NTNU
Norwegian University of
Science and Technology

Question analysis of coding questions on StackOverflow

130533 - Knut Lucas Andersen

31-05-2016

Master's Thesis
Master of Science in Applied Computer Science
30 ECTS
Department of Computer Science and Media Technology
Norwegian University of Science and Technology,

Supervisor 1: Assoc. Prof. Simon McCallum
Supervisor 2:

Preface

This is my Master thesis concluding the two years spent at NTNU Gjøvik: Master Applied Computer Science - Web, Mobile, Games track. The thesis was carried out during the spring semester 2016, from January to the end of May.

The main concept for the thesis was based on discussions with supervisor. The original plan was to create a Chat Agent that could answers students questions and give feedback to their question quality, by using StackOverflow as a knowledge base. However, during the Master thesis project presentation, other professors noted that the scope of the project was to large for a Master thesis. The thesis were therefore narrowed down to focus on coding questions posted on StackOverflow, in an attempt to evaluate question quality and predict the future votes for a given question.

31-05-2016

Acknowledgment

I would like to thank the following persons for their help and support during these years. It would not have been possible without them.

My supervisor, Simon McCallum, for understanding my difficult situation and for his patience and helpful advices on how to proceed so that I could complete my Master thesis.

Mariusz Nowostawski for his advice on how to get started with text processing and ideas for building the SVM model.

Rune Hjelsvold for helpful advice in relation to text analysis.

My best friend, Njål Dolonen, for always being there for me, and helped me get through this.

My grandmother, Mimmi H. Underland, may she rest in peace. None of this would have been possible without your support, understanding, love and care. This is for you.

I would also thank my family and friends for believing in me and supporting me through this.

K.L.A

Abstract

When you have a question you need an answer to, the solution many uses is to go online and use Google or another type of search engine. However, the degree of difficulty in regards to finding an answer varies with the problem you need an answer to. Today, the Internet offers a wide range of resources to acquire new knowledge, everything from encyclopaedias to blogs, forums and Question-Answer communities. However, when you are writing code, or developing complex programs, it can be easy to get stuck and not figure out why something stopped working.

Depending on the complexity of the problem, the only solution might be to go to an online community for help. One such community is StackExchange (consisting of 154 different communities), which is home to StackOverflow. A requirement for all StackExchange sites is that questions should be of good quality. The quality of a question is measured by use of votes and reputation. The votes are used to grade the question (or answer), whereas the reputation belongs to the user.

By using Machine Learning (ML), I wanted to see if it was possible to predict whether a new question would be viewed as a good question on StackOverflow. This was done by using the dataset containing all posts on StackOverflow and the ML method Support Vector Machines (SVM) (by using scikit-learn for Python). A lot of time was used understanding how scikit-learn worked and how to properly implement SVM to analyse the questions.

One of the interesting finds was that the average down-vote score for 10,000 was 7. To develop the feature detectors, a total of 100 good and bad questions were looked at to see if there was anything that stood out. It was a lot harder to find anything significant for what was a good question, but it was easier to spot the bad questions (e.g. homework, no code-samples, no explanation to what had been done before, etc.).

Before any processing was done, the SVM had a vocabulary of 69,766 features. To reduce this amount, stop-words were added (reducing to 69,462 features) and removal of numbers and hexa-decimal values (27,624). The most significant change was when the document term frequency were set to 1%, which gave a total of 440 features.

Write something more about feature detectors, results, etc.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic

StackOverflow is a part of the StackExchange community, where each community is related to a specific field with domain experts. However, a requirement is that the questions are of good quality [1, 2, 3]. Through peer-review, the quality of questions are filtered by using votes. If a question is good, it gets up-voted, if it is bad, it gets down-voted. The votes is only for the current question, but they still have an impact on the user, by use of reputation [4]. If a question is bad, it can be closed or deleted (e.g. duplicate, off topic, unclear, etc) [5].

The goal of this thesis is to research and analyse coding questions posted on StackOverflow. Most systems focus on finding good answers for the question asked, and not if it is a good question. It would therefore be interesting to see if it was possible to develop a system that could predict the quality of a question before posting it on StackOverflow.

