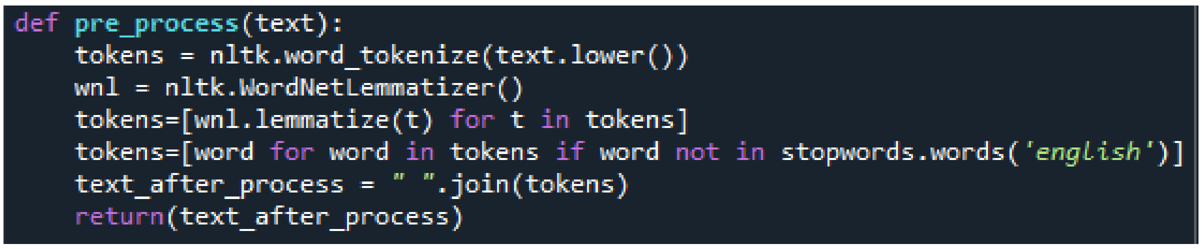
ii. The 25th quartile – 75th quartile score was 0-1 and the scores were integers, choosing >= 2 clearly using 75th quartile onwards.
iii. This step has downsized the records from ~880K → ~118K

By selecting questions with at least 3 tags and above, with tag occurrence of at least 100, the data set was further downsized the records to 9,228 (an ideal size for speed). To ensure that each question has an answer, questions without accepted answers were dropped and the final size was 7,613 records.

3.3 Data Pre-processing

3.3.1 Classifier – Python

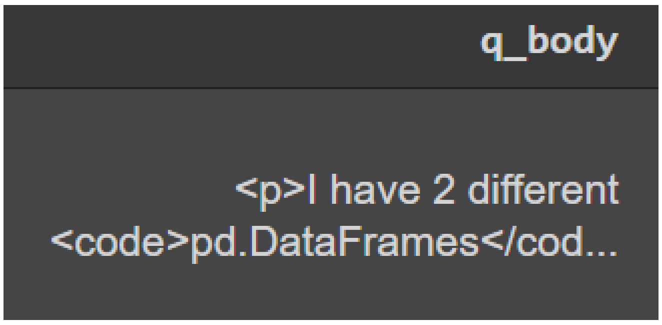
Some pre-processing steps done to prepare the dataset for python/non-python prediction includes tokenization, lemmatization, and stop words removal. This was done mainly using the nltk package and the sample code can be seen in Figure 2.



```python
def pre_process(text):
    tokens = nltk.word_tokenize(text.lower())
    wnl = nltk.WordNetLemmatizer()
    tokens=[wnl.lemmatize(t) for t in tokens]
    tokens=[word for word in tokens if word not in stopwords.words('english')]
    text_after_process = " ".join(tokens)
    return(text_after_process)
```

Figure 2: Pre-processing function used for python classifier

There are special characters, as shown in Figure 3, such as html links and punctuations like '<p>', '</p> to denote the start and end of the question and '<code>', '</code>' to represent the start and end of a code.



q_body

<p>I have 2 different <code>pd.DataFrames</cod...

