Answering Frequently Asked Questions In Slack Using A Slack Bot

Safwan Ull Karim
University Of Adelaide
Adelaide, SA, Australia
a1679297@adelaide.edu.au

Abstract

Slack is referred as the 21st century team communication app where teams can share messages, codes, links, pictures and much more. Team members using this method of communication often faces the struggle of answering repeated questions. This might happen when a new member joins the team or has been away from the team chat for a while. Sites like Google, Stack overflow or other websites normally will not be of any help as the information shared through Slack are mostly internal and under NDA (non-disclosure agreements). The ultimate goal of this project is to address this problem and try to rectify it using a slackbot to answer to repeated questions. Our results show that we were able to create a bot that successfully detects repeated questions however, it answers the questions with a link, pointing to the original question. This project builds strong a foundation for bots that focuses on answering repeated questions and gives insight on how to improve this topic.

1. Introduction

At an age where technology is at the forefront of almost everything one can think of, it is hard to imagine work without it. In most recent times, communication applications (apps) [1] like Slack, HipChat and Campfire have proven its usefulness to many organisations for its ease of interaction amongst team members, customers and other individuals related to the organisations. Teams can not only send conventional messages but also share links, pictures, documents, code snippets, you name it. The best part of using these apps are, these can serve as repositories for all kinds of information shared by a team.

Throughout this paper, we will use **Slack** [2] as the main communication app and use its API [3] to allow us to access information needed to tackle the obstacles faced by this project.

1.1 Motivation

A channel or a group in Slack is simply a chat which is generally used by more than one users. The major differences between a channel and a group are:

1. Anyone in their respective Slack team can join a **channel**, while to join a **group**, one needs to be invited in.
2. Messages in a **channel** can be viewed by anyone in their respective Slack team, while messages in a **group** can only be viewed by members of that group.

For simplicity, this paper will refer to both channels and groups as channels. Where difference arises between a channel and a group, it will be explicitly specified.

Members of a channel can come from various different fields who are joined by one major task through a channel. As a result, any idea, code or document shared by one member in a channel may follow up with questions from other members. More so, if a new member joins this channel or re-joins the channel after a period of leave, they may have questions which needs clarifications. More often than not, these questions have been asked and answered previously.

The aims of this paper can be summarised by the key research questions below:

RQ1 Would LSP [5] and rule based approach [6, 7] be an effective way to detect questions in chats?

RQ2 What are some possible algorithms that would best detect repeated questions?

RQ3 What are some possible algorithms that would best detect answers to asked questions?

Along with the obstacles mentioned above, this paper also aims to create a bot that can be used by any Slack team in order to resolve their struggle of answering repeated questions. Allowing the flow of chat to move forward instead of going back every now and then. The frontend of the bot, alias FAQBot, is written in Nodejs which will allow users of a Slack team to interact with the

bot. The backend of bot is done using Python. Team data extracted from Slack are be stored in a MYSQL database and messages are be stored in local files.

2.Related work

Finding ways to detect question-answer pairs is not a new topic in the world of natural language processing (NLP), neither is detecting repeated questions.

2.1 Previous studies on question-answer pairs

Many studies have been done that tries to extract questions-answer pairs from well formatted, question-answer forums like Stack Overflow, Yahoo answers and other community question answering websites (COA). One study of COA [4] uses the concept of graphs, where each node is a user and edges point from a user asking questions to another user answering back. These question can then be saved along with their answers and thus, allowing similar questions asked in the future to be answered by these pairs. A similar approach could be used in this study where questions and answers are nodes but separately marked. Question will be marked automatically by the system and then users, upon receiving a valid answer can mark the answer, creating an edge from the question node to the answer node. However, this is not a preferred way in a chat as constantly asking users to mark questions and answer will derail them from using the app all together.

Another critical study about extracting question-answer pairs from online forums done was by Gao Cong et al. [5], where the concept of labelled sequential pattern (LSP) was introduced to match questions with specific rules. To extract answers, the answers for that particular post were analysed using cosine similarity, KL-divergence language model and few others algorithms. The key point taken from this study was the use of LSP.

