NTNU
Norwegian University of
Science and Technology

Question analysis of coding questions on Stack Overflow

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

Acknowledgement

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 the Support Vector Machines (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

Stack Overflow (SO) is today for many developers a well known Question-Answering (QA) system. However, SO has a high requirement to the questions and answers posted, which is reflected through their voting and reputation system. This peer-review processes can be used as an indicator to a questions quality, where questions with high up-votes can be defined as good questions. In this thesis, a system has been developed using Machine Learning (ML) and Support Vector Machines (SVM) to see if it is possible to predict whether or not a new question will be considered as a good or bad question.

This was achieved by using the Stack Exchange (SE) data set, specifically using the one for SO. Questions were dived into two classes, where bad questions was question with a vote score below zero, and good questions were those above zero. Based on content in the various questions, a set of feature detectors was developed and tested against the raw data set. Surprisingly, the features actually lowered the accuracy score. The raw, unprocessed classifier achieved a score of 79.90%. The classifier using Porter stemming and all features achieved a score of 75.97%, and the classifier without stemming using all the features got a score of 79.12%.

Contents

List of Figures

List of Tables

1 Introduction

Today, many uses the Internet as a resource to find answers to their questions and problems. In the past, one were often restricted to only use keywords and not being able to pose the problem as you would when asking another human being. Most search engines today can handle natural language queries, which makes it easier to find the answer you are looking for. The Internet offers a wide range of resources to acquire new knowledge, everything from encyclopaedias to blogs, forums and Question-Answering (QA) communities. One well known QA community is the Stack Exchange (SE) community, which is built upon the same model as Stack Overflow (SO) [1]. SE has grown large since its release in 2009, and now contains 154 different communities.

As a developer, one often find oneself in the situation that a part of the code does not work, you get weird error messages, or you are simply stuck. This is were SO comes in. SO is a part of the SE community, although SO was actually released before SE. Jeff Atwood and Joel Spolsky wanted to offer programmers a QA site where they could get the answer they wanted without having to read through a lot of text, see others posting "I also have the same issue" or having to subscribe and pay to see the solution [57]. Question (and answer) quality is maintained through the use of a peer-reviewed gamification system, where users are awarded with votes, reputation and badges for their participation [38, 53, 57, 70]. One of the requirements is that the questions should be of good quality [60, 65, 64]. If a question is bad, users can vote to close or delete it (in which the question will be put on hold). A question can be put on hold or closed if they meet any of the following criterias: Exact duplicate (same question has been asked before), off-topic (not related to SO), unclear what is being asked, too broad (e.g. could write a book about question being asked) or primarily opinion-based [9, 62].

1.1 Problem description

Most of the systems that have been developed so far focuses on finding the best answer to a question asked by the user. Few, if any, focuses on the quality of the question being asked. What defines a good question, and can we in anyway predict whether or not a new question posted on SO will be considered good or bad by the community? There are many users who has either a negative view or relationship in regards to SO. Many experience that their questions gets down-voted, closed or even deleted. For some, they simply do not know how to ask an acceptable question. Questions related to homework are one example of questions that

add examples here

are not accepted on SO. There is even a post on Meta.StackExchange discussing whether or not it should be acceptable to use greetings and sentiments in posts [8]. Therefore, the question becomes: What is and is not a valid question on SO?

1.2 Research questions

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

1.3 Methodology to be used

The theoretical background in this thesis is mainly focused on Question Classification (QC) and similar research in relation to SO. What has been the focus of other researchers, and in what way did they proceed to solve their questions? The analysis of the questions are done by using the publicly available database dump, which is available via SE archive[1] [58]. There are several others who have used the same dataset [1, 2, 19, 38, 41, 54, 66, 75]. Taking into consideration that SO was released in 2008, it means that it now contains approximately 8 years of peer-reviewed data. Because of the size of the data set, and the total amount of posted questions, going through all questions manually would be too time-consumings. Therefore only a select few were studied too see if it was possible to identify what separated the highly up and down-voted questions.

The goal was to develop a Machine Learning (ML) learning system which was based on SVM, since many papers document that this has the best classification accuracy for text classification. The methodology therefore also includes a documentation on the development process, and how and why the given features used were selected.

For the sake of replicability, and also be able to undo potential errors, the system is available in a a GitHub repository[2]. In addition to the source code, the repository also contains both the samples that was used (stored in CSV files), and the models that was created.

1.4 Justification, Motivation and Benefits

Many systems focuses only on finding a good answer, and does not ask if it is a good question. As a famous Norwegian saying goes[3]: "A fool may ask more than ten wise men can answer". This means that new research possibilities could be opened up in relation to researching question quality by expanding the system.

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

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

[3] Although its origin comes from a Danish word collection from 1682: https://snl.no/En_d%C3%A5re_kan_sp%C3%B8rre_mer_enn_ti_vise_kan_svare.

Since all the communities within SE is based on the same model, few modifications would be needed to scale the program to be used within the other communities. As noted in several papers [38, 39, 41, 53, 70, 75], question quality is measured based on the amount of votes given. Which can also be compared against the peer-review process in academia, and given that SO targets professionals and experts, using SO as a scientific reference is not that unusual[4]. SE has also been the focus of various researchers these past years [72]. Improving ones own ability to ask better questions can also have a pedagogical effect, which means that this system could be implemented in education.

1.5 Limitations

The selection of questions is only 20,000 (10,000 good and bad), which is a lower number than the number used in. The retrieval of questions could also have been better, since the vote score was based on static values, rather than selecting those with the highest/lowest score. Some features were not very representative (e.g. Hexadecimal, which only occurred in 160 of the 20,000 questions), and may therefore have a negative impact on classifier accuracy. The version of Scikit-learn that was used was the latest development version (v0.18.dev0), instead of the stable. This means that potential bugs and un-finished implementations can have an effect on the prediction and probability (e.g. giving the wrong results). A limitation is also that the focus is only on SO, which means that one would need to make additional adjustments and add more filtering to account for the differences that may occur in each community.

cites

1.6 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. Instead of posting bad questions that can get downvoted or closed, the developed system could be able to give feedback to the questions quality. Furthermore, the research presented could open up for new research in relation to how we ask questions online, and in what ways these best can be analysed. It can also be used for educational purposes, e.g. having questions iteratively improve their question quality by asking the system questions.

1.7 Thesis structure

The thesis is structured as follows. In Chapter 2, relevant research is presented. This includes SO, a definition of what a question is and short about SVM, to give an overview of the current state. The thesis continues with a presentation of the methodology that was used in Chapter 3. This includes information on the data set, the created database and how the development progressed. A short

[4] Posnett et al. [41, p. 1] noted that SO "ranked 2nd among reference sites, 4th among computer science sites, and 97th overall among all websites".

explanation to the selection of feature detectors is also included. Following is a discussion on the choices that were made, the issues that occurred and the results, which is presented in Chapter 4. The thesis ends with a conclusion and suggestions for further work in Chapter 5.

2 Related work

2.1 Stack Overflow (SO)

2.1.1 Stack Overflows (SO) Design

The following lists the factors that were used when designing SO (based on Sewak et al. [53, p. 6-7] and Treude et al. [70, p. 805]):

1. Votes: Questions and answers which are considered good (or bad) by the community can be given a score. This gives a filtering mechanisms, which allows users to ignore answers that are bad or wrong. Furthermore, answers are sorted by votes, and you can also sort questions on SO by vote score (see Figure 1).
2. Accepted answer: If the user asking a question gets an answer that they find satisfactory, they can select it as the "accepted answer". This answer will be the first displayed of the answers, and is also viewable when searching for questions (see Figure 1 and 2).
3. Tags: Each question is associated with a tag[1], which can be a topic, a programming language, a methodology, etc.
4. Badges: Similar to achievements in games, Badges are used to reward the user for their participation.
5. Reputation and Bounty: Currency system for user participation. E.g. voting for questions, getting your answer selected as the accepted one, etc. Bounty is a trade, where if your question goes unanswered for too long, you offer up parts of your reputation to receive an answer.
6. Data dump: Available data dump containing all content available within the SE community [58]. You can either download single files, or everything by using a Torrent client.
7. Pre-Search: Encouraging users to check that their question is not already posted by presenting a search bar when asking a new question.
8. URL keywords and Google: The questions title is included in the URL, allowing it to be processed by search engines. In addition, Google uses their crawlers every 10 second to have the latest updates from in their search engine [16].
9. Critical mass: Before Atwood and Spoolsky launched SO, they invited developers and programmers to participate to have some domain experts available.