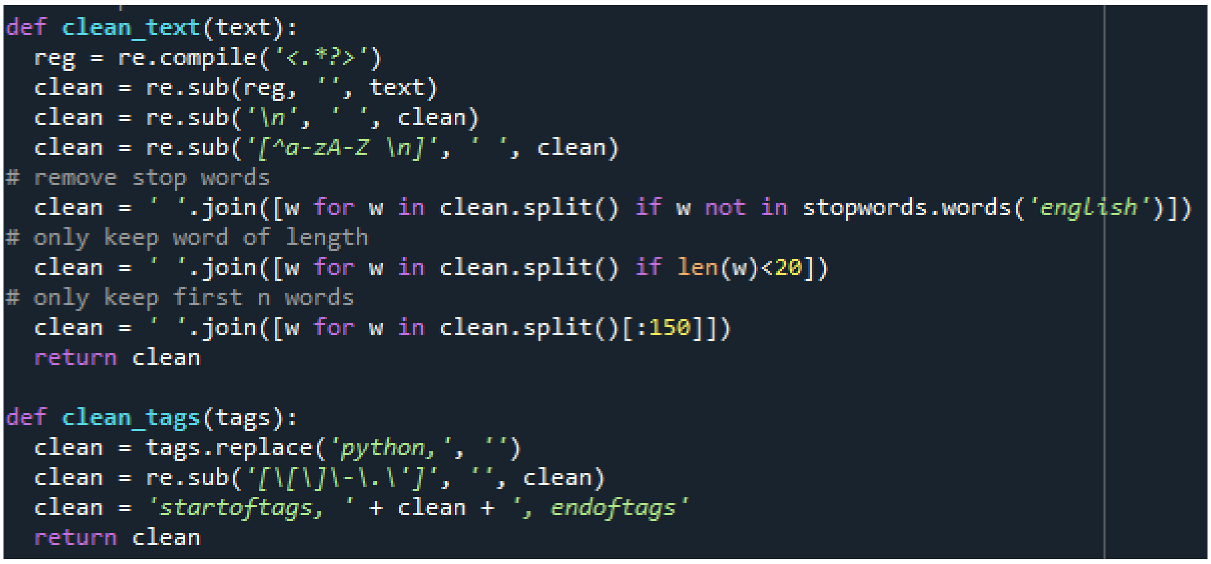
Figure 3: Special characters in corpus

However, these special punctuations and html tags were not removed in the final system as our experiments have shown that these have no effect, as these tokens occur in all records and would have been take care of by the tf-idf function used in the downstream process.

3.3.2 Classifier – Tags

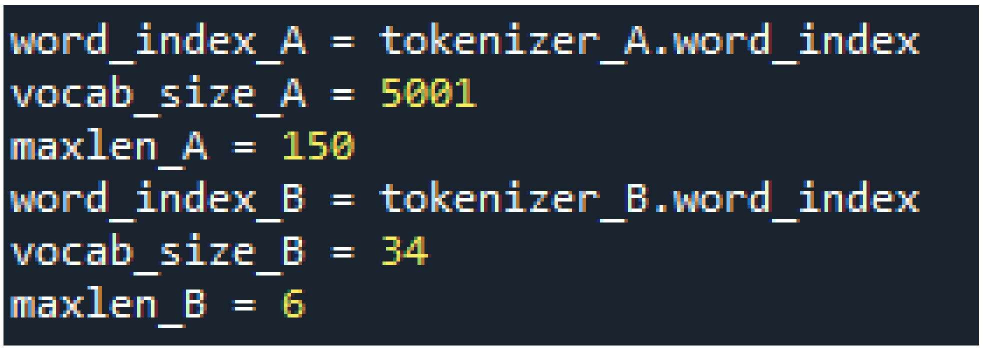
To build the classifier for tags, we utilise a bi-LSTM encoder-decoder model. As such, the dataset would have to be input as two sequences, with sequence A being the questions and sequence B being the tags.

Sequence A is set as the question title concatenated with the question body as its training text. To ensure that no learning was done based on the special punctuations and html tags, as well as numbers and symbols, such information were removed via the clean_text function shown in Figure 4. Words of length less than 20 was also selected to ensure longer chunks of code were removed. As certain sequences may be very long due to the question body that contains details on erroneous codes, only the first 150 words were kept as this would most likely capture the essence of the question. The vocabulary list was capped at 5001 as one of the attempts from preliminary exploration returned more than 100,000 tokens, with some tokens resembling long file paths or websites.



```
def clean_text(text):
  reg = re.compile('<.*?>')
  clean = re.sub(reg, '', text)
  clean = re.sub('\n', ' ', clean)
  clean = re.sub('[^a-zA-Z \n]', ' ', clean)
# remove stop words
  clean = ' '.join([w for w in clean.split() if w not in stopwords.words('english')])
# only keep word of length
  clean = ' '.join([w for w in clean.split() if len(w)<20])
# only keep first n words
  clean = ' '.join([w for w in clean.split()[:150]])
  return clean

def clean_tags(tags):
  clean = tags.replace('python,', '')
  clean = re.sub('[\[\]\-\.\']', '', clean)
  clean = 'startoftags, ' + clean + ', endoftags'
  return clean
```



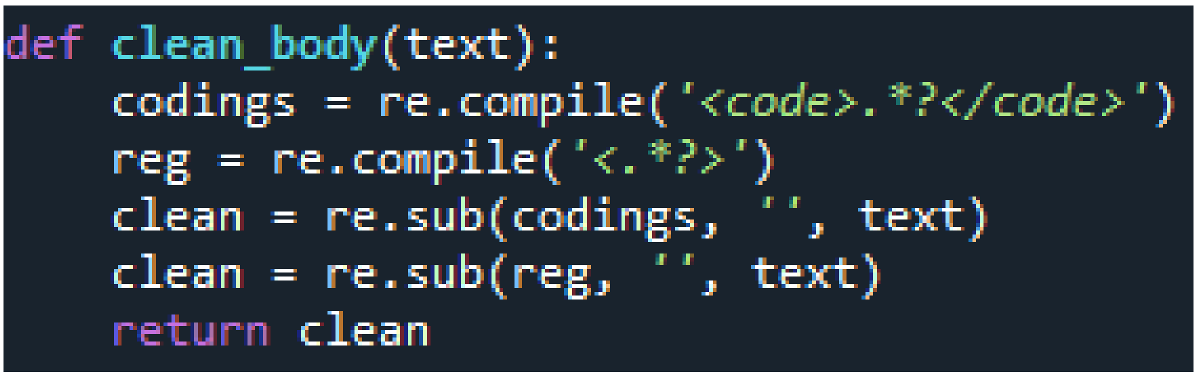
```
word_index_A = tokenizer_A.word_index
vocab_size_A = 5001
maxlen_A = 150
word_index_B = tokenizer_B.word_index
vocab_size_B = 34
maxlen_B = 6
```