Others like Kyle Dent and Sharoda Paul [6] and Baichuan Li1 *et al.* [7] have tried to extract the question-answer pairs from informal sites like Twitter. Again they followed similar approaches like LSP and rule based approach.

Although the studies stated are similar to the one being done here, there is one critical difference. This paper focuses on extracting questions and answers from a chat, not forums or blogging sites, and therefore are bounded by many constraints, these are:

- Multiple questions being asked one after another.
- Multiple answers being given one after another.
- Answers maybe given in different order with respect to their questions.
- Questions or answers maybe asked in a form other text, i.e. using pictures, links, documents.

As a result of these constraints, extracting answers for its respective questions goes beyond the scope of this paper. Instead, we will be replying questions with links that will take the user to the original question. Then the user can skim through the conversation thread, hopefully finding the answer they were looking for.

2.2 Previous studies on detecting repeated questions

Again many studies were done on detecting repeated questions. But very little in regards to a chat box. A study done by Huizhong Duan *et al.* [8] suggested that a parse tree could be used to break up a user query and questions previously marked and return questions those were semantically the closest to the query. However, we decided to go with the idea of TFIDF [9], as done by Palakorn Achananuparp *et al.* [10]. They used TFIDF, Jaccard similarity and also parse trees to get similarity measures. But for our project, it was decided to start of slow and if possible build on our results. Some of our other measures of detecting similarity are described in section 3.2.

3. Methodology

Our methodology gets broken up into primarily 2 cases. Extracting questions from channels being the first, and detecting repeated questions asked in those channels, the second. For this project, Spacy [11] was used as the NLP library, for extraction and detection of questions.

3.1 Question extraction

To extract questions from Slack channels, we use LSP, combined with rule based approach to detect questions of various sorts. LSP works by taking n consecutive words in a sentence and creating a tuple of it. The message "How are you going today ?" with $n = 3$, will have 3 LSP of :

1. [How, are, you],
2. [are, you, going],
3. [you, going, today]

Detecting questions that ends with question mark or contains 5W1H words are not adequate [5]. Unlike online forums or blogging sites, Slack channels do not have a specified place to write questions or give answers. It all happens in one place. Therefore, whether a channel has well structured questions or not really depends on the flow of the channel. According to 5 different Slack channels, used for research purposes at the University of Adelaide, 82.6% of questions ended with question mark and 39% of questions had 5W1H words in it. It should be mentioned that, some channels had more well defined, formal questions, while a couple had more informal questions.

3,249 sentences were manually analysed to find patterns that would define a question. These patterns were then put together to define a set of rules which will be used to detect questions. The rules to detect questions are made up from many key words grouped into particular categories. A rule is made up of a combination of such categories, along with specific words or POS tags. These tags are formed using Spacy's part-of -speech tagging. Some rules are made to exclude particular keywords or tags (section 9, rule 20 & 24), in order to mark a tuple as a question. Full list of categories and rules can be found in **section**.

For every sentence the bot extracts, LSP are made and passed on to the rule analyser where the LSP were checked against all the rules. For each rule it fell under, the total count of that rule was incremented and the entire message got marked as a question. Figure 3.1 shows how the method is carried out. The value of *n* chosen for this project was ***n=4***. This was chosen because it would give the rule analyser more chance to check if the tuple does in fact fall under a rule that has length of 2 or 3, by checking the previous or next elements of the tuples. This also goes well with our list of rules as the longest rule is 4. On the other hand, having *n* too small would mean too many LSP tuples. The length of each rule would then have to be small as well, as there will be no point in having rules larger than the length of LSP. Having *n* too small gives more opportunities for false positives. Table 3.2 shows an example of how we can get a false positive and figure 3.3 shows how the weights are calculated.

Query: working on how to get rails done

Message	*n* = 2	*n* = 4
Working on how to get rails done.	Yes question, due to rule [how, to]	No question, due to rule [!on, how, to]

Table 3.2 shows what happens if value *n* is too small

$$weight = \left(\frac{\sum matches}{length_{query} + length_{question} - \sum matches} \times 100 \right) / length_{query}$$

Figure 3.3 shows how weights are calculated

Having *n* too big would not be suitable as some questions will not have *n* words in it. Also, it would mean rules will get far too complicated to make and unnecessary computation will occur even after a rule with length 4 was matched.

