Predicting coding question quality using Stack Overflow ratings

Slide 2: Overview

- Stack Overflow (SO)
- What is a question?
- Support Vector Machine (SVM)
- Methodology
- Experiments and Results
- Summary
- Demo

Slide 3: Stack Overflow

Stack Overflow (SO) was created by Jeff Atwood and Joel Spolsky, and was released in September 2008. They created it to offer programmers a place where they could ask questions and get answers to their questions. To be able to measure quality, they used votes, where users could up-vote posts they found useful, and down-vote posts that were wrong (posts being questions, answers and comments). Furthermore, questions can have multiple answers, allowing users to add updates based on future changes, and also see alternative solutions to their problems.

In fact, Stack Exchange (SE), which were released one year later, is built upon the same model. This means that if you develop a system for SO, you could also expand it to cover all of SE.

SO uses gamification to reward the users for their participation. Gamification means that you use game elements in a context which normally would not be considered a game. In SO, users can be rewarded in various ways, but the three main elements are votes, reputation and badges.

Votes are used as a measurement of the questions (and answers) quality and usefulness, but are only shown on the given post. SO allows you to sort questions by score, and this can also be done for answers. The answers are by default sorted by score, with the exception being questions that has an accepted answer. The accepted answer is the answer which the user who asked found to be correct for their problem.

Reputation and Badges follows the users, where one can compare Badges to achievements in games. Reputation is not only used to show how much you have participated, but it also restrains the amount of freedom you have on the site (e.g. commenting, voting, answering, etc). Reputation can in fact be used as a measurement of expertise, because there is a limit to how much reputation you can earn daily.

Users can only earn up to 200 points of reputation each day, where an answer gives +10, and question +5. That amounts to posting 20 answers, or 40 questions. After reaching the daily cap, the only way users can earn more reputation, is by having their answer marked as accepted or earning bounties. Bounties can be seen as a currency system. If a user has a question no one answers, or the answers given does not solve their problem, users can trade parts of their reputation for a solution.

SO neither takes nor gives payment to users for posting questions and answers. Imagine how much it would cost if someone were to be paid to rate each question and answer. On Amazon Mechanical Turk, the lowest price for rating I could find was 0.15$, or 1.22NOK. That does not sound like a lot. Well, in the data set I used, there were 11,203,031 questions. That would mean just to get a rating for each question once, you would have to pay over 11 million Norwegian Kroner.

I could not find a good identifier for what a good question is. There is, however, what I would call a bias factor. By this I mean that if a certain amount of people share the same problem, it becomes a good question. Not because of the question asked, but because of the problem needed to be solved. However, these are in most cases weighed up by using the Wiki feature, namely Community Wiki. Community Wiki are usually added to posts that are considered to be helpful to the community (and you can even search for Wiki posts in the search field).

The bad questions where easier to spot. A common denominator was the lack of effort when asking their questions. For the most code related questions, they added a code snippet, and said "This doesn't work. Why?". Other examples were large code examples showing a lot of code that was not related, or no code at all and just showing the error message.

If you went to your teacher with 50 lines of code, and said "This doesn't work. Why?". Do you think s/he would know what was wrong? In most cases, the obvious reply would be "what have you tried?", "what is the expected output" or "what is the error that you are getting". A lot of code examples were also badly formatted and had syntax error. There were also obvious signs of homework/school related topics, and in one question the code even contained the namespace "assignment". SO also wants questions to be unique, so duplicates are often down-voted. There are however some cases where this does not happen, and it could therefore be interesting to see in the long term which would be the most viewed.

Slide 4: Picture of good SO question

"How do you undo the last commit?", +10,493 votes, Community Wiki
In right bottom corner, number is reference to source (last slide)

Slide 5: Picture of bad SO question

"Forcing function to return if false" [locked, closed, off-topic], -154

Slide 6: Picture of Badges on SO

Picture of question badges

Slide 7: What is a question?

Questions can be generalized into either factoid or broad. Factoid questions usually only have a set amount of correct answers, whereas broad questions can have many answers that are all correct.

For education, questions are usually used as a learning tool to either help students learn something new, or through examination to evaluate your knowledge. For research, it could be the goal you are after, because you need to define a problem. You cannot just post a bunch of numbers, and say "These are my results". You need to ask the question "What are these results?", "What does these results tell me?", "What problems arise from these results?".

An interesting reverse situation is a game show from the early 90's called Jeopardy. The slogan for the show was "We have the answers, you have the questions". In this show, players were presented with an answer, and had to ask the question to which gave the answer. This could also be an interesting addition to learning, because what if the exam was not based on the teacher asking you a lot of questions? What if the exam was based on you asking the teacher questions to show that you understood the curriculum. Would you be able to ask hard enough questions to show that you grasped the curriculum?

The goal of Question classification (QC) is to categorize questions, since in most cases the aim is to find the answer to a question. By categorizing it, e.g. PERSON, LOCATION, DATE, you can reduce the amount of answers that could be related to the question.