[1] A full list can be seen here: http://stackoverflow.com/tags/

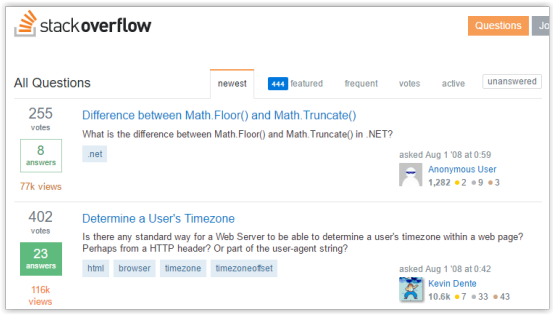

Figure 1: List of questions, where one can see those with an accepted answers are marked with a green background.

Figure 2: Example of a question on Stack Overflow[2]

2.1.2 Stack Overflow (SO) and Gamification

Deterding et al. [10] defines Gamification as "the use of game design elements in non-game contexts", and is the definition this section will be based on. Several papers make notes of the pedagogical and educational aspect of SO [39, 41, 75], and [39, 75] use the term gamification in their paper. One of the founders, Jeff Atwood said in an interview that he wanted users to not just give good answers, but also trick them into improving their communication skills [41][3]. In the course

[2]Source: http://stackoverflow.com/questions/178325/checking-if-an-element-is-hidden

[3] From this interview:
http://www.wired.com/2012/07/stackoverflow-jeff-atwood/2012.

IMT4007 Serious Games Simon McCallum and Marius Nowostawski, presented their game GoRad, which was based on us students reading articles and posting questions which were voted on. The SO system awards users based their activity by using votes, reputation and badges [53, 38, 70, 59, 63].

If you look at SO as a game, users could be represented as the four player types presented in [33, p. 3]: Achievers, Explorers, Socializers and Killers. These player types can be used as a representation[4] for the various users of SO. Achievers are there for the reputation and badges, socializers are to interact, discuss and share knowledge. Explorers might find joy in looking at various topics, or searching for unanswered questions. The only exception would be the "Killer" type. Maan [33, p. 3] defines Killers as those "... who always want to create trouble/problems for other participants" (although this would be more fitting for the term "Griefer"). In an online QA system (or Internet in general), these are what are commonly referred to as "Trolls" [14, 4]. However, due to the system used in SO, Trolls would not be able to survive, simply because the reputation controls what you have access to [61]. If you down-vote a post, you lose reputation. If your post gets down-voted, you also lose reputation. Users who are not willing to follow the guidelines can be locked out of SO [3]. However, today there is a lot of blogs complaining about the current structure of SO, who claims that a lot of the moderators are trolls[5].

2.1.3 Stack Overflow (SO) and reputation

Many QA sites includes domain experts to ensure some quality is upheld, and uses voting and reputation as a quality measurement [2]. Furthermore, questions topics, page views and votes can be used by search engines as a ranking mechanism, and it helps users to find the answers they are looking for. Anderson et al. [2] identifies two principles for the answer process. This process starts with the question being filtered down through the users, starting with domain experts. If the domain experts does not answer, it goes further down the chain, until it in the end either gets an answer, or is not answered at all. Both Anderson et al. [2] and Treude et al. [70] defines an unanswered question to be a question where no accepted answer is chosen[6]. The second principle is that a questions activity level does not just indicate the interest for the question, but could also be an indicator for quality (because a question can have multiple answers).

[4] Yang et al. [75] characterised users as "Sparrows" and "Owls", where sparrows answers question for reputation and owls answers the difficult ones (domain experts).
Ahmed et al. [1, p. 2] defined users as "lurkers, help-seekers (askers) and givers (responders)".

[5] https://www.reddit.com/r/programming/comments/3cafkp/is_stack_overflow_overrun_by_trolls/.
https://medium.com/@johnslegers/the-decline-of-stack-overflow-7cb69faa575d
Last accessed 23.05.2016.

[6] However, they do not take into considerations users who find a solution on their own, or simply forget or neglect to mark a an answer as accepted.

Since users can only gain 200 reputation points daily, the only way to earn more is by having your answer marked as accepted or through bounties [63]. Movshovitz-Attias et al. [38] found that users earn more reputation by providing good answers rather than good questions[7]. Most questions was asked by the users with a low reputation, but on average users with high reputation asked more questions. This indicates that reputation could be used as a measurement for expertise. Ahmed et al. [1] also found that there was a correlation between amount of answers given and the users reputation.

Yang et al. [75] found that the activity level of a user is not equal to knowledge, and divided users into two groups; "Sparrows" and "Owls". The sparrows are the basic users who earns reputation and badges by answering the easy questions, and has a greater interest in the gamification element. They found that the sparrows usually has a low average score and targets questions that are easy, or non-relevant. Nonetheless, they are still important since they are able to provide quick feedback. As for the owls, they are considered to be the domain experts. The owls earn reputation by asking more advanced questions, providing better answers (i.e. getting their answer accepted) and answering popular and difficult[8] questions.

Posnett et al. [41] views SE and SO as a learning community, since users help each gain new knowledge, and motivates learning. They wanted to see if the quality of the users answers improved over time. By constructing a posting history for each user, they found that the overall answer score decreased, and that the answer quality was static.

Nasehi et al. [39] did a qualitative analysis of code examples posted on SO. Their focus was on questions related to Java programming, with the requirements that the question should at least have a score of +4 and the answer +7. In addition, a code example should be included (by checking for <code> in the post). They found that the code explanation was just as important as the code examples (but you are still restricted to the quality of that example). For the code to be considered good, they listed the following attributes:

1. Concise code: Code samples should not be too long. They should be simple, and only focus on the parts that are relevant to the topic. Additional or non-relevant parts should instead be documented by using descriptive comments.

[7] However, as stated in Movshovitz-Attias et al. [38, p. 3], the reputation system was changed at one point. Originally, up-votes on questions and answers gave users a +10, but this was later changed into up-votes on questions only giving +5.

[8] Popularity was measured based on page views and the time between a question was posted until an answer was selected as accepted. The popularity can also therefore be seen as a measurement for difficulty. The longer it takes to answer, the more difficult the question is [75, p. 273].

2. Question context: If the code is not working properly, suggestions for improvement should be added. One could also explain best practices and suggestions for improved readability. This will also have a pedagogical benefit, since the user asking the questions will learn to write better code.
3. Highlighting important elements: "Straight to the point", clearing up misunderstandings, pointing to relevant resources, etc.
4. Step-by-step solution: Splitting code into chunks, and explaining each chunk and its functionality. Comparison of languages; e.g. "How can I do X in C#, when I'm used to Java?"
5. Providing links to extra resources: Answers can be kept short by adding links to external resources, but a short summary should still be added.

2.2 Asking questions

2.2.1 What is the definition of a question?

The context of a question varies within the setting it is used. A question can be broad, where multiple answers can all be correct, or they can be factual, having only one right answer. When you are asking someone a question, you ask because you want to either find a solution to a problem, or learn something new. In the context of learning, questions are used for evaluating the students knowledge, or help them learn something new Nielsen et al. [40].