1.2 Problem description

Many systems focus on finding the answer matching the users question, and not on the quality of the question asked. Many users have a negative view on StackOverflow due to their questions being down-voted or closed, because they simply do not meet the standard. Being able to ask a question that is considered good would be helpful both to the community and the user, since it would help the user be better at asking questions. Furthermore, this could also be used in education to help computer science students improve their questions when seeking help.

1.3 Research questions

- What defines a good (coding) question?
- Can we predict a questions quality by using SVM?
- What type of features increases the accuracy of the SVM?

1.4 Methodology to be used

To be able to solve this problem, one needs to understand how text classification is done, and what methods others have used for this. A requirement is also to have a dataset that contains the questions that have been asked on StackOverflow. This is achieved by using the dataset that is available through StackEx-

changes archive[1] [6]. Stackoverflow was started in 2008, meaning it now contains approximately 8 years of data, which has been continuously reviewed by domain experts.

Considering the size of the dataset and the amount of questions that are posted on StackOverflow, going through all questions manually would be to big a job for such a limited timeframe. Therefore only a selected amount of questions was looked at to get an overview to see if it was possible to see any differences between good and bad questions.

To analyse more questions a program was developed in Python, based on SVM. Since SVM is complex, the library scikit-learn [7] was used. In the beginning the word quality was tested by using term frequency, and at the end feature detectors were developed to see if it was possible to further enhance accuracy.

To be able to have proper backup, and also be able to see the development process, the program was uploaded to a GitHub repository[2]. In addition to the source code, the repository also contains an SQL script for creating the database tables[3], the sampled dataset that was used (.csv file) and the created SVM models.

1.5 Justification, Motivation and Benefits

The quality of a question is based on the votes given by the users of the StackOverflow community. Therefore the vote score was used to label the question as either good (score > 0) or bad (score < 0). This is justified by the fact that you can draw a comparison between the peer-review process used in academia and the peer-review process used in StackOverflow. The process starts with someone posting a question, which is then read, scored and given feedback (through comments or answers). The feedback can then improve the question through edits by the poster, based on the feedback given (e.g. adding additional information or explaining how the given problem(s) occurred).

Scikit-learns SVM implementation is based on libsvm [8], but it is simpler to use. In addition, scikit-learn focuses on code quality[4], to ensure that everything works as it should [7, p. 3].

It is not a lot of research being done related to asking and defining good questions. Although the focus is only on programming questions, it can still be useful to improve question quality. Even if you cannot predict if it is a good question,

[1] StackExchange dataset: https://archive.org/details/stackexchange (Downloaded 30. March 2016).

[2] GitHub repository: https://github.com/klAndersen/IMT4904_MasterThesis_Code

[3] Note that this only creates the database and the table structures, it does not contain any of the StackOverflow data

[4] Coverage on their GitHub repository was 94% on 06. May 2015.

you can still avoid asking a bad questions. This in turn can help to improve the question quality (at least acceptability), and avoid asking questions that will be closed or ignored on StackOverflow.

1.6 Limitations

The greatest limitation is the time available. The total dataset has a size of 141,8 GB (but that is for all the StackOverflow data, not just the questions). Filling the database, retrieving the relevant questions and training the SVM takes time[5]. Furthermore, it also takes a lot of time to read through questions and look for differences between good and bad questions. Developed feature detectors may not have that much impact on accuracy, which means more time must be used on re-training and finding other features that can be of interest.

1.7 Thesis contribution

This thesis contribution can be summarized as to the following: Predicting (programming) question quality by using Artificial Intelligence (AI) and ML to improve the questions quality. Stackoverflow was started in 2008, which means that it now contains approximately 8 years of data, which has been continuously reviewed and judged by domain experts. Yet, many still fails at asking the good questions. Many are of course ignored or down-voted because they are interpreted as homework, but there are also a lot of users that just do not know how to ask a good question. Hopefully this research can help with that, so that future users (and students) can benefit to improve the quality of their questions.