Figure 4: Pre-processing steps functions and variables used for tag classification

The input to the decoder as Sequence B are the tag labels, which are a cleaner set. To indicate the start and end of the sequence, ‘startoftags’ and ‘endoftags’ were added respectively.

3.3.3 Text Summary

For text summary, the question-answers pairs had their code portion and html tags removed as shown in clean_body function in Figure 5.



```python
def clean_body(text):
    codings = re.compile('<code>.*?</code>')
    reg = re.compile('<.*?>')
    clean = re.sub(codings, '', text)
    clean = re.sub(reg, '', text)
    return clean
```

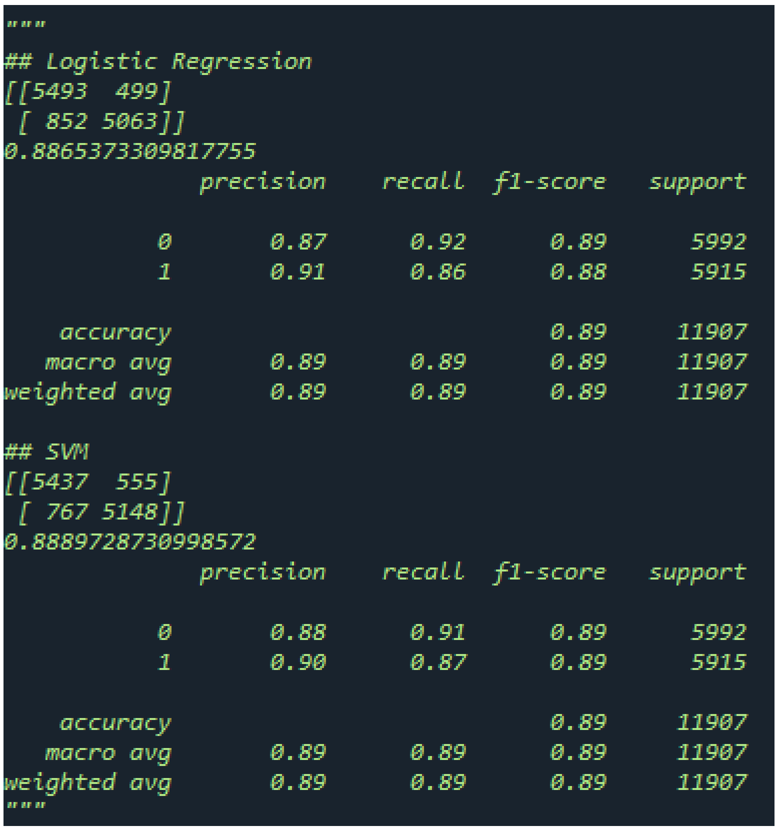
Figure 5: Pre-processing function used for text summary

This was primarily aimed to obtain code-free summary in the “same language style” used when doing the summary.

4.0 Approaches

4.1 Classifier – Python

A few models were trained before the final model of tf-idf + support vector machine (SVM) model was chosen. A train-test split of 80%: 20% was utilised during training and testing. A logistic regression model was also trained. From Figure 6, both the logistic regression model and SVM model performed similarly with an accuracy of 89%. We chose SVM model as there were fewer misclassifications when looking at the confusion matrix. Logistic regression mis-classified a total of 1,351 (499+852) records while SVM only mis-classified a total of 1,322 (767+555) records.



```
"""
## Logistic Regression
[[5493  499]
 [ 852 5063]]
0.8865373309817755
              precision    recall  f1-score   support

           0       0.87      0.92      0.89      5992
           1       0.91      0.86      0.88      5915

    accuracy                           0.89     11907
   macro avg       0.89      0.89      0.89     11907
weighted avg       0.89      0.89      0.89     11907

## SVM
[[5437  555]
 [ 767 5148]]
0.8889728730998572
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      5992
           1       0.90      0.87      0.89      5915

    accuracy                           0.89     11907
   macro avg       0.89      0.89      0.89     11907
weighted avg       0.89      0.89      0.89     11907
"""
```

Figure 6: Results of logistic regression and SVM for python-classifier

The classifier will ultimately predict if a question is python (1) or not python (0) during run time. For example, for an unseen question “How to concat two columns in pd dataframe?”, the classifier will return 1 as the query is related to python. Similarly, for an unseen question “MainActivity which has a button when on pressed reads data from a QRscan. After that, a background service is started and data from QRscan is sent to it. It has an update method defined to create a fragment once it is invoked by”, the system will return 0.