You have WH-words, which are usually the first word in the question. Examples are "What, which, when, where, who, how, why". Some of which are harder to classify, because they are not as restrained as the others ("What", "Why", "How" and "Which").

N-grams is simply reducing the full text into fragments. These fragments can either be words or the characters in the word, and the 'N' represents the amount of fragments. One of the more known is the Bag-of-words (BOW), which is also called a unigram. BOW simply takes every word, counts the frequency and ignores the order. Bi-gram takes every second word, which means some order is kept. Meaning that for higher N-grams, the more focused you are on the sentence, the higher the N-gram should be.

If you want each word and sentence to be equal, you can remove the case-sensitivity. When comparing characters a large 'A' is not equal to a small 'a'. Semantics can be used for word filtering, e.g. to remove duplicates or synonyms (word with same meaning). E.g. WordNet has a built in function for synonyms called synset. Hypernyms could also be of interest. Hypernyms are words that belong to the same category, sharing a parent-child relationship.

If word reduction is desired, stemming is also an option. Stemming reduces the word to its base-form, e.g. "crying" would be converted into "cry". Text can be split through tokenization, and you can also use stop words, which ignores frequently used words in the given language. You can also extract grammatical properties by using Part of Speech (POS).

Slide 8: Support Vector Machine (SVM)

SVMs are good for solving regression and classification problems, where it attempts to solve a linear problem by using a hyperplane. It is mainly used for binary classification, where the two classes are separated by the hyperplane. Between the hyperplane and the classes, there is a margin, which is the distance between the closest data points from the class and the hyperplane. These datapoints are called support vectors.

SVM consists of four kernels:

- Linear
- Radial Basis Function (RBF)
- Polynomial
- Sigmoid

Slide 9: Methodology

The data set that was used in this thesis was downloaded from the Stack Exchange Archive on March 30th. Originally, I used a data set from August 2015, but found that this was outdated, and I therefore downloaded a newer one. The data set contains XML files, where each element is a row containing the data from their database. Given that some of these files were several gigabytes in size, the only way to get access to the data was to load it into a MySQL database.

Since all the questions and answers are stored in the same table, these had to be separated, since it was time-consuming to run queries against it. Therefore two tables were created (based on the original table), where one table contained all questions with score < 0, and the other questions with score > 0 (since questions with score of 0 was not of interest). A nice library for Python is Pandas. Pandas is a data analysis library, which includes something that is called DataFrame. In addition to maintaining the same structure as it is in the database, it is also supported by Scikit-learn as an array container.

For this development I used Python 3.5, and the development version of Scikit-learn. This was mainly because I had a lot of installation issues, and I had to install it from the GitHub repository. Given the amount of changes between the stable and the development version, and the uncertainty related to when this would become the next stable version, I decided to stick with the development version.

Since all the questions were stored as HTML, they needed to be processed. This was done by creating a HTML parser class, and using BeatifulSoup to account for missing tags and "beautifying" the HTML.

Since there was a lot of code examples, these had to be removed. This was done by checking for the <code> tag, and then simply removing the text. However, this meant that you lost the fact that this question contained code, so this was later replaced with the feature detector 'has_codeblock'. When also looking at the printed vocabulary, I saw that it contained a lot of hexadecimal and numerical values, so therefore these were also replaced with a feature detector called 'has_hexadecimal' and 'has_numerical'. Furthermore, there were a lot of dummy words used (an example is shown on the next slide), which had to be removed. The solution was the minimum document frequency parameter, which can be used to ignore words that appear in either less than a given percentage of the documents or a given number of documents.

To be able to find out what could be good features for separating the good and the bad questions, 200 questions where looked at on SO (based on highest/lowest voted). As previously mentioned, it was easier to spot the bad questions (e.g. homework, bad code examples, etc).

For the homework words, I first attempted to use WordNet.synset, but it only gave three words: 'homework', 'prep', 'preparation'. On Thesaurus, I saw a lot more words listed, and therefore created my own dictionaries using a selection of these words instead. However, one issue were that the word 'assignment' could also be used in relation to programming terms (e.g. assignment operator), and I therefore had to split the Homework feature into two, one for homework words and one for the word 'assignment'.

I also saw that a lot of questions used links, either to link to external resource/tutorial or to show the "possible duplicate" flag.

SO also uses Tags for their questions, where a question can have from 1 to 5 tags total. Tags comes in two variations, what I call external and attached. External are the list of tags (and their synonyms) that exists on the site, whereas attached are the ones the user has linked to their question.

There were three problems with the Tags. First, they were all in lower-case. Second, the attached Tags were wrapped in HTML tags. Third, they needed to be sorted by length. At first, only pure string replacement was used, which replaced each occurrence with the set feature value. However, the problem was that for single character words, e.g. 'C', it replaced occurrences even within the words. So a switch was made to using regular expressions, since string replacement had no filtering. However, a new problem now was that it failed at words containing symbols, e.g. 'C++', because it read the +'s as a regular expression rather than as a word (the '+' as regex meaning 1 or more). This meant that I had to re-add the string replacement for Tags longer than one character, and addition sort by length so that the regular expression did not replace singular occurrences (e.g. making 'C++' into 'has_tag' ++).