When doing research, you need research questions and hypotheses to decide what the goal of your research is. What questions are you trying to find an answer to, and what does that answer tell you? Slowiaczek et al. [55] defines asking a question as information selection and the answer(s) to a question as information usage. If you are working with statistical data, and you just post the numbers, this will not inform anyone. You need to explain what the numbers mean, and how you got them. The quality of an answer is also restricted to the quality of the question you ask. One can therefore assume that if you ask a good question, you increase the chance of getting a good answer [55].

2.2.2 Question Classification (QC)

QC is the process of categorizing a question into a class or category based on its structure, usually to decide what the expected answer type is [30, 31, 32]. To classify a question, it is important to select only those features that helps you identify the class it belongs to. To get a classification results, you use what is known as a classifier. The quality of a classifier can be measured by its accuracy and precision (see Equation 2.1 and 2.2; taken from [30, p. 13]).

$$\text{Accuracy} = \frac{\#\text{ of correct predictions}}{\#\text{ of predictions}} \tag{2.1}$$

$$\text{Precison}[c] = \frac{\#\text{ of correct predictions of class c}}{\#\text{ of prediction of class c}} \tag{2.2}$$

WH-words

WH-words are mostly found in factoid questions [32]. Huang et al. [22] listed eight different WH-words: What, which, when, where, who, how, why, and rest (rest being the type does not belong to any of the previous type). Letovsky [28] also listed "Whether" and "Discrepancy"[9]. However, not all are equally easy to use for classification, because even if the questions ask for the same answer, wording and syntactic structures can make it difficult to classify. Question containing words like "What", "Why", "How" and "Which", can be harder to classify due to the lack of limitation in regards to answer types[10] [22, 32].

Bag of Words (BOW) and N-grams

N-gram is a model that is used for splitting text into either characters (character model) or word frequencies (word model). The Bag of Words (BOW) model (or unigram) only looks at singular words, ignoring the order and relies only on the frequency for each word [35, 43]. Bi-grams takes dual values, tri-gram takes three, etc.

One problem with N-grams is that the dimension of the feature space is equal to the amount of words in the vocabulary [43, 31] When using categorization, there can be issues with mapping new words that does not exist in the vocabulary [76]. The impact of N-gram is also related to the size of the text being analysed. Zhang and Lee [77] found that there was not a big difference when using between bag-of-ngrams (all continuous word sequences in the question) and BOW as features.

Word mapping and processing: Case-sensitivity, Stemming, Stop words and Tokenization

To reduce the number of words used, there are more steps that can be taken. By removing the case-sensitivity, all words will be equal (e.g. is the word 'Hello' equal to the word 'hello'?). [22] includes case-sensitivity under a definition called word shape, consisting of five elements: upper case, all lower case, mixed case, all digits, and other.

Semantics can be used for word filtering, e.g. removal of duplicate words or words with same meaning. WordNet has a built in function called synsets() which removes synonyms (words having the same meaning). You can also look for hypernyms (words belonging to a category with a parent-child relationship) or use stemming. Stemming reduces the word to its base-form, e.g. crying would be converted into the word cry. Word separation is also possible through tokenization, which splits the text into an array based on a set delimiter. There is also usage of stop words for removal of frequently used words in a given lan-

[9] "Questions that reflect confusion over a perceived inconsistency." [28, p. 5]

[10] An answer type (or named entity) is the expected type of the answer to a given question (e.g. a Location, Organization, Person, Date, etc) [20, 32, 44, 76].

guage.

Grammatical properties can be extracted by using Part of Speech (POS), e.g. by using Natural Language Toolkit (NLTK)[11], which can be helpful in reducing ambiguities [6]. Li and Roth [30] uses the word head chunks to identify what the question is asking for when multiple types are introduced (avoid ambiguity). The same concept is used in [22] and [31], but there it is referred to as headwords.

2.2.3 Text classification

The goal of text classification (or text categorization) is to be able to process multiple documents or large amounts of text into categories. It shares similarities with question classification, although an obvious difference would be the size of the text that is processed. Some examples are spam filtering [43, 25], to identify languages, or filing documents based on content [25]. Documents can belong in more then category, and since categories can overlap they must be treated as a binary classification problem [24] . Text classification starts with retrieval of the documents, usually by using Information Retrieval (IR) methods, and then transforming the text into features for the classification. When you have a large amount of text, you can easily get a feature space that is very high dimensional, and that is why feature selection is important. Feature selection is the selection of features (or attributes) that are important for the classification. E.g. if you were classifying documents based on colour description, then hypernyms for colour would be an important feature.

2.2.4 Question-Answering (QA)

QA is mostly used as a method for finding the answer to a question from an unknown amount of documents. When using a search engine, one can accept that there are several results that are listed because at least one of the search terms exists. However, when using a QA system, users wants the answer straight away instead of having to read through several documents. In addition, QA sites allows users to search for questions in the same way they would ask another human (natural language[12]), and there are also different types of QA sites. Domain specific QA focuses on a specific topic (e.g. SO) and open domain where everything goes. QA sites can also function as an archive or a knowledge base, since all the posts are available even years after they were posted.

Yen et al. [76] found that it was more efficient searching for answers in a small dataset, then the document as a whole. By using a passage retriever, the documents were split into paragraphs and ranked by using evaluation metrics. Isozaki [23] could not use TF-IDF based paragraph retrieval, because the paragraphs

[11] NLTK includes in their POS tagger the following grammatical properties: Adjective, adposition, adverb, conjuction, determiner, article, noun, numeral, particle, pronoun, verb, punctuation mark and others [67, See Section 2.3].

[12] However, there is a problem with linguistics in natural language systems [32].

were too short to cover all query terms. If the terms that were used were too short or too long, the passage scores would not reflect the density distribution. Xu et al. [74] built an online QA system for tourism, which consisted of question analysis, information retrieval and answer extraction. Since rule-based approach requires expert knowledge, creating features that are domain specific can improve accuracy (what they called "domain term concept hierarchy"). To validate the classification, they tested the results by using 5-fold cross-validation.

Li and Roth [30] used semantics to categorize questions based on the possible semantic answer type. One issue with questions are that since they can be very short, they contain little text. However, the lack of long text improves both the accuracy and analysis.

Zhang and Lee [77] says that QC is important, and simply looking for WH-words is not enough. By using WH-words, headwords, WordNet semantics, N-grams and word shapes as features, and a linear SVM and Maximum Entropy model, they reached an 89.2% and 89.0% over a standard benchmark dataset. They also experimented with four other algorithms, Nearest Neighbours (simplified version of k-NN), Naive Bayes and Decision Tree and Sparse Network of Winnows (SNoW). However, these were outperformed by the SVM.

2.3 Support Vector Machines (SVM)

SVMs are good for solving regression and classification problems, and attempts to solve a linearly separable problem by using hyperplanes [11, 27, 43]. It is usually used for binary classifications, where classes are represented as either +1 or -1 [35, 43]. The hyperplane is the plane which separates the two classes:

$$\vec{w}^T\vec{x} = \text{-}b \tag{2.3}$$

where b is the intercept term and $\vec{w}$ is the weight vector (or decision hyperplane normal vector). The data set can be represented as $\mathbb{D} = \{(\vec{x}_i, y_i)\}$ where $\vec{x}_i$ is a data point, and y_i is its belonging class label. The linear classifier is represented in Equation 2.4 [35, p. 295-296].

$$f(\vec{x}) = sign(\vec{w}^T\vec{x} + b) \tag{2.4}$$

In addition to using a hyperplane, the SVM has an additional separation known as support vectors. Support vectors are the training points closest to the margin (the margin is the distance from the hyperplane) [11, 35]. Which then gives that the optimal hyperplane is the furthest from the support vectors [27]. SVM has four different kernels, but a limitation is that there is no way to select the best kernel function. Therefore, SVM often uses hyper-parameters and select the classifier based on the best results [69]. Equation 2.5 - 2.7 shows the kernel functions (K) for polynomial, radial basis function (RBF) and sigmoid (taken

from [27, p. 273]).
Polynomial - for a given polynomial degree d, the following is used:

$$K(\mathbf{x}_j, \mathbf{x}) = [(\mathbf{x} \cdot \mathbf{x}_j) + 1]^d \tag{2.5}$$