3.2 Detection of Repeated Questions

To detect repeated questions, we compare two categories of sentences:

1. **Query** asked by a user to which we are trying to find a similar question for.
2. **Questions** those were marked from the above, question extraction method step.

For each query, a *query-question* (Q-Q) pair were made. We used a total of 7 different algorithms and tried to come up with the best combination of these algorithms that would give us the best result. As suggested [10], we used algorithms that would tokenise each question sentence by word and find the TFIDF value of each word. In our case, each question in a channel is like a single document and the total set of questions in a channel are like the total set of documents. After we got TFIDF values of each word, we compared every word of a query with every word of a question in a Q-Q pair. The total weight of that pair would be the sum of the TFIDF values of each word that matched. Figure 3.4 and table 3.5 shows how TFIDF values are calculated and used.

$$tfidf_{i,j} = tf_{i,j} \times \log \frac{N}{df_i}$$

i = one particular word
j = sentence in which i is found
tf_{if} = total number of occurences of i in j
df_i = total number of sentences containing i
N = total number of sentences

Figure 3.4 shows the calculation of TFIDF

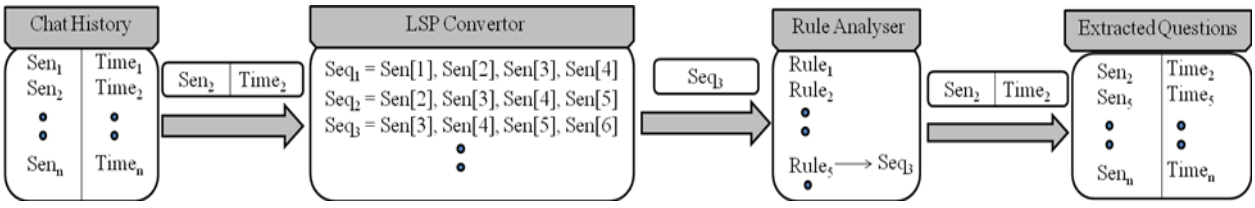


Figure 3.1 showing the procedure of extracting a questions

Word	TFIDF score
what	36.00
are	103.84
the	4.59
requirements	140.23
for	30.68
sprint	456.14
8	203.02

query: sprint 8 requirements
question: what are the requirements for sprint 8
weight of Q-Q pair: 456.14+203.02+140.23

Table 3.5 shows the TFIDF scores or each word in a query

Two other algorithms used were called **sentence_similarity** (sen_sim) and **lemma+sentence_similarity**. The former separates each word in a Q-Q pair and find how many matches were found. The latter algorithm does the same except it converts each word in a Q-Q pair to its lemma form and then tries to match. After an initial weight was found, we normalised the score by dividing it by the length of the query. Doing this we introduced a notion of relativity with the query. If we did not normalise the scores, chances of a Q-Q pair being a match would be more just because a question had a longer length.

Similar to the above two algorithms, we used **token_similarity** (tok_sim) and **lemma + token_similarity**, where instead of taking each word in a Q-Q pair, we removed all the stop words from every Q-Q pair and then matched the remaining words. Again we normalised the weight by dividing it by the total length of query, excluding the stop words.

The last two algorithm used took an average of the weights achieved by the above four algorithms. **Combine_similarity** was where we took the average of sentence_similarity and token_similarity. **Lemma + Combine_similarity** was where we took the average of lemma+sentence_similarity and lemma + token_similarity. Table 3.6 shows for two string how their respective scores are calculated.

query: should I be following manual 1 = 6
question: were we required to follow manual 1 = 7

Algorithm	Query	Question	Weight
Sen_sim	should, i, be, following, manual, 1	were, we, required, to follow, manual, 1	3.0
Lemma+ Sen_sim	should, i, be, follow, manual, 1	be, -PRON-, require, to, follow, manual, 1	7.4
Tok_sim	following, manual, 1	required, follow, manual, 1	10.0
Lemma+ Tok_sim	follow, manual, 1	require, follow, manual, 1	25.0

Table 3.6 shows algorithms and their weights produced for a Q-Q pair.