To avoid selecting some random parameter values, I used the default values from the Scikit-learn tutorials and GridSearchCV, which uses an exhaustive search. An exhaustive search means that it matches all the set values against each other to find the best match. As for the classifiers, two were selected, Support Vector Classification (SVC) and Stochastic Gradient Descent (SGD).

To have a replicable setting for the results, I used train_test_split from Scikit-learn, which splits the data into a set amount of training and test data. You can also assign a random state, and if the random state value is the same, then the results will be the same for each time you run it with the same data and settings. I also used cross-validation, which does its own validation on the training data, by splitting it into folds. The data is then trained on k-1 folds, and evaluated against the last fold.

Slide 10: Picture of dummy word

Title: Wordrap a very long string

Slide 11: Experiments and Results

A total of six different features were selected:

- Code samples
- Hexadecimal
- Homework (synonyms for homework and assignment)
- Links
- Numerical
- Tags (external and attached), external replaced to many "normal words"
- All features (excluding assignment and external tags)

Four different experiments was done:

1. Unprocessed data set vs. all singular feature, and all questions
2. Unprocessed data set vs. all singular features, and question occurrence only
3. Unprocessed data set vs. selected set of features only
4. Stochastic Gradient Descent (SGD) as classifier

For the first two experiments, the basis was that the feature classifiers should be based on the results from the unprocessed classifiers training. This was to ensure that the results were comparable.

In the first experiment, only numerical achieved a higher accuracy score for its prediction than the unprocessed. The feature for Tags did the worst, being almost 4% below the unprocessed classifier. The classifier using all the features was almost 1% lower than the unprocessed.

To get a better view of the features impact, a new experiment was done, where only questions that contained the given feature was used. In the previous experiment, Hexadecimal had no impact, and the reason was that it was only present in 160 questions. This means it was not even a part of the selection in the first experiment, and is therefore not a good feature. What was also interesting to see was that all of the features was mostly represented for the bad questions, which does explain why some features performed worse here. E.g. Numerical now did worse, with a score 0.5% lower than the unprocessed.

One obvious problem is the classifier for Tags and for Homework. What should have been done here is that they should have been also trained separately. Both of these contain two feature detectors, whereas the classifier using all features excludes two of them (external and assignment).

In the third experiment, two new classifiers were trained. The first was based on simply running an exhaustive search to find the best parameters, and the second also included stemming. Stemming converts the word to its root form, e.g. the word "crying" would be converted to "cry". The hypothesis was that the stemmed would perform better, but as it turns out, it worsened the results. When compared to the unprocessed its score was almost 4% lower, and 3% lower than the non-stemmed classifier. The comparison of the non-stemmed (based on the unprocessed) vs. the non-stemmed that was re-trained achieved a 0.5% higher score because it was better at predicting bad questions.

In the last experiment, a comparison was made between the SVC and SGD. As expected, SVC did better (SGD is usually used when you have more than 10k samples, but it was used in a lot of scikit-

learns text tutorials). However, SGD was better at predicting bad questions, but with a very low maximum document frequency for the unprocessed data set. For the unprocessed, it had set it to use only words appearing in less than 50% of the questions, and 75% for the classifier using all features. Whereas for the SVC classifiers, I had set the min_df to 95%, because I did not want all the common words. However, seeing these results could indicate that it would have been better for the exhaustive search to select these values. It could also indicate that if a larger sample size where used, SGD might perform better than SVC.

add a short explanation to what SGD is

A comparison was also made to see if the system were applicable for other sites within the Stack Exchange community. For this, Tex.StackExchange was selected, with the hypothesis that SO would be more predictive, given that it contains a lot more questions. The Tex.StackExchange did get an accuracy of 99%, however, it failed at predicting any of the bad questions, and it also contained only 93 questions with a score lower than zero.

Slide 12: Conclusion

Stack overflow as a question quality metric

Questions on SO are rated by use of voting and reputation. This means that if you want only the "good stuff", you could select questions that have a high amount of up-votes, and are asked by users with high reputation. SO also has strict guidelines for what type of questions can be posted, and questions that could fall into the bias factor are often marked as Community Wiki. Questions that are bad are often marked with a notable number of down-votes, and are usually closed. Bad questions can also be marked with a set topic, e.g. too broad, off-topic, duplicate, etc. Questions can also be filtered out by using Tags, allowing you to focus on one specific programming topic.

Limitations and issues

Using set values for database/question retrieval.
Using development version of Scikit-learn.
Tags and Homework vs. All features.
Exhaustive search and parameter selection.
The issue with no verbose, hanging/freezing and gridsearchcv; switching to windows and not using all cores.

Further work

- Code blocks, Links and Numerical as a feature set
- Code analysis
- Sentiment analysis
- Version numbering

Slide 13: Demo

Show demo (if time)

Slide 14: Thanks for listening

Slide 15: References