Radial - for a given γ value, the following is used:

$$K(\mathbf{x}_j, \mathbf{x}) = e^{-\gamma|\mathbf{x}-\mathbf{x}_j|^2} \tag{2.6}$$

Sigmoid - for a given sigmoid function S, we get a kernel function of parameters v and c:

$$K(\mathbf{x}_j, \mathbf{x}) = S(v(\mathbf{x} \cdot \mathbf{x}_j) + c) \tag{2.7}$$

For text classification, in most cases it will be non-linearly separable. One can therefore allow the SVM to do some mistakes. However, there is a cost for misclassifications, represented by what is called a slack variable ξ_i. If ξ_i is set (not zero), then the vector can miss the margin requirement at the cost of ξ_i. Equations are shown in Equation 2.8 - 2.10 (taken from [35, p. 301]).
Find

$$\vec{w}, b \text{ and } \xi_i \geq 0 \tag{2.8}$$

so that we can minimize the optimization problem

$$\frac{1}{2}\vec{w}^T\vec{w} \; + \; C\sum_i \xi_i \tag{2.9}$$

and for all

$$\{(x_i, y_i)\}, \; y_i(\vec{w}^T\vec{x} \; + \; b) \geq 1 - \xi_i \tag{2.10}$$

However, this gives a trade-off between the size of the margin and how much the data points can be adjusted. To avoid overfitting, the parameter C is used for regularization, where the size of C decides how much flexibility you get from the slack variable [35]. SVM neither suffer from the Curse of dimensionality, because the computational complexity is independent of the kernels dimensional space [69].

Joachims [24] found that SVM consistently achieved good performance during text categorization. Since SVM could generalize in high dimensional feature space, the need for feature selection was removed, and SVM was also robust. Since SVM does not require parameter tuning, they could find good parameter settings automatically. Loni et al. [31] found that SVM are successful on high dimensional data when it is sparse, but they still suffer from redundant features.

3 Methodology

3.1 Dataset and MySQL Database

3.1.1 Dataset

The dataset contains all information that is currently available in the SE 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 SE, 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 3).

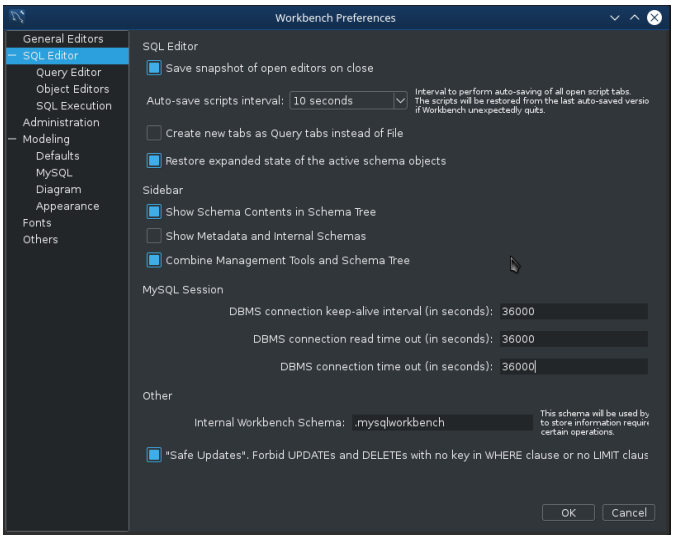

Figure 3: MySQL Workbench: Setting timeout values to avoid connection loss

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 were 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.2, p. 45. 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 [13]).

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 [51] [47, see 4.2.3]. When looking at this vocabulary, a lot of of unimportant words was found (a lot which came from the code samples) 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.

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[11]. Attempts were made to filter them out by using corpus.words.words() and corpus.wordnet.synset() from NLTK[12], and PyEnchant[13]. 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 setting it to 0.01, words that appeared in less 1% of all documents were ignored.

To be able to run the system without relying on an Integrated Development Environment (IDE), making it run from the Terminal using basic command setup seemed like a good idea. At first optparse was used, which ironically turned out to be deprecated and replaced by argparse. However, the problem was that you could only run one command at a time, whereas I wanted the program to be able to run until exited. The reason for this was because it needs to load a model before it can make a prediction, in addition the user might want to predict multiple questions. This was therefore replaced with a basic while loop that runs until the users enters the exit command. The setup used for argparse was kept, so users from *nix system might be more familiar with similar commands (shown in Listing 3.5). At the end, there were some commands that were not added to the menu, since these were mostly used for testing.

Listing 3.5: System menu

```
Menu:
d: Loads default model (if exists) from ./pickle_models
e: Exit the program
h: Displays this help menu
l: Load user created model. Arguments:
        path: Path to directory with model(s) (e.g. /home/user/my_models/)
        filename: The models filename
        suffix: File type — Optional (default: '.pkl')
```

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

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

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

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

p: Predict the quality of the entered question. Arguments:
    question: Question to predict quality of
t: Train a new model based on an existing (or new) data set. Arguments:
    path: Path to directory with training data (e.g. /home/user/my_data/)
    filename: Filename for data set (model name will be the same as this)
    db_load: Load from database (Enter 0: No, 1: Yes)
    limit: Limit for database row retrieval (integer) — Optional unless 'db_load' is '1'
u: Create an unprocessed data set based on database content (from database set in dbconfig.py).
Arguments:
    filename: Filename for data set (model name will be the same as this)
    limit: Limit for database row retrieval (integer)
    feature_detectors: Create singular feature detectors based on data set? (Enter 0: No, 1: Yes)
    create_model: Create classifier model(s) based on data set?
    0: No, 1: Unprocessed model, 2: Feature detector model(s) 3: Both (1 and 2)

3.3 Feature sets, attributes and processing

When retrieving the questions from the database, the vote score was set to less than -10 for bad question and greater than 50 for good questions (retrieval limit set to 10,000; 20,000 total). However, the vote score was set too low for the bad questions, since only 683 rows was returned. Therefore, the score was then set to less than -5. What was also found when using pandas.Categorical to get an overview (code snippet in Listing 3.7 and result in Table 1), one can see that for the 10,000 bad questions, the average vote score was -7. This could be an indicator that when a question has a vote score below -5, they are ignored.

Class	Statistics	AnswerCount	Score	Question length
-1	mean	2.0483	-7.0275	319.226
	std	1.3129	2.676	382.115
	min	0.0	-147.0	13.0
	25%	1.0	-7.0	153.0
	50%	2.0	-6.0	239.0
	75%	3.0	-6.0	379.0
	max	20.0	-6.0	13673.0
1	mean	11.9379	182.5483	459.329
	std	13.707824	317.47217	531.187559
	min	0.0	51.0	13.0
	25%	6.0	67.0	189.0
	50%	9.0	96.0	328.0
	75%	14.0	173.0	558.0
	max	518.0	9432.0	18867.0

Table 1: Results from pandas.DataFrame and pandas.Categorical. -1 is for bad questions (votes < -5), and 1 are for good questions (votes > 50).