1.8 Thesis structure

The following is the structure of this thesis:

- Chapter 2: State of the art and relevant research
- Chapter 3: Methodology
- Chapter 4: (unused placeholder)
- Chapter 5: Discussion on development, the thesis and limitations
- Chapter 6: Conclusion and suggestions for further work

[5]Between 2-3 hours after feature reduction

2 State of the art

basically a summary of existing and relevant research for this thesis

2.1 Text classification

a more general introduction to text classification and the AI techniques used
e.g. papers comparing systems to use as argument for why svm was chosen

2.2 Question-Answering (Q/A)

i'm thinking this needs to be split into sub-subsections or several sub-sections, mainly because I'm a bit unsure what I should put here
the following are some examples:

- question-answering techniques and analyses
- ontology and semantics
- StackExchange and other online communities for qa (potentially separate section)

2.3 Support Vector Machines (SVM), LIBSVM and Scikit-learn

basically papers that have used (lib)svm for text classification, qa, etc

3 Methodology

3.1 Dataset and MySQL Database

3.1.1 Dataset

The dataset contains all information that is currently available in the StackExchange community (at the time the dataset was created). The following is a list of the tables found in the dataset:

- Badges: Badges awarded to users.
- Comments: Comments given either to a question or an answer.
- Posts: Posts on StackExchange, this contains both questions and answers.
- Posthistory: The history of a given post (e.g. edits, reason for closing, etc.).
- Postlinks: Link to other Posts (e.g. duplicates).
- Users: Information about the given user registered at the given community.
- Votes: Type of vote given to a Post (e.g. up/down, vote to close, etc.).

In the beginning, the dataset that was used was downloaded in August 2015. However, since this turned out to be outdated, the latest dataset was downloaded from (https://archive.org/details/stackexchange) on 30. March 2016. The dataset comes in zip-files, where each zip-file contains all the rows found in the given table. These rows are presented in an XML file, as shown in Listing 3.1.

Listing 3.1: Content in stackoverflow.com-Tags.xml

```
<?xml version="1.0" encoding="utf-8"?>
<tags>
<row Id="1" TagName=".net" Count="227675"
ExcerptPostId="3624959" WikiPostId="3607476" />
<row Id="2" TagName="html" Count="511091"
ExcerptPostId="3673183" WikiPostId="3673182" />
...
</tags>
```

3.1.2 MySQL Database

In the beginning, the issue was getting access to the file and see how it looked like. Since most of these XML files had a large file size (ranging from 3,9 MB to 71,9 GB) none of the editors could open them. Attempting to open them through Python code also failed, since there was not enough memory to process everything. The only solution was therefore to create a MySQL database that could contain all the data.

Setting up the MySQL database was not a straight forward process. The operative system I was running was Arch Linux, where they had switched from using Oracle's MySQL to MariaDB[1]. One of the main problems was the available storage space[2] and the varying file sizes. Some of the issues were mainly connection timeout, no more disk space and connection loss (e.g. "Error Code: 2013. Lost connection to MySQL server during query"). To avoid losing the connection to the database, the timeout values had to be changed in MySQL Workbench (shown in Figure 1).

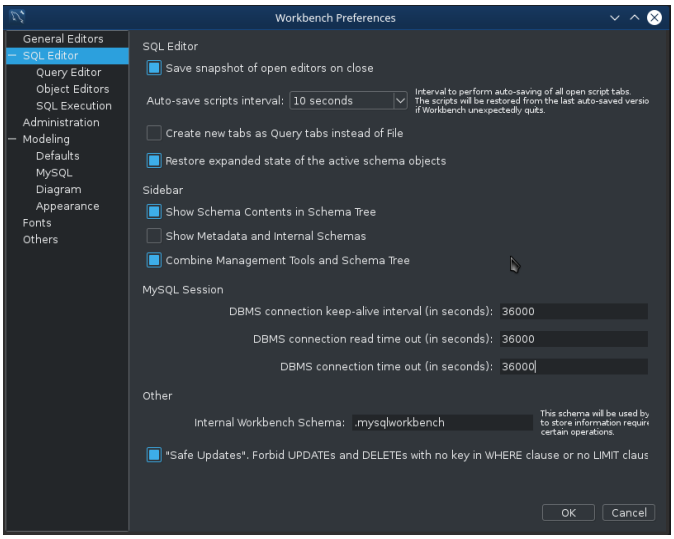

Figure 1: MySQL Workbench: Setting timeout values to avoid connection loss

The next problem was the lack of disk space. MySQL by default stores all databases and belonging tables in /var/lib/mysql/, and it also creates temporary backup files (where the file size is equal to the size of the current database). Since the default folder for temporary files was on /root, the disk space was used up in less than 30 minutes. Therefore, two things needed to be done. First, disable the storage of temporary files, and secondly change the storage location for the database. The problem when tinkering with the configuration file is that things easily break. Which is what happened, and a clean install was needed for both MariaDB and MySQL (the changed settings can be seen in Listing 3.2). The final step was to create symbolic links that linked the database to the location where the tables where stored (this has to be done before creating the tables, if not MySQL Workbench will store the tables in /var/lib/mysql/)[3].