4. Experimental Setup

All experiments were done on 5 different slack channels used by some of the students and staffs in the University of Adelaide.
To get history and other useful events of a channel in slack, one can use either a custom integrated [12] bot, which will grant the bot to access all API calls available by Slack. Another way would be to create a Slack app which allows a bot user to exist. However, this way will reduce the possible API calls that can be made by the bot in the app. For research purposes where there are lots of training data available in one team, going with a custom integrated bot is suggested.

4.1 Question Extraction

After setting up a custom bot, history of each of the 5 channel specified were extracted along with its time. The original history of each channel was then pre-processed to remove all the unnecessary details, code snippets, links, emojis and mentions of other users. Spacy's *word.is_link* function came handy while removing links. Next, each processed history were passed on to the question extraction phase where any symbols or punctuation marks were removed before being marked as a question. However, point to be noted, any symbols that resembled an equation, mathematical formulae or code symbols i.e '+', '-', '=', '#', '@', where added back on after being marked and saved in their own separate file along with the time. The files produced, containing both marked questions and unmarked sentences were then compared with their respective annotated files containing the same processed history. From this we got how many rules correctly identified a question. Precision, recall and F1 values are then calculated in order to answer RQ1.

4.2 Detection of Repeated Questions

Once the marked questions were stored in files, different types of queries were asked to each of the 5 channel. Q-Q pairs were made from each query and their respective channel's questions, and were passed on to the 7 mentioned algorithms to produce similarity weights. The weights achieved by different algorithm were then analysed to come up with a well defined threshold for each algorithm and then repeat the process with their respective thresholds. Seven different queries were asked all together in the testing phase that covered various edge cases. Table 4.1 shows what the queries were and their reasons. From this we hoped to answer RQ2 by finding F1 values for all the algorithms used. If an algorithm's result overlaps with others, then only the best will be chosen. From the best remaining, we will try and see if a combination of the algorithms can give a better F1 value to work with.

No.	**Queries**	**Channel**	**Reason**
Q1	how to set up wireless-n300 in bridging mode	buddy-external	very specific question
Q2	how breadboards works	buddy-external	vague question
Q3	picture ingenuity presentation	sanfl	question that's not there
Q4	sprint	sanfl	1 keyword
Q5	croning	maptek	1 keyword in a different form, (actual form was cron)
Q6	picture ingenuity presentation	defence-external	multiple keywords
Q7	how to set up wireless-n300 in tracing mode buddy	buddy-external	subtle difference in question

Table 4.1 shows testing queries and their reason.

5. Results

The results achieved overall for both question extraction and detecting repeated questions were comparable with previous studies.

5.1 Question Extraction

After comparing the result of question extraction with the annotated files, the minimum F1 [13] percentage achieved was 79.7%. Table 5.1 shows the precision, recall and F1 percentage of our algorithm on 5 different slack channel. And table 5.2 shows similar percentages but with algorithms used by different papers.

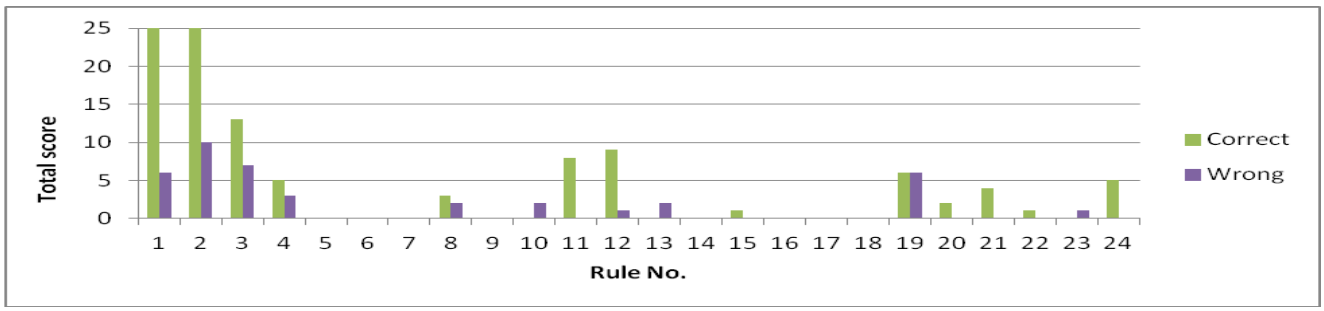
Channel Name	% Prec	% Rec	% F1
Buddy	83.0	92.0	87.3
Defence	75	85	79.7
Maptek	96.6	96.6	96.6
Sanfl	84	86.6	85.3
Slack-faq	83	93.8	88

Table 5.1 shows precision, recall and F1 % of our algorithm on different slack channels.