Listing 3.6: Getting Categorical data from pandas.DataFrame

```
from pandas import DataFrame, Categorical

# get statistics from pandas.DataFrame
temp_df = __so_dataframe.loc[:, ("Score", "Body", "Title",
        "AnswerCount", "length")]
temp_df.loc[:, CLASS_LABEL_KEY] = Categorical(__so_dataframe.loc[:,
                                                "label"])

# prints out the questions AnswerCount, Score and length
print(temp_df.groupby("label").describe())
# prints all selected columns
print(temp_df.groupby("label").describe(include='all'))
```

To be able to develop some theories on what the difference between good and bad questions was, a total of 200 questions were reviewed (by sorting questions based on votes[14]). It was easier to see certain patterns in down-voted questions rather than those that were up-voted. A repetitive pattern was that many had either no code example, or poorly written code. These questions could also show

[14] http://stackoverflow.com/questions?sort=votes

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 2: Feature reduction steps before and after text was processed.

indications of not having tried anything, or that they were based on either homework or school assignments. This in turn lead to a hypothesis that if a question contains indicator of word synonyms for homework[15], it would be considered a bad question. In addition, some code examples had syntax errors, which made the minimum working example (MWE) not executable. Some questions also contained links, either to external resources or indicators of potential duplicates. Therefore links was also considered a potentially useful feature. Tags was also considered as a feature, which was divided into two: Attached and External tag. Attached tags are tags which the user has linked to the question, whereas external tags are all the tags available on SO. Version numbering was also considered, but this was not included due to the complexity of writing a proper filtering method to account for all possible variations.

Features were added in the same manner as was done for code samples, numerical and hexadecimal values. However, there were some issues when attempting to replace the tags and the synonyms for homework. At first, WordNet was used for synonyms (using wordnet.synset()). The only problem was that for the word 'homework', wordnet.synset() only returns ['homework', 'prep', 'preparation']. Whereas Thesaurus[16] had a lot more suggestions, and was therefore used instead. Words were selected based on whether or not it was plausible that they could be used in programming related question setting. A new problem now arose, namely the issue that the word "assignment" did not necessarily need to occur in a homework setting, since it could also be used as a programming word (e.g. assignment operator[17]). Therefore features for homework were split into two types: 'has_homework' and 'has_assignment'.

Tags were without a doubt one of the most annoying features to detect and replace. Site tags (or external) are single text values in the database, whereas the question can have up to five tags attached. Those attached tags are then

[15] http://www.thesaurus.com/browse/homework
[16] http://www.thesaurus.com/browse/homework
[17] http://stackoverflow.com/questions/5368258/the-copy-constructor-and-assignment-operator

separated in the following format: "<c><multi-threading>", which had to be processed by removing the '<' and the '>'. After the removal, each tag value was added to a list, so that all attached tags was indexed based on the question they belonged to. Furthermore, a combination of string replacement and regular expression was needed. The regular expression was used for single character tags (e.g. 'C'), and word replacement for longer words. The reason for this was that when using string replacement, single character tags replaced occurrences even if they appeared in the middle of a word. If the tags contained characters that could be interpreted as a regular expression (e.g. C++), it would give error about multiple repetitions. In addition, the tags needed to be sorted based on their length, since for questions that contained tags which included both <C> and <C++>, if <C> came first, it replaced the <C++> with 'has_*_tag'++. The text also had to be converted to lower-case to ensure proper tag matching.

Listing 3.7: Replacing tags in the question

```
for word in word_set:
        if len(word) == 1:
                # if its only one character (e.g. 'C'), ensure that it is a singular word by using regex
                text = re.sub(r"\b%s\b" % word, replacement_text, text, flags=re.IGNORECASE)
        else:
                text = text.replace(word, replacement_text)
```

3.4 Selecting estimator and parameters for classification

Two different classifiers were used, Support Vector Classification (SVC) and Stochastic Gradient Descent (SGD). The parameter values that were used for these two are shown in Listing 3.8 and 3.9. These are the default values used in Scikit-learns tutorials [46, 49] ([22, 34, 77] used default values for training their system). SVC was chosen because this is what I started with in the beginning, based on the tutorial written by [42]. SGD was used in some of Scikit-learns tutorials, and are mostly used for sample sizes greater than 100,000 (see Scikit-learns "roadmap" in Appendix A.6, Figure 7 p. 50). Since the sample size in this thesis was only 20,000, the expectation was that SVC would present better predictions..

Listing 3.8: Parameters for SVC

```
param_svm = [
{'clf__C': [1, 10, 100, 1000], 'clf__kernel': ['linear']},
{'clf__C': [1, 10, 100, 1000], 'clf__gamma': [0.001, 0.0001], 'clf__kernel': ['rbf']},
{'clf__C': [1, 10, 100, 1000], 'clf__gamma': [0.001, 0.0001], 'clf__kernel': ['sigmoid']},
]
```

Listing 3.9: Parameters for SGD

```
grid_parameters = {
'vect__min_df': (0.01, 0.025, 0.05, 0.075, 0.1),
'vect__max_df': (0.25, 0.5, 0.75, 0.95, 1.0),
'tfidf__use_idf': (True, False),
'tfidf__norm': ('l1', 'l2'),
'clf__alpha': (0.00001, 0.000001),
'clf__penalty': ('l1', 'l2', 'elasticnet'),
'clf__n_iter': (10, 50, 75, 100),
# 'clf__loss': ('hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'),
}
```

Instead of simply selecting random values for the classifier, exhaustive grid search was selected. The downside with using grid search is that it takes a lot of time to train, since all parameters are matched against each other to find the best combination [5, 37]. An example is presented in Equation 3.2 and 3.2, showing how many fits the exhaustive search must do before it is completed.

$$\begin{aligned} &\text{count(C)} + (\text{count(C)} \cdot \text{count(gamma)}) + (\text{count(C)} \cdot \text{count(gamma)}) \\ &\qquad \Longrightarrow \text{result} \cdot \text{cross-validation} = \text{fit\_amount} \\ &\qquad \Longrightarrow 4 + (4 \cdot 2) + (4 \cdot 2) = 20 \cdot 5 = 100 \end{aligned} \tag{3.1}$$

$$5 \cdot 5 \cdot 2 \cdot 2 \cdot 2 \cdot 3 \cdot 4 = 2400 \cdot 5 = 12,000 \tag{3.2}$$

From Scikit-learn, there are two options for exhaustive search: GridSearchCV and RandomizedSearchCV. The main difference is that GridSearchCV matches all parameters, and RandomizedSearchCV only uses a selection of the parameters. RandomizedSearchCV is faster than GridSearchCV, but as stated in [37], it may give lower scores. Markham [37] also suggests to start GridSearchCV and then compare the results against RandomizedSearchCV. Since I wanted to find the best parameters from the selection, GridSearchCV was used. However, if the goal was to fine-tune the parameters, RandomizedSearchCV could be more fitting. E.g. start with a high spread of values, and after each completed training, select a new set of values based on the best results. If the score was then too low, or if you wanted better results, you could use GridSearchCV to find the best parameters.

Before running the grid search, the questions were split into two parts, a training set and a test set by using train_test_split[18]. train_test_split splits the data randomly, but if the random_state has a fixed value, data will be split the same way (replicability). For the grid search, cross validation was also included, using StratifiedKFold with 5-folds. StratifiedKFold was selected because it is often

[18] http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html

used for classification tasks [27]. Cross-validation splits data into k-folds, where training is done on k-1 folds, and then evaluated against the last fold [5]. This means that only 80% of the 16,000 questions is used for training (12,800 questions for training, and 3,200 for evaluation). After the training was completed, the classifier models prediction accuracy was evaluated by using the test set.

4 Discussions

Working on re-writing 4.2 and re-writing 4.4

4.1 Data set and Question selection

The dataset was retrieved from Stack Exchange Archive [58], and contains all the data posted since the beginning in 2008. In this thesis, only the data for SO was used, specifically the posted questions. A simplified overview is shown in Table 3, listing questions based on vote score.

	Amount	Oldest	Newest	Vote (lowest)	Vote (highest)
Votes < 0	659,955	06.08.2008	06.03.2016	-147	-1
Votes $= 0$	5,256,105	06.08.2008	06.03.2016	0	0
Votes > 0	5,286,971	31.07.2008	06.03.2016	1	13845
All questions	11,203,031	31.07.2008	06.03.2016	-147	13845

Table 3: Overview of the questions in the Stack Overflow dataset.