4.2 Classifier – Tags

To build the classifier for tag prediction, a bidirectional long-short term memory (bi-LSTM) encoder and decoder system was trained for 15 epochs. Training beyond 20 epochs showed drastic overfitting during experimental stage. The performance of the model can be seen in Figure 7, with accuracy taken only as a reference as the tags predicted can return in different orders.

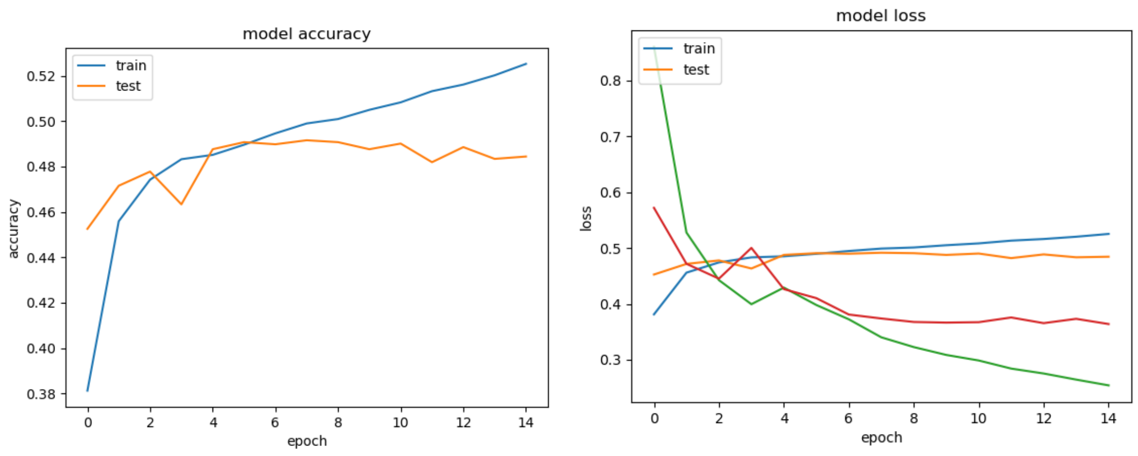
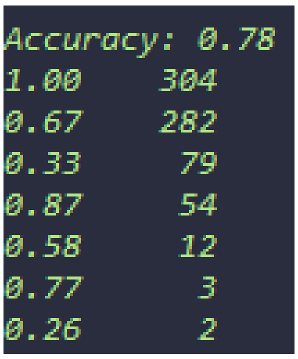


Figure 7: Accuracy (left) and loss (right: yellow/blue is accuracy, red/green is loss) for bi-LSTM encoder-decoder model

Accuracy of the model was manually obtained by calculating the cosine similarity of lemmatized tags for actual and predicted. 737 unseen records were held for evaluation, with ~89% (655/737) of the predicted tags having a similarity of 0.5 or higher, which translates to first sub tag predicted correctly. Figure 8 shows the model performance, with our manually calculated cosine similarity.



```
Accuracy: 0.78
1.00     304
0.67     282
0.33      79
0.87      54
0.58      12
0.77       3
0.26       2
```