Other Algorithms	% Prec	% Rec	% F1
SM [14]	83.0	92.0	87.3
FAQPOF [5]	96.6	96.6	96.6

Table 5.2 shows precision, recall and F1 % of other algorithms.

When looking at what rules were triggered the most when detecting questions, it was found that question mark ('?') did play a crucial role in detecting questions, being correct 97.5% of the times. The graph 5.3 shows the score of each rule, where rule 1 and 2 got 233 and 82 respectively.



Graph 5.3 shows the score of a rule.

5.2 Detection of Repeated Questions

The 7 chosen queries were passed through our detection algorithms and the a summary of the results can be seen on section 10. The best threshold, along with its column, are also highlighted. To answer RQ2 from our workspace, less capable algorithms were removed to allow better overall F1value. Sentence similarity and Lemma + combine similarity algorithms were removed as combine similarity algorithm and lemma + sentence similarity gave lesser amount of **total** repeated questions respectively (results of (f) (Q2,2) are better than (a)(Q2,2) and results of (g)(Q1,2),(Q2,2) were better than (b)(Q1,2)(Q2,2) respectively). After analysing the results, the thresholds, shown in table 5.4 , were selected. And table 5.5 shows the precision, recall and F1 value.

No.	Algorithm	Threshold/arbitrary unit
1	Lemma+Sentence_Sim	22
2	Token_Sim	30
3	Lemma+Token_Sim	4
4	Combine_Sim	16
5	TFIDF	38

Table 5.4 shows final algorithms used and their threshold.

Algorithms	**Prec %**	**Recall %**	**F1 %**
Final (all algorithms)	83	83	83
TFIDF	66.7	33	44.1
Token_sim	100	66.7	80
Lemma+Token_sim	83	83	83
Lemma+Sen_sim	100	50	66.7
Combine_sim	100	66.7	80

Table 5.5 showing precision, recall and F1 scores.

6. Struggles and limitations

This project had essentially two parts of work, first was the research part where natural language processing (NLP) was used to process sentences to extract questions and detect repeated questions; second was to create a Slack bot that encapsulated these features and provided an easy interaction for users. A reasonable amount of time was also lost due to annotating such a huge number of sentences manually. Having more than one annotator would be desirable as this shares the workload and also allows comparisons of the annotated sentences done by different annotators to be made.

One of the other early struggles faced was to decide whether to use only one programming language to create server for the bot and do NLP or not. It was suggested by Slack to use Nodejs to create the server as it had extended public libraries on github. However, javascript, which builds nodejs, did not have sophisticated libraries for NLP. It was decided that either Standford CoreNLP or NLTK or Spacy were to be used with a javascript wrapper to avoid having to switch between different languages. None the less, we ended up using 2 languages, javascript and python's Spacy, as the installation process of the wrappers for the NLP libraries were not well documented and had too many build errors.

Another major struggle was shifting from a bot that was available locally to our team to creating a bot, inside a Slack app, that was publicly available. This meant that huge amounts of previous, local bot, code needed refactoring and new code with specific url endpoints were required. Along with this, Slack also desires their apps to complete Oauth2 verification. Oddly enough, most of the tutorials regarding Slack authentication were done in python and not javascript. So converting that python code in to javascript near the very end of the project was quite a struggle. Another Slack's requirements was that all urls used by the app's bot needed to be a SSL link. This meant buying a domain name, an external server and also a SSL certificate, which can be a struggle to get and costly at the same time. Although we have managed to get a domain and server up and running, getting the SSL link was unsuccessful. Therefore, an alternate to this was to use *ngrok* [15] to create an ssl tunnel, which was only online when we were logged on to the server. Once offline, the secure tunnel gets destroyed.