As previously mentioned in Section 3.3, the value used to extract the questions was set to -5 and +50. The problem with these values are that they were simply selected. Originally, the value for bad questions was -10, which were then changed to -5, because -10 only retrieved 683 rows (and I wanted 10,000 samples for both question types). However, as can be seen from Table 3, +50 is most likely a too low value for the good questions. To get more representative results, three different alternatives could be considered.

The first alternative is the easiest. If you are only interested in a set sample size, then you could simply retrieve the amount sorted by score. If you want 10,000 of each, you would then get the 10,000 that are scored highest (or lowest). The second alternative would be to get a set limit based on the actual score, by using the mean or average. For all the questions of a given type, retrieve the average and then select questions which has a score higher (or lower) than the average. The third alternative would be to use quartiles. Quartiles is measured by a given percentage of the observations, where each quartile represents 25%; Q_1 (25%), M (median, 50%) and Q_3 (75%) [18]. The equation to calculate the quartiles are shown in Equation 4.1 - 4.3.

$$Q_1 = \frac{(n+1)}{4} \tag{4.1}$$

$$M = \frac{(n+1)}{2} \tag{4.2}$$

$$Q_3 = \frac{3 \cdot (n+1)}{4} \tag{4.3}$$

As previously mentioned in Section 3.3, there was no clear indicator for the good questions. Some consisted only of two sentences, e.g. "I committed the wrong files to Git. How can I undo this commit?"[1]. Common for many of the top-voted questions is that they are short, but that is not the case for all of them[2]. There is also what I would call a bias factor. If enough people have the same problem, then it will automatically become a good question. Not because of the questions quality, but simply because many will encounter it (e.g. Bugs, IDE behaviour, tweaks, etc). The same can be said for bad questions. In many cases, a question is bad not because of the question that is asked, but simply because it does not follow the guidelines. If a question is a duplicate, it is automatically voted for closure, and will in most cases receive multiple down-votes. Questions which gives a hint of being school related will also receive down-votes, mainly because SO targets professionals and experts. SO is neither a fan of unnecessary text like greetings and gratefulness [8, 21].

Is SO fit for measuring question quality? In a closed domain setting, the answer is yes. However, as mentioned, what type of quality being measured must be taken into consideration. As it is, the system would not be useful for an educational setting, because it bases its prediction on questions asked on SO. If a student were to ask the system a question, it might respond saying that this is a bad question (but the result may be based on the fact that it is a duplicate). For an educational setting, a better solution would be to develop a system similar to the one in [29]. With the current state of the system (if it were to be used), it would be more appropriate to use it as a measurement tool for new SO questions, rather then general question quality.

Considering the amount of data this data set provides, there is a lot of research that could be used in relation to AI and ML development. One example is Schutte [45], who used the same data set to build an auto-complete for Javascript. When taking Big Data into consideration, a question has also been presented on whether or not it is the data, and not the algorithm which improves AI [26, 56, 73].

[1] This question currently have a score of 10,406:
http://stackoverflow.com/questions/927358/how-do-you-undo-the-last-commit.

[2] One example is this question, with a score of 3,009:
http://stackoverflow.com/questions/826782/css-rule-to-disable-text-selection-highlighting.

4.2 Feature and classifier results

A total of six features were detected and converted from each question. These features were code samples[3], hexadecimal, numerical, synonyms for homework and tags. As mentioned in Section 3.3, homework and tags contained two feature types. One problem with the external tags (all tags listed on the given site), was that it replaced even normal words[4] (e.g. "this", "can", "let", etc). Since the external tags and the word "assignment" was conflictive, these were only included when creating classifier models for the singular feature detectors[5].

To be able to compare if a feature would have an impact, the creation of classifier models was divided into three parts. In the first part, the raw, unprocessed data set was used to create a classifier model by using GridSearchCV. The only modification done to the data was removing the HTML and converting the text to lower-case. In the second part, the best parameter values from the unprocessed classifier model was used to create new models for the singular feature detectors. This ensures that a correct comparison can be made, since the results for both models are based on the same values. If they were all trained separately with GridSearchCV, there is a risk that it would select other parameters for the model. The last part consisted of using all the features, both with the values from the unprocessed classifier model, and using GridSearchCV to select the best parameters. Porter stemming was also included, but this was only used for the last classifier model. The confusion matrices for each classifier model is presented in Appendix A.4, 47.

The results for the first and second part of the training is shown in Table 4. In this table, all classifiers have been trained using the same parameter settings, based on parameters found from the grid search on the unprocessed data set. Without doing anything to the questions, the accuracy is already at 79.9%. The two features that sticks out the most is Numerical and Tags. Numericals accuracy score is 80.55%, which is better than the unprocessed, and Tags accuracy score is almost 4% lower than the unprocessed. However, the results for both Homework and Tags are questionable, because they contain features (external and homework) not represented in the classifier for all the features. Furthermore, the total number of questions containing Homework is only 374 (including the assign-

[3] Nothing was done to the content of the code sample, it was only removed and replaced with 'has_codeblock'.

[4] For a visualization of how bad it was, see Figure 5 and 6 in Appendix A.5, p. 49. See also the files "*training_data_10000_unprocessed_UP_has_tags.csv*" and "*training_data_10000_unprocessed_has_tags.csv*", found in *./IMT4904_MasterThesis_Code/extraction_sets/*.

[5] The singular feature detectors are based on the premise that each feature should be tested individually, and none of the other feature detectors should be used in the model creation process.

ment feature), and Hexadecimal is only present in 160 questions (see Table 5 and 6). This means that if Hexadecimal was not a feature, it would not even be included (because the min_df is set to exclude words appearing in less than 1% of the questions). Therefore, Hexadecimal is not a good feature which is shown by the accuracy score in Table 4 and 5 (this is also shown in the confusion matrix in the Appendix). To get a more representative score for Homework and Tags, an additional classifier model should have been made using only the homework features and the attached tags. Overall, the only feature that had an increase in its prediction accuracy was the classifier for Numerical (only 375 misclassified as bad).

	Accuracy Score
Code block	79.17%
Hexadecimal	79.90%
Homework	79.90%
Links	79.35%
Numerical	80.55%
Tags	76.17%
All features	79.07%
Unprocessed	79.90%

Table 4: Classifier results based on the parameters found for the raw (unprocessed) questions. Classifier=SVC, with Kernel=RBF, C=1000 and Gamma=(γ) 0.0001

It would also be interesting to compare the unprocessed against each feature detector for the questions containing it. Following the same basis as the previous part, new classifier models were made, but this a new grid search was executed for each feature. This way, the classifier would be adjusted to the number of questions for each feature. The results from this training is displayed in Table 5, and the confusion matrix is presented in Appendix A.4.2, p. 48). One thing that needs to be taken into consideration is the total sample size that was used for training and testing. When all questions were used, a total of 16,000 questions was used for training and 4,000 used for evaluation, which is not the same used here. The number of questions that contained each feature is shown in Table 6, and the number used for training and evaluation (test set) is shown in Table 7.

One thing that is clear is how much effect the external tags has. Tags appear in 19,967 questions, whereas the detector for all features only appears in 17,558. This means that at least 2,409 contains external tags, and that if they were to be included, a filtering mechanism would be needed (e.g. ignoring everything that could be considered a word). The feature for code blocks now have a higher accuracy than the unprocessed, and is better at classifying bad questions. This could indicate that questions including a lot of code samples is not a good

question[6]. Numerical now has a worse accuracy than the unprocessed. However, there are two things worth mentioning. First, the classifier for Numerical is only trained on 9,024 questions, instead of 20,000. Second, as can be seen in Table 7, only 30% of the training data are good questions. When also looking at the confusion matrix (Appendix A.4.2, p. 48), it predicts more questions as bad.