[1] See https://wiki.archlinux.org/index.php/MySQL.

[2] The HDD with Arch Linux installed had a disk size of 500 GB, with four partitions; root, var, swap and home. 40 GB was used for /root and /var, 12 GB was used for swap and the remainder was used for /home.

[3] It should be noted that after an upgrade of MariaDB, MySQL and MariaDB could no longer

Listing 3.2: Changes made to config file: /etc/mysql/my.cnf

```
# disable storage of temporary files
#tmpdir = /tmp/
# disable storage of log files
#log-bin = mysql-bin

# set directory for storing database files
datadir = /home/mysql
```

Listing 3.3: Load XML file into a table in the MySQL database

```
LOAD XML LOCAL INFILE
path_to_xml_file
INTO TABLE db_table
ROWS IDENTIFIED BY '<row>';
```

Listing 3.3 shows how the files were loaded into the tables, and the complete database can be seen in Appendix A.3, p. 16. Since the Posts table is large (~29,5 million rows) and it contains both questions and answers, two new tables were created; "posvote_Posts"[4] and "negvote_Posts". posvote_Posts contains questions with a score higher then zero (score > 0) and negvote_Posts contains all questions with a score lower then zero (score < 0).

3.2 Development process

When starting the development, the focus was on retrieving the data from the database, and processing it for text analysis. To be able to store all the retrieved columns and the belonging rows without creating object classes, the pandas.DataFrame[5] was used.

The questions retrieved needed to be processed before any analysis could be done. The reason for this is because the questions was written as HTML (including HTML entities). An example is shown in Listing 3.4. Every question starts with the <p> tag, and if the question contains code samples, these are wrapped with a <code> tag. To convert the HTML text into readable text, a HTML parser class was created (based on answer by [9]).

find the tables, even if they still were in the /home/mysql/ folder. It is therefore advisable to dump the database after inserting all the tables, since it goes a lot faster to restore the database from dump rather than insertion from XML files.

[4]The Posts table has a file size of ~43,6 GB, whereas posvote_Posts file size is ~11,2 GB. negvote_Posts has a file size of ~1,33 GB.

[5]Pandas: http://pandas.pydata.org/.

Listing 3.4: Question before HTML is removed (Question ID: 941156)

```
<p>
Why do we need callbacks in ASP.NET or any server side technology?
</p>&#xA;&#xA;<p>One answer can be, to achieve asynchronous calls.
</p>&#xA;&#xA;<p>But I am not satisfied with this answer.</p>
&#xA;&#xA;<p>Please explain it in a different way.
</p>
&#xA;
```

To process the questions, CountVectorizer from scikit-learn was used. CountVectorizer uses the vocabulary found in the text and counts the frequency of each word [10] [11, 4.2.3]. When looking at this vocabulary, a lot of of un-important words was found (a lot which came from the code samples) provided in some of the questions. At first all code samples were removed from the text, but later on they were replaced with the value 'has_codeblock', indicating that this question contained one or more code samples. This was achieved by using a combination of lxml[6] and bs4[7] (BeatifulSoup). lxml was used to construct an XML tree containing all the tags (to be able to retrieve the content by searching for a given tag), and bs4 was used for beautifying the HTML (since in some cases an error was thrown complaining about "Missing end tag").

However, for some questions, part of the text was lost, and for others, some <code> tags was not removed. On inspection, it was found that the trailing text following the <code> samples was stored in a .tail attribute. Since the <code> was removed, the .tail attribute was also removed. This was fixed by storing the the content of the <code> .tail attribute into its <parent>[8] (where <parent> is the tag that contained the given <code></code>) .tail attribute. As for the non-complete removal of <code> tags, this error mostly occurred for code samples that contained XML or HTML code[9], because the lxml parser failed. The solution was to replace the lxml parser with bs4 and just change the content of the <code> tag to the value 'has_codeblock'.