Finally when detecting similar questions, the training data did not have many repeated questions. Therefore, we had to manually create repeated questions. And due to the fact that we had to manually analyse 7 queries passed on to 7 algorithms, in 5 different channels, it became a lengthy process and not much time was left for more testing.

7. Conclusion and Future Work

From the results it can be concluded that this project is still unfinished and will need some more work to actually get the expected end results. We were successful to answer to RQ1, as our results showed that a combination of LSP and rule based approach would be sufficient in detecting most questions asked in a chat channel. RQ2 was also partially answered, as the results shows that a mixture of algorithms, with certain threshold, would best detect

repeated questions. We were also successful in creating a Slack FAQBot that can be used publicly.

The unfinished parts were that, we were unable to extract answers for questions, RQ3, and had to fall on our contingency plan, answering repeated questions with a link that takes the user to the time where the first question was asked. Another unfinished part was the fact that our bot could not be live all the time, but only when we are logged on to the server.

Future work on this project may look to handle the unfinished parts of this project. That includes getting an SSL link and doing more research on how to extract answers from a channel, as it may not be similar to papers where the studies were done on online forums.

Other areas of improvement would be to use machine learning to detect questions, instead of our LSP and rule based approach. More similarity measures could also be added along with the existing ones like Jaccard or cosine similarities. We suggest using python for both server and backend programming.

At the very end, future work may involve making the bot more user friendly and provide support to users if required.

8. References

[1] Scott Gerber, 9 Most Effective Apps for Internal

[2] Slack Help Center, https://get.slack.help/hc/en-us/articles/115004071768-What-is-Slack-

[3] Slack API Calls, https://api.slack.com/methods

[4] Liu Yang, Minghui Qiu, Swapna Gottipati, Feida Zhu, Jing Jiang, Huiping Sun, Zhong Chen, CQARank Community Question Answering, 2013,2-3.

[5] Gao Cong, Long Wang, Chin-Yew Lin, Young-In Song, Yongheng Sun, Finding Question-Answer Pairs from Online Forums, 2008, 1-8

[6] Kyle Dent and Sharoda Paul, Through the Twitter Glass: Detecting Questions in Micro-text, 2011, 1-6.

[7] Baichuan Li1, Xiance Si, Michael R. Lyu1, Irwin King , and Edward Y. Chang, Question Identification on Twitter, 2011, 1-4.

[8] Huizhong Duan, Yunbo Cao, Chin-Yew Lin and Yong Yu, Searching Questions by Identifying Question Topic and Question Focus, 2008, 1-7

[9] Rajaraman, A, Ullman, J. D, Mining of Massive Datasets, 2011, 19-20.

[10] Palakorn Achananuparp, Xiaohua, Hu Xiaohua, Zhou Xiaodan Zhang, Utilizing Sentence Similarity and Question Type Similarity to Response to Similar Questions in Knowledge-Sharing Community, 2008, 1-8.

[11] MATTHEW HONNIBAL, Introducing Spacy, 2015

[12] Slack Bot Users, https://api.slack.com/bot-users

[13] What is F1 value, https://en.wikipedia.org/wiki/F1_score

[14] S. M. Harabagiu and A. Hickl. Methods for using textual entailment in open-domain question answering. In *Proc. of ACL*, 2006.

[15] NGORK, https://ngrok.com/docs#bind-tls

9.Rules for question extraction

1: [["?"]],
2: [["QW"], ["AW"]],
3: [["PW"], ["PMW"]],
4: [["PW"], ["have"]],
5: [["have"], ["PW"]],
6: [["have"], ["DT"], ["PW"]],
7: [["any"], ["idea"]],
8: [["is"], ["NN"]],
9: [["is"], ["there"]],
10: [["are"], ["there","you","those","these"]],
13: [["are"], ["NN"]],
12: [["were"], ["there","you","those","these"]],
13: [["but"], ["QW"], ["AW"]],
14: [["but"], ["WHQ"], ["QW"]],
15: [["why"], ["OW"]],
16: [["why"], ["IW"]],
17: [[["WHQ"], ["to"]],
18: [["why"], ["IW"]],
19: [["whether"], ["AW"]],
20: [["-","VBZ"], ["WHQ"], ["to"]],
21: [["wonder"], ["IW"]],
22: [["PW"], ["is"]],
23: [["PW"], ["are"]],
24: [["-", "VBZ"], ["WHQ"], ["QW"], ["AW"]]