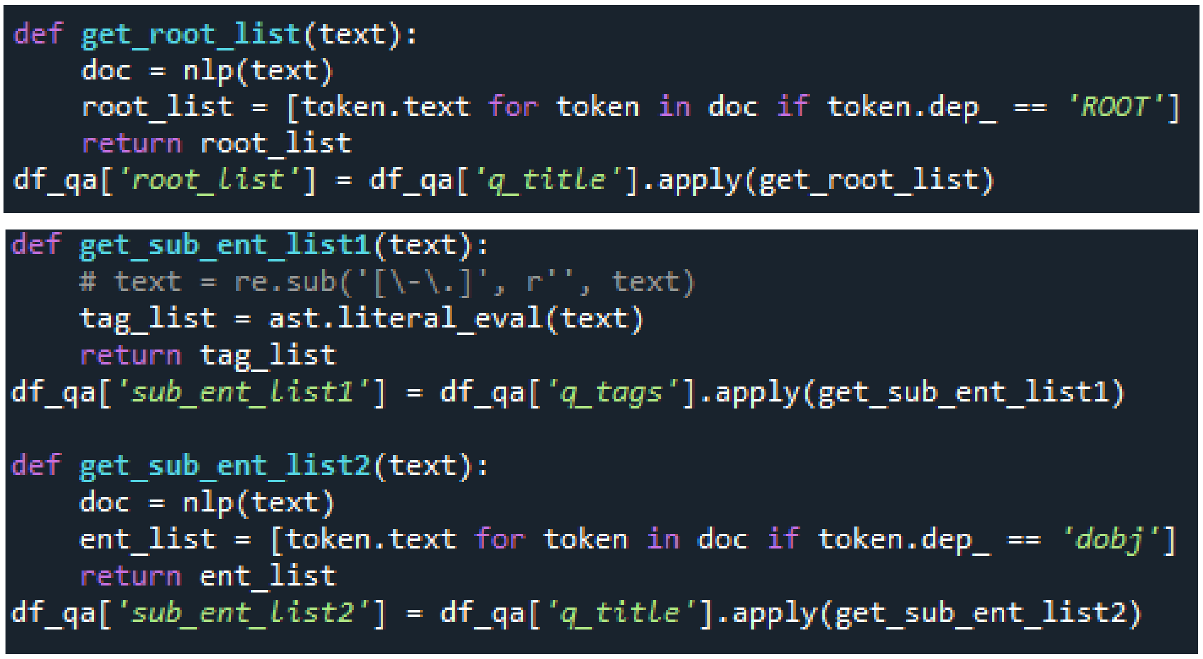
Figure 8: Performance of Bi-LSTM encoder-decoder model based on manually calculated cosine similarity

A cosine similarity score of 1.0 implies that all tags were correctly predicted. For example, actual tags of ‘python’, ‘pandas’, and ‘dataframe’ were predicted to be the same, in any order. A score of 0.33 implies that 1 out of 3 tags were correctly predicted, such as ‘python’, ‘tensorflow’, ‘keras’, where only ‘python’ was predicted correctly.

Overall, the Bi-LSTM encoder-decoder system performed quite well, with an accuracy of 78%.

4.3 Extract and Match ROOTS and dObj

To extract the root and direct object (dObj) from the query, dependency relations of ROOT and dObj were utilised from spaCy. Figure 9 shows the codes used to split the question titles from our dataset into roots and dObj.



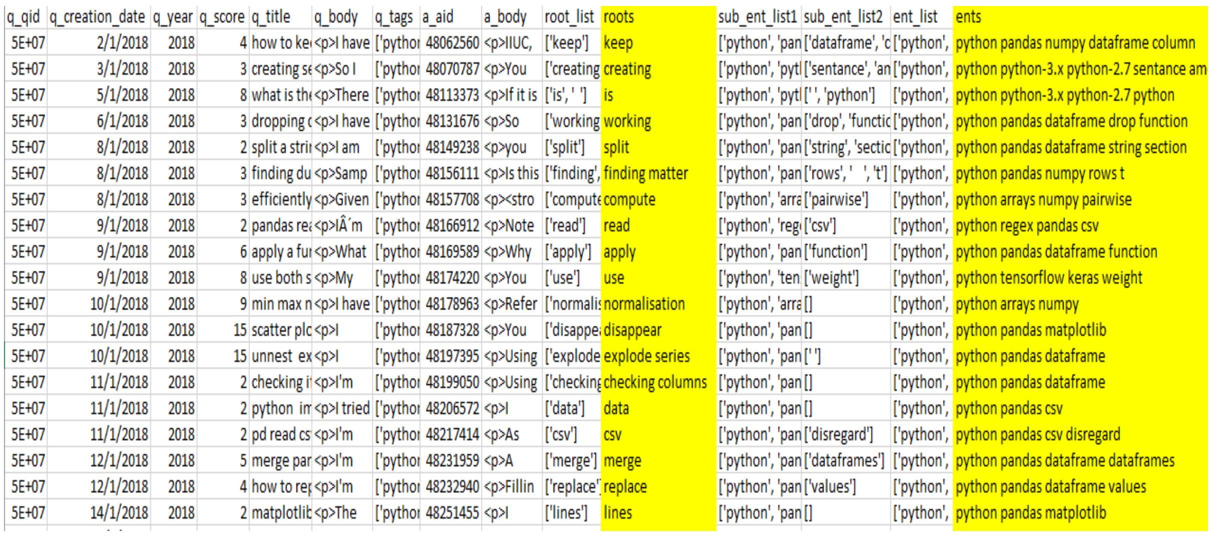
```
def get_root_list(text):
    doc = nlp(text)
    root_list = [token.text for token in doc if token.dep_ == 'ROOT']
    return root_list
df_qa['root_list'] = df_qa['q_title'].apply(get_root_list)
```

```
def get_sub_ent_list1(text):
    # text = re.sub('[\-\.]', r'', text)
    tag_list = ast.literal_eval(text)
    return tag_list
df_qa['sub_ent_list1'] = df_qa['q_tags'].apply(get_sub_ent_list1)

def get_sub_ent_list2(text):
    doc = nlp(text)
    ent_list = [token.text for token in doc if token.dep_ == 'dobj']
    return ent_list
df_qa['sub_ent_list2'] = df_qa['q_title'].apply(get_sub_ent_list2)
```