Considering the size of the dataset, and that the source code was hosted on GitHub, I was hesitant to store the training data in a separate file. However, when loading 20,000 samples from the database with a 'WHERE' parameter, things tend to go more slow. At this point, it was decided to try to dump the loaded data from the database to a file. This was achieved by using pandas.DataFrame.to_csv[10].

[6] lxml: http://lxml.de/

[7] BeatifulSoup: https://www.crummy.com/software/BeautifulSoup/

[8] It was also necessary to check if the <parent> had a .tail, if not, the .tail attribute had to be set for the <parent> to avoid the error: "NoneType + str: TypeError".

[9] One example is this question:
http://stackoverflow.com/questions/19535331/print-page-specific-area-or-element.

[10] pandas.DataFrame.to_csv:

At a later point, the unprocessed dataset was also dumped to a CSV file for replicability[11].

Further examination showed that the vocabulary contained a lot of numerical and hexadecimal values, but also a lot of non-English words. The numerical and hexadecimal values were replaced using regular expressions to 'has_hexadecimal' and 'has_numeric'. The non-English words were a bit more troublesome to handle, since these were mainly used to prove a point or show an example of the issue they were having[12]. Attempts were made to filter them out by using corpus.words.words() and corpus.wordnet.synset() from Natural Language Toolkit (NLTK)[13], and PyEnchant[14]. However, WordNet does not have a complete database of all English words, and they all claimed some words were not English even though they were.

The solution turned out to be a lot simpler. Instead of creating filters, the CountVectorizer already had one built in. By adjusting the minimum document frequency (min_df) and set it to 0.01, it would ignore words that did not appear in less then 1% of all documents.

To write:
Using SGD (based on tutorials from scikit-learn)
Testing out different text classification algorithms (SVC, SGD and LinearSVC)
Attempting to make program runnable from command line; e.g. started with opt/argparse, ended with while loop

3.3 Feature sets, attributes and processing

what features were selected and why?
how was data processed (e.g. retrieved from db, converted to scikit-learn format, and so forth)
attributes: length, symbols, question sentence only, code snippet, votes, closed, etc.

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html.

[11] The only change made to the unprocessed dataset was removing the HTML tags.

[12] http://stackoverflow.com/questions/856307/wordwrap-a-very-long-string.

[13] http://www.nltk.org/

[14] http://pythonhosted.org/pyenchant/

Step	Text processing	Vocabulary count	CountVectorizer
1	None	69766	analyzer="word"
2	Stop words	69462	analyzer="word", stop_words="english"
3	Removal of code, hexadecimal and numerical values	27624	analyzer="word", stop_words="english"
4	Minimum document frequency	440	analyzer="word", min_df=0.01, stop_words="english"

Table 1: Feature reduction steps before and after text was processed.

4 chapter4

empty placeholder section

5 Discussions

5.1 Data and Testing

discussion on the data set and how it was tested.
the results and what they showed.
potential improvements, etc.

5.2 Artificial Intelligence (AI) Methods

alternative methods and options (e.g. one could have used ann or k-nn, but as shown in. . .)
not sure if this section is relevant?

5.3 Implementation Architecture

discussion on the code that was written and its functionality
what worked, what should be updated/changed, etc.

5.4 Limitations and other issues

why didn't something work as intended?
why wasn't X completed/implemented?
etc.

6 Conclusion/Summary

6.1 Overview of main results

basically what the title says; a summary of the results

6.2 Further work

additions and updates to the system
new research possibilities based on results

Bibliography

[1] Stackoverflow.com. How to ask (online). 2016. URL: http://stackoverflow.com/questions/ask/advice?

[2] Stackoverflow.com. What topics can i ask about here? (online). 2016. URL: http://stackoverflow.com/help/on-topic.

[3] Stackoverflow.com. What types of questions should i avoid asking? (online). 2016. URL: http://stackoverflow.com/help/dont-ask.

[4] Stackoverflow.com. What is reputation? how do i earn (and lose) it? (online). 2016. URL: http://stackoverflow.com/help/whats-reputation.