Below are the categories and their keywords.

intermediate words (IW)
Is
are
so
you
but
any
if
have
had
Has

5W1H words (WHQ)
what
where
how
why
which
whom
whose
wherever
when
Who

after words (AW)
that
i
you
it
this
there
are
anyone
one
any
anybody
doing
anything
we
someone

punctuation (PUNC)
! . ,

opposite words (OW)
cant
not

questions words (QW)
can't,
could
can
would
will
do
does
did
wonder
were
should

past myself words (PMW)
am
micellious words (MW)
getting
having
outputting

opposite words (OW)
cant
not
problem words (PW)
problem
error
help
question

questions words (QW)
can't,
could
can
would
will
do
does
did
wonder
were
should

Figure 9.1 shows all the possible categories with their keywords and symbols

10. Repeat Question's Graphs

	1	2	3	4
Q1	Y/2/16	Y/2/2	Y/2/2	Y/0/1
Q2	Y/1/15	Y/1/5	Y/1/3	X
Q3	Y/!/10	Y/! /3	X	X
Q4	Y/1/2	X	X	X
Q5	X	X	X	X
Q6	X	X	X	X
Q7	Y/1/10	Y/1/2	Y/1/2	Y/1/1

(a) Sentence similarity

	13	14	15
Q1	Y/2/4	Y/2/3	Y/2/2
Q2	Y/1/3	Y/1/1	Y/1/2
Q3	Y/! /1	X	X
Q4	X	X	X
Q5	X	X	X
Q6	X	X	X
Q7	Y/1/4	Y/1/3	Y/1/2

(b) Lemma + combine similarity

	36	38	40
Q1	Y/2/2	Y/2/2	Y/2/2
Q2	Y1/3	Y/1/1	Y/1/1
Q3	X	X	X
Q4	X	X	X
Q5	X	X	X
Q6	X	X	X
Q7	Y/1/3	Y/1/2	Y/1/2

(c) TFIDF

	29	30	31	32
Q1	Y/2/2	Y/2/2	Y/2/2	Y/2/2
Q2	Y/1/1	Y/1/1	Y/1/1	Y/1/1
Q3	X	x	X	X
Q4	Y/1/1	Y/1/1	Y/1/1	Y/1/1
Q5	X	X	X	X
Q6	X	X	X	X
Q7	Y/1/2	Y/1/2	Y/1/2	Y/1/2

(d) Token similarity

	3	4	5	6
Q1	Y/2/8	Y/2/5	Y/2/4	Y/2/3
Q2	Y/1/8	Y/1/6	Y/1/3	Y/1/1
Q3	Y/! /11	Y/! /5	Y/! /1	X
Q4	Y/1/2	Y/1/2	X	X
Q5	Y/1/2	Y/1/1	X	X
Q6	X	X	X	X
Q7	Y/1/9	Y/1/6	Y/1/4	Y/1/3

(e) Lemma+ token similarity

	8	16	24	30
Q1	Y/2/2	Y/2/2	Y/2/2	Y/0/1
Q2	y/1/2	y/1/1	X	X
Q3	X	X	X	X
Q4	y/1/2	Y/1/1	X	X
Q5	X	X	X	X
Q6	X	X	X	X
Q7	Y/1/2	Y/1/2	Y/1/2	Y/1/1

(f) Combine similarity

	21	22	23	24	25
Q1	Y/2/2	Y/2/2	Y/2/2	Y/2/2	Y/2/2
Q2	Y/1/4	Y/1/2	Y/1/2	Y/1/2	Y/1/1
Q3	Y/! /1	X	x	X	X
Q4	X	X	X	X	X
Q5	X	X	X	X	X
Q6	X	X	X	X	X
Q7	Y/1/2	Y/1/2	Y/1/2	Y/1/2	Y/1/2

(g) Lemma + sentence similarity