Figure 9: codes used to retrieve roots and dObj from text

In addition to the dependency relation based on spaCy, we also developed a dictionary (Figure 10) to map the query to manually derived unique entities from the dataset, based on similarity scores. Although this may not be the best way to perform entity extraction, this fast and naïve matching via rules do provide the speed needed for iTechQnA.



q_qid	q_creation_date	q_year	q_score	q_title	q_body	q_tags	a_aid	a_body	root_list	roots	sub_ent_list1	sub_ent_list2	ent_list	ents
5E+07	2/1/2018	2018	4	how to ke	<p>I have	['pythor	48062560	<p>IIUC,	['keep']	keep	['python', 'pan	['dataframe', 'c	['python',	python pandas numpy dataframe column
5E+07	3/1/2018	2018	3	creating se	<p>So I	['pythor	48070787	<p>You	['creating	creating	['python', 'pytl	['sentance', 'an	['python',	python python-3.x python-2.7 sentance am
5E+07	5/1/2018	2018	8	what is th	<p>There	['pythor	48113373	<p>If it is	['is', ' ']	is	['python', 'pytl	[' ', 'python']	['python',	python python-3.x python-2.7 python
5E+07	6/1/2018	2018	3	dropping c	<p>I have	['pythor	48131676	<p>So	['working	working	['python', 'pan	['drop', 'functic	['python',	python pandas dataframe drop function
5E+07	8/1/2018	2018	2	split a stri	<p>I am	['pythor	48149238	<p>you	['split']	split	['python', 'pan	['string', 'sectic	['python',	python pandas dataframe string section
5E+07	8/1/2018	2018	3	finding du	<p>Samp	['pythor	48156111	<p>Is this	['finding',	finding matter	['python', 'pan	['rows', ' ', 't']	['python',	python pandas numpy rows t
5E+07	8/1/2018	2018	3	efficiently	<p>Given	['pythor	48157708	<p><stro	['compute	compute	['python', 'arra	['pairwise']	['python',	python arrays numpy pairwise
5E+07	9/1/2018	2018	2	pandas rea	<p>IÂ´m	['pythor	48166912	<p>Note	['read']	read	['python', 'reg	['csv']	['python',	python regex pandas csv
5E+07	9/1/2018	2018	6	apply a fu	<p>What	['pythor	48169589	<p>Why	['apply']	apply	['python', 'pan	['function']	['python',	python pandas dataframe function
5E+07	9/1/2018	2018	8	use both s	<p>My	['pythor	48174220	<p>You	['use']	use	['python', 'ten	['weight']	['python',	python tensorflow keras weight
5E+07	10/1/2018	2018	9	min max n	<p>I have	['pythor	48178963	<p>Refer	['normalis	normalisation	['python', 'arra	[]	['python',	python arrays numpy
5E+07	10/1/2018	2018	15	scatter plo	<p>I	['pythor	48187328	<p>You	['disappe	disappear	['python', 'pan	[]	['python',	python pandas matplotlib
5E+07	10/1/2018	2018	15	unnest ex	<p>I	['pythor	48197395	<p>Using	['explode	explode series	['python', 'pan	[' ']	['python',	python pandas dataframe
5E+07	11/1/2018	2018	2	checking i	<p>I'm	['pythor	48199050	<p>Using	['checking	checking columns	['python', 'pan	[]	['python',	python pandas dataframe
5E+07	11/1/2018	2018	2	python im	<p>I tried	['pythor	48206572	<p>I	['data']	data	['python', 'pan	[]	['python',	python pandas csv
5E+07	11/1/2018	2018	2	pd read cs	<p>I'm	['pythor	48217414	<p>As	['csv']	csv	['python', 'pan	['disregard']	['python',	python pandas csv disregard
5E+07	12/1/2018	2018	5	merge par	<p>I'm	['pythor	48231959	<p>A	['merge']	merge	['python', 'pan	['dataframes']	['python',	python pandas dataframe dataframes
5E+07	12/1/2018	2018	4	how to rep	<p>I'm	['pythor	48232940	<p>Fillin	['replace'	replace	['python', 'pan	['values']	['python',	python pandas dataframe values
5E+07	14/1/2018	2018	2	matplotlib	<p>The	['pythor	48251455	<p>I	['lines']	lines	['python', 'pan	[]	['python',	python pandas matplotlib

Figure 10: Dictionary mapping entities (dObj) to query

For a given question "how to keep dataframe column and index names after calculation with np where", the above functions will return the root as "keep" and dObj as "python pandas numpy dataframe column".