	Unprocessed	Feature	C	Gamma (γ)	Kernel
Code block	Accuracy: 78.84%	Accuracy: 79.04%	1	N/A	Linear
Hexadecimal	Accuracy: 81.25%	Accuracy: 81.25%	1	N/A	Linear
Homework	Accuracy: 84.00%	Accuracy: 82.67%	1	N/A	Linear
Links	Accuracy: 83.72%	Accuracy: 81.78%	1	N/A	Linear
Numerical	Accuracy: 80.22%	Accuracy: 79.66%	1000	0.0001	RBF
Tags	Accuracy: 79.36%	Accuracy: 76.46%	1000	0.0001	RBF
All features	Accuracy: 79.24%	Accuracy: 79.15%	1000	0.0001	RBF

Table 5: Comparison of raw data set (unprocessed) and singular features, for questions containing the given feature. Classifier: SVC.

	Bad: -1	Good: 1	Total
Code block	5,090	4,765	9,855
Hexadecimal	109	51	160
Homework	261	113	374
Links	778	1,798	2,575
Numerical	5,804	3,220	9,024
Tags	9,987	9,980	19,967
All features	8,466	9,092	17,558

Table 6: The number of questions containing the given feature.

Table 8 shows a comparison between the classifier for unprocessed and all features. This also includes a comparison of stemmed vs non-stemmed features. There were two stemmers that were considered, Porter and Lancaster. Lancaster is based on Porter, but is more aggressive, and what I experienced was that words which was not even of the same root (e.g. 'user' and 'using' both became 'us') [68]. The expectation was that stemming would enhance the prediction and increase the prediction accuracy, but this was not the case. When compared with the unprocessed, the accuracy score was 3.93% lower, and 3.15% lower than the data set with all feature detectors without stemming.

[6] There was no analysis done on the code samples, but during the development, I noticed that some questions also used the <code> tag on words like "Integer" and "Double".

	Training: -1	Training: 1	Evaluation: -1	Evaluation: 1
Code block	4102	3782	988	983
Hexadecimal	88	40	21	11
Homework	205	94	56	19
Links	627	1433	151	365
Numerical	4650	2569	1154	651
Tags	8002	7971	1985	2009
All features	6795	7251	1671	1841

Table 7: The number of questions used for evaluation. Bad questions: -1, Good questions: 1.

	Unprocessed	All features	All features (no stemming)
Score	79.90%	75.97%	79.15%
Gamma (γ)	$1e^{-04}$ (0.0001)	$1e^{-03}$ (0.001)	$1e^{-03}$ (0.001)

Table 8: Comparison of the classifier for the raw (unprocessed) questions vs. questions with all features. Classifier: SVC, Kernel=RBF and C=1000.

Table 9 shows a comparison between unprocessed and all features (stemmed) using the SGD classifier. A full exhaustive search was also ran on the unprocessed, but this is omitted because it resulted in the same values as for loss='log'. The reason loss was set to 'log' is because you can then get a probability score when using prediction. Probability was desired, because when a user then enters a new question to predict, the output would also contain the probability and not just "This is a good/bad question".

continue re-writing from here

As expected, when comparing Table 8 with Table 9, SVC did get a slightly higher score. The difference is barely notable, since the classifier for the unprocessed SVC achieved a score of 79.90% vs. SGD which achieved 79.87% (only 0.3% difference). Whereas for the classifier using all features, SVC achieved a score 75.97% and SGD achieved 75.55% (0.42% difference). This could indicate that if one were to increase the sample size to 30,000 or higher, SGD could be a better classifier.

One thing that is interesting is the comparison of the stemmed and non-stemmed classifiers for the SVC. The stemmed classifier has the worst prediction, but the non-stemmed classifier using its own parameters have better prediction for bad questions. The SGD is better at predicting bad questions, but performs worse for good. When comparing the unprocessed SGD classifier against the one using all features, it is much better at predicting good questions.

move or delete this paragraph

In Appendix A.4, 47, the confusion matrices for all these are listed. The confu-

	Unprocessed (loss='log')	All features (loss='log')
Score	79.87%	75.55%
Min DF	0.01	0.01
Max DF	0.5	0.75
Use IDF	False	True
Alpha	$1e^{-05}$ (0.00001)	$1e^{-05}$ (0.00001)
Normalization	l2	l2
Penalty	elasticnet	l2
Iteration	50	100
Loss	log	log

Table 9: Comparison of the classifier for the raw (unprocessed) questions vs. questions with all features. Classifier: SGD.

sion matrix is based on the test set that was created using train_test_split. These show a clear indication that the system is much better at predicting bad questions, then good. This also explains why it was easier to spot the bad questions, rather than a good one when attempting to find features.

this is not intended to be in the final version, just commentary before rewrite

To see if the system was indeed expandable for communities in SE, Tex.StackExchange was also tested to see which would be more predictive (tables are shown in Appendix A.3, p. 46). The results clearly shows that SO is more predictive, given the amount of data available when compared to Tex.StackExchange[7]

[7] However, it should be noted that this dataset was downloaded in August 2015, but the latest post in that database was from 2014.

4.3 Artificial Intelligence (AI) Methods

main goal is to write about alternative methods and potential limitations with using svm, etc.

4.4 Limitations and other issues

This whole section will be re-written

One of the major issues was the fact that the latest development version was used instead of the stable one. The question became at one point whether or not a switch should be made from using the development version into the stable version. There were two things that needed to be taken into consideration. First of all, it was unknown when this development version would become the next stable one. If the development version became the new stable version, a lot of alterations would have to be made to the source code. Furthermore, for the long term, if this system would become successful, it would be easier to maintain in the future if it relied on the latest version.

When you are only one person with only one computer there is a certain limitation to how much work can be done simultaneously. A lot of time were spent having to rebuild the database and processing the data (e.g. finding features, replacing it, training classifiers). To ensure expandability (for the SE community) and ensuring replicability (having a unprocessed data set), more than once code had to be updated or re-written. This easily caused a lot of unforeseen issues. The worst one was the realization that after the update of the unprocessed data set (changing it to contain HTML like in the database), the HTML was not removed from the text. The impact was that all the models that were created using the parameters and data from the unprocessed had to be re-trained to get correct results.

One of the more peculiar issues (which I do not have an explanation for), was when the real training started (using the SVC and GridSearchCV). The reason that this issue cannot be explained, is because I do not know what caused it to happen. What happened was that when the training was started, no verbose was printed at all (which was weird considering I had used the same values before and it then gave a verbose output). The program ran for hours, without giving any feedback, errors or output.

It took almost three days to find a solution, where part of the solution was switching to Windows (which had its own issues, since x64 is not supported by Numpy). The main difference was that in Windows, at least verbose was printed, although it only printed verbose once[8]. There were two things that were changed, which finally made it both print verbose and complete the training. The first part was changing the n_jobs value to something else than -1. By setting n_jobs=-1, GridSearchCV will run all jobs in parallel, using all logical cores (e.g. on a CPU with 4 physical cores, it will use all 8 logical cores). However, multi-threading is not supported in Windows [17]. To ensure that progress was made, and that the

[8] Verbose was printed once, sometime twice between the first 20 - 60 minutes, then nothing. The longest run time that was registered was around 12 hours without any verbose printed.

program had not frozen again, the verbose level was increased. By increasing the verbose level, you can force the algorithm to print more information about progress, but with an increase in the time it takes to finish [36, 71]. The most fascinating part was that after increasing the verbose level, it not only printed out continuously, but it even finished training in less than 3 hours! This have not been tested in Linux, but the reason nothing happened may have been the same which happened for Windows, that it could not utilize all CPU cores[9].

either add this to limitation/issues or delete it

The most useful resource were of course Scikit-learns tutorials [50][10] and documentation [47, 52], but there were also two other tutorials that proved helpful ([42] and [12]). The main problem was that Scikit-learns documentation and most of the tutorials written by others is based on v0.17.1. Due to various problems with installation in regards to Numpy and Scipy, Scikit-learn had to be installed from GitHub (which is v0.18.dev0). In the development version, a lot of functions and modules had been re-written and moved, making many tutorials and examples outdated.