[5] Stackoverflow.com. What does it mean if a question is "closed" or "on hold"? (online). 2016. URL: http://stackoverflow.com/help/closed-questions.

[6] StackExchange, I. Stack exchange data dump (online). 2016. URL: https://archive.org/details/stackexchange.

[7] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[8] Chang, C.-C. & Lin, C.-J. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2, 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[9] Eloff. Strip html from strings in python (online). 2009. URL: http://stackoverflow.com/a/925630.

[10] Scikitlearn.org. sklearn.feature_extraction.text.countvectorizer (online). 2016. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[11] Scikitlearn.org. 4.2. feature extraction (online). 2016. URL: http://scikit-learn.org/stable/modules/feature_extraction.html.

A Appendix

A.1 Acronyms

AI Artificial Intelligence. 3, 4

ML Machine Learning. iii, 3

SVM Support Vector Machines. iii, 1–3

A.2 Data sets/Statistical Overview

A.3 MySQL Database

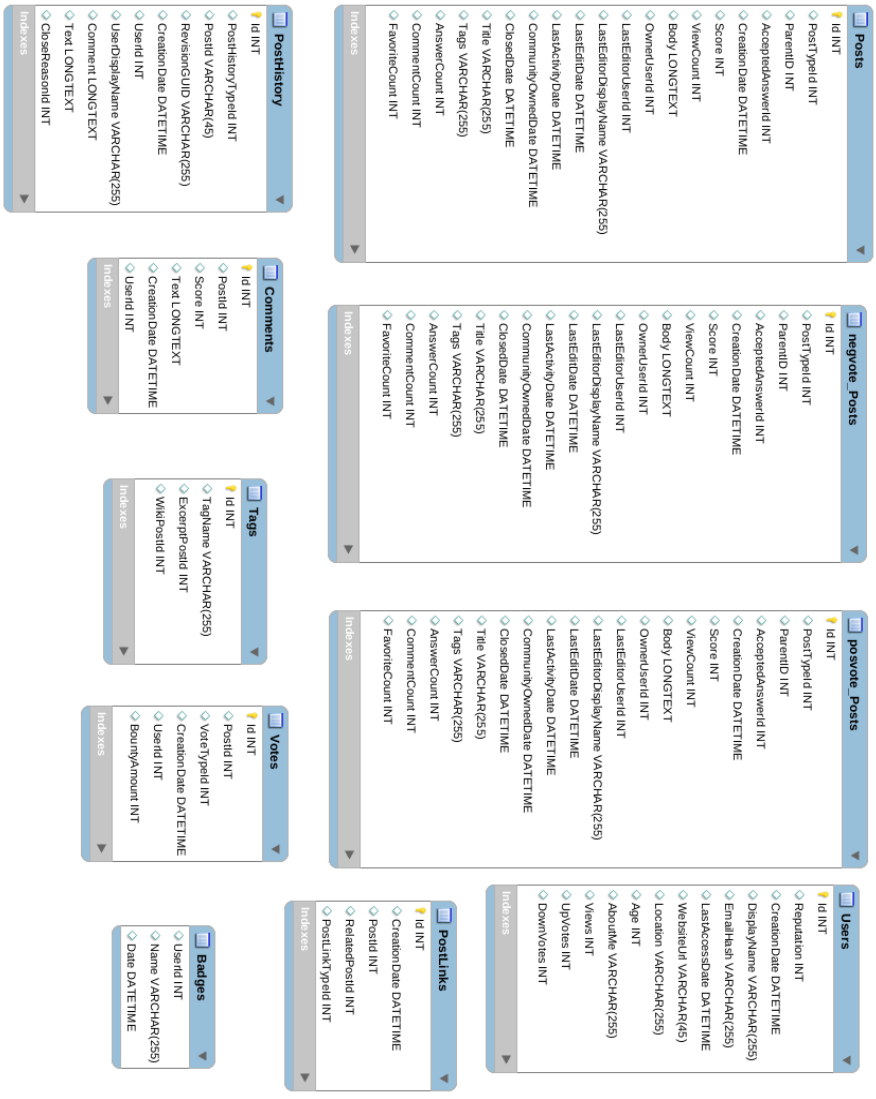

Figure 2: MySQL Database used for dataset