4.4 Text Summary

The manually built transformer covered in the Text Processing using Machine Learning module was borrowed to perform this task. The encoder and decoder length were set to 250 and 90 words respectively.

The transformer was trained by using the question body against its question titles and applied onto the totally unseen answers. We did this as we had to have a domain-specific text to summary corpus, and this was available within our dataset based on the question body to question titles. Despite being a bold approach, the results are acceptable.

The below paragraph is the actual answer found on StackOverflow, with a summarized answer of

"how to get the value of a dataframe based on another column":

> Method 1: numpy.split &amp; DataFrame.loc:
>
> We can split your columns into evenly size chunks and then use .loc to create the new columns:
>
> ```
> for idx, chunk in enumerate(np.split(df.columns, len(df.columns)/4)):
>     df[f'A{idx+1}_avg'] = df.loc[:, chunk].mean(axis=1)
> ```
>
> Output "… (many lines of array)"
>
> Method 2: .range &amp; iloc:
>
> We can create a range for each 4 columns, then use iloc to acces each slice of your dataframe and calculate the mean and at the same time create your new column:
>
> ```
> slices = range(0, len(df.columns)+1, 4)
> for idx, rng in enumerate(slices):
>     if idx &gt; 0:
>         df[f'A{idx}_avg'] = df.iloc[:, slices[idx-1]:slices[idx]].mean(axis=1)
> ```
>
> Output "… (many lines of array)"
>
> [20 rows x 25 columns]

Another full answer, as shown below, was summarized as

"pandas dataframe groupby and add the value of the value".

> I believe you need groupby:
>
> ```
> df['D'] = df["C"].shift(1).groupby(df['A'], group_keys=False).rolling(2).mean()
> print (df.head(20))
> ```
>
> "… (many lines of array)"
>
> Or:
>
> ```
> df['D'] = df["C"].groupby(df['A']).shift(1).rolling(2).mean()
> print (df.head(20))
> ```
>
> "… (many lines of array)"