[9] My assumption is that when all the logical cores are used, there is no processing power left for the Operative System (OS). This in turn would then cause an infinite deadlock, since by using all the cores, there is no processing power left for the OS. There is also a known issue with parallelization in Linux, see: https://pythonhosted.org/joblib/parallel.html#bad-interaction-of-multiprocessing-and-third-party-libraries.

[10] The examples that were worked through can be seen here (as stated in ReadMe, the only thing I have altered is comments and adjustments for v0.18.dev0): https://github.com/klAndersen/scikit-learn_tutorials.

5 Conclusion/Summary

5.1 Overview of main results

- short about SO today
- short about the work that was done
- short about the results and what they indicate

5.2 Further work

- syntax errors/Lint check, e.g. http://esprima.org/demo/validate.html
- SO as an IDE - https://emilschutte.com/stackoverflow-autocomplete/
- potential for data set - sentiment analysis
- symbol usage, e.g. '?', '!', emoticons, etc
- use of greetings, thanks, etc.
- version numbering
- fine tuning of parameters (e.g. RandomizedSearchCV)
- using N-grams

Bibliography

[1] Saif Ahmed, Seungwon Yang, and Aditya Johri. “Does Online Q&A Activity Vary Based on Topic: A Comparison of Technical and Non-technical Stack Exchange Forums”. In: *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*. L@S ’15. Vancouver, BC, Canada: ACM, 2015, pp. 393–398. ISBN: 978-1-4503-3411-2. DOI: 10.1145/2724660.2728701. URL: http://doi.acm.org/10.1145/2724660.2728701.

[2] Ashton Anderson et al. “Discovering Value from Community Activity on Focused Question Answering Sites: A Case Study of Stack Overflow”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’12. Beijing, China: ACM, 2012, pp. 850–858. ISBN: 978-1-4503-1462-6. DOI: 10.1145/2339530.2339665. URL: http://doi.acm.org/10.1145/2339530.2339665.

[3] Jeff Atwood. *A Day in the Penalty Box*. 2009. URL: http://blog.stackoverflow.com/2009/04/a-day-in-the-penalty-box/ (visited on 04/05/2016).

[4] Jeff Atwood. *What is Trolling?* 2015. URL: https://blog.codinghorror.com/what-is-trolling/ (visited on 05/23/2016).

[5] Christopher M Bishop. *Pattern Recognition and machine learning*. Springer, 2006. ISBN: 978-0-387-31073-2.

[6] Stephan Bloehdorn and Andreas Hotho. “Boosting for text classification with semantic features”. In: *WebKDD*. Springer. 2004, pp. 149–166.

[7] Codementor.io. *How to Set up NumPy on a 64 bit Windows OS*. 2015. URL: https://www.codementor.io/numpy/tutorial/installing-numpy-64-bit-windows.

[8] CommunityWiki. *Should ’Hi’, ’thanks’, taglines, and salutations be removed from posts?* 2016. URL: http://meta.stackexchange.com/questions/2950/should-hi-thanks-taglines-and-salutations-be-removed-from-posts (visited on 05/07/2016).

[9] CommunityWiki. *What is a "closed" or "on hold" question?* 2016. URL: http://meta.stackexchange.com/questions/10582/what-is-a-closed-or-on-hold-question (visited on 04/05/2016).

[10] Sebastian Deterding et al. “From game design elements to gamefulness: defining gamification”. In: *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*. ACM. 2011, pp. 9–15.

[11] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley-Interscience, 2001. ISBN: 978-0-471-05669-0.

[12] Irfan Elahi. *Advanced Document Classification using Python (Sklearn, NLTK)*. 2016. URL: http://www.irfanelahi.com/data-science-document-classification-python/ (visited on 05/07/2016).

[13] Eloff. *Strip HTML from strings in Python*. 2009. URL: http://stackoverflow.com/a/925630 (visited on 03/22/2016).

[14] Howard Fosdick. *Why People Troll and How to Stop Them*. 2012. URL: http://www.osnews.com/story/25540 (visited on 05/23/2016).

[15] Jan-Philip Gehrcke. *How to set up a 64 bit version of NumPy on Windows*. 2015. URL: https://gehrcke.de/2015/02/how-to-set-up-a-64-bit-version-of-numpy-on-windows/.

[16] Pascal-Emmanuel Gobry. *Google Is Indexing Stack Overflow At 10 Times Per Second*. 2011. URL: http://www.businessinsider.com/google-stackoverflow-2011-3?r=US&IR=T&IR=T (visited on 05/23/2016).

[17] Isaac GS. *GridSearchCV no reporting on high verbosity*. 2015. URL: http://stackoverflow.com/questions/29995249/verbose-argument-in-scikit-learn (visited on 05/27/2016).

[18] Per Chr. Hagen. *Innføring i sannsynlighetsregning og statistikk*. Cappelen Damm, 2011. ISBN: 978-82-02-31567-2.

[19] Benjamin V. Hanrahan, Gregorio Convertino, and Les Nelson. “Modeling Problem Difficulty and Expertise in Stackoverflow”. In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion*. CSCW ’12. Seattle, Washington, USA: ACM, 2012, pp. 91–94. ISBN: 978-1-4503-1051-2. DOI: 10.1145/2141512.2141550. URL: http://doi.acm.org/10.1145/2141512.2141550.

[20] Matthias H Heie, Edward WD Whittaker, and Sadaoki Furui. “Question answering using statistical language modelling”. In: *Computer Speech & Language* 26.3 (2012), pp. 193–209.

[21] Josh Heyer. *Stack Exchange is not a forum: the role of "niceness" on a Q&A site*. 2012. URL: https://blog.stackoverflow.com/2012/08/stack-exchange-is-not-a-forum-the-role-of-niceness-on-a-qa-site/ (visited on 05/07/2016).

[22] Zhiheng Huang, Marcus Thint, and Zengchang Qin. “Question Classification Using Head Words and Their Hypernyms”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. EMNLP ’08. Honolulu, Hawaii: Association for Computational Linguistics, 2008, pp. 927–936. URL: http://dl.acm.org/citation.cfm?id=1613715.1613835.

[23] Hideki Isozaki. “An Analysis of a High-performance Japanese Question Answering System”. In: 4.3 (Sept. 2005), pp. 263–279. ISSN: 1530-0226. DOI: 10.1145/1111667.1111670. URL: http://doi.acm.org/10.1145/1111667.1111670.

[24] Thorsten Joachims. “Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings”. In: ed. by Claire Nédellec and Céline Rouveirol. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. Chap. Text categorization with Support Vector Machines: Learning with many relevant features, pp. 137–142. ISBN: 978-3-540-69781-7. DOI: 10.1007/BFb0026683. URL: http://dx.doi.org/10.1007/BFb0026683.

[25] Celso Antonio Alves Kaestner. “Support Vector Machines and Kernel Functions for Text Processing”. In: *Revista de Informática Teórica e Aplicada* 20.3 (2013), pp. 130–154.

[26] Gary Klein. *Blinded By Data*. 2016. URL: https://www.edge.org/response-detail/26692 (visited on 04/24/2016).

[27] Igor Kononenko and Matjaž Kukar. *Machine learning and data mining: introduction to principles and algorithms*. Woodhead publishing, 2007. ISBN: 978-1-904275-21-3.

[28] Stanley Letovsky. “Cognitive processes in program comprehension”. In: *Journal of Systems and Software* 7.4 (1987), pp. 325–339. ISSN: 0164-1212. DOI: http://dx.doi.org/10.1016/0164-1212(87)90032-X. URL: http://www.sciencedirect.com/science/article/pii/016412128790032X.

[29] C Galina E. Lezina and Artem M. Kuznetsov. *Predict Closed Questions on StackOverflow*. 2013. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.394.5678.

[30] Xin Li and Dan Roth. “Learning Question Classifiers: The Role of Semantic Information