5.0 System Architecture / Conversation Flow

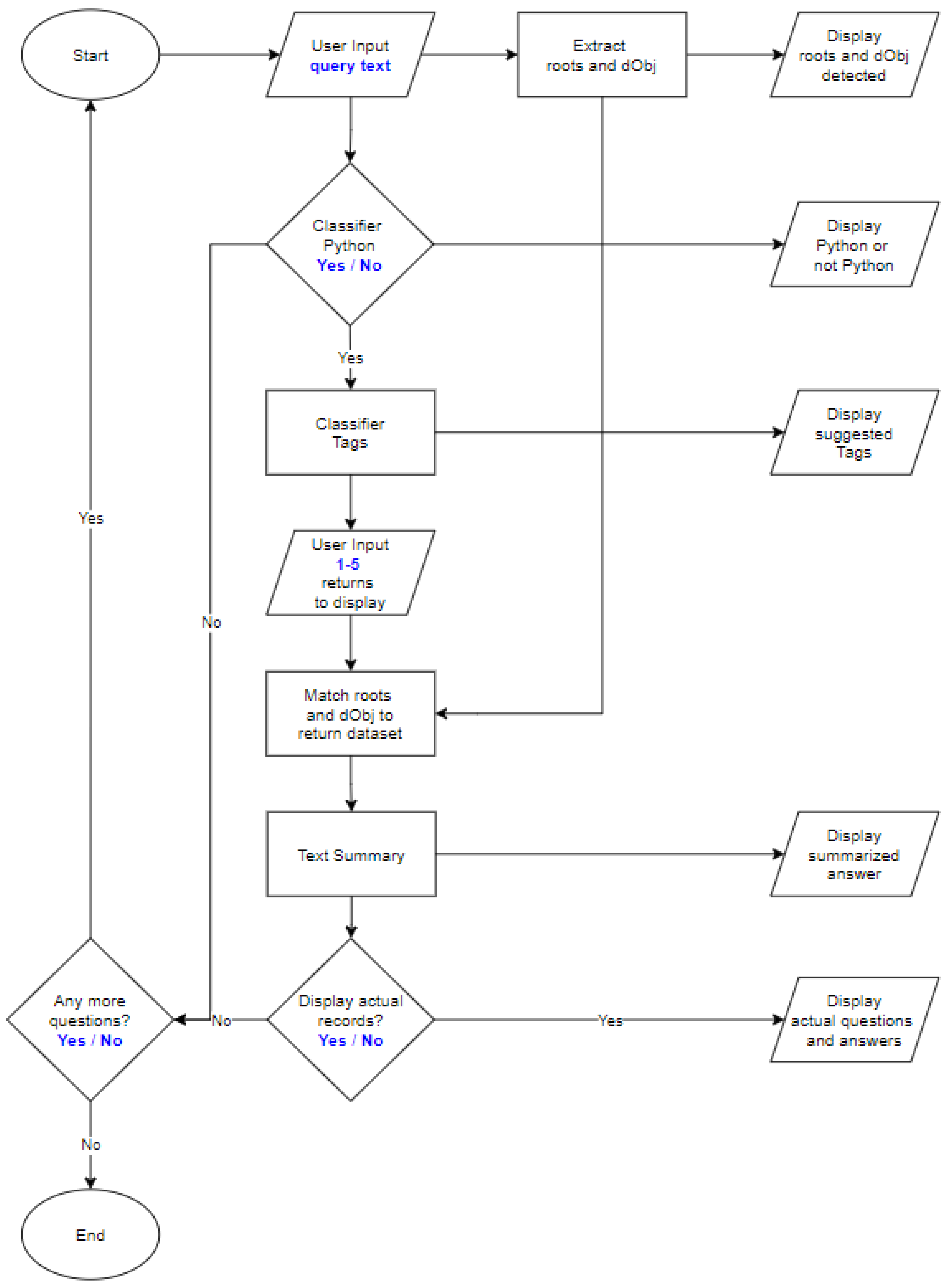


Figure 11: System Architecture/Conversation Flow

Figure 11 depicts the conversation flow as well as the system architecture. First, iTechQnA is initiated through command prompt. The user then provides the query input, and iTechQnA will then classify if the query is related to the topic python or not. If the query is not related to python, the system will prompt the user for another question.

Upon successful identification of python questions, iTechQnA then extract the roots and direct object (dObj) in the user query and displays them. At the same time, the possible tags related to the user query is also detected and displayed. The user will then have to input the number of matches to display, from 1 to 5.

Upon confirmation of the number of answers to display, iTechQnA then return the records with the highest similarity scores by matching the roots and dObj of query with the data. The answer is then summarized and displayed.

The user also has the option to choose the full answer to check more details. Finally, the user is then prompted if they have any other questions. If there are no further questions, iTechQnA will then exit.

6.0 Conclusion

iTechQnA is a system that helps developers be able to retrieve answers specific to python domain, with focus on speed and simplicity. Below shows our rating of the various approaches and models deployed in iTechQnA, based on three levels of

[“Gold standard”, “Good enough”, “Better than nothing”].

Classifier – Python:	“Good enough” -- “Gold standard”
Classifier – Tags:	“Good enough”
Extraction of ROOTS and dObj:	“Better than nothing” -- “Good enough”
Text Summary:	“Good enough”
Conversation flow:	“Good enough”

There are many available methods and models to implement a similar system to iTechQnA. iTechQnA distinguished itself by keeping its size small (150mb without initial data source which is not required during the run) and providing close to instant returns on common PC/laptop systems. The balance of a system built in terms of its speed and performance is a continuous art to be learnt by itself which require more explorations for further exploitations.

6.1 Future work

iTechQnA have much potential and can be further enhanced in future work. Instead of specifying a static timeframe for data to be collected from, a dynamic desired time frame can be downloaded through incremental API calls. This would allow the user to be able to regularly update their corpus.

Besides this, due to the choice of models used in iTechQnA, the refreshing of all the data models would take less than a day on middle-graded laptops. This would also definitely aid the user in regularly updating their corpus.

The current conversation logic flow is kept simple for iTechQnA and this can definitely be enhanced or migrated for any web-based usage/API real usage.

Ideally, as iTechQnA largely relies on data from StackOverflow, it would be a great milestone for iTechQnA to be integrated with StackOverflow as a supplementary “known-it-all, history-retrieval, recommender bot”.

7.0 Reference

Stack Exchange Inc., (2021) StackOverflow. Retrieved from https://stackoverflow.com/

Stack Overflow (2019, Mar 21). Stack Overflow Data. Retrieved from https://www.kaggle.com/stackoverflow/stackoverflow?select=comments

Wegner, A. et. Al. (2016) StackAPI. Retrieved from https://stackapi.readthedocs.io/en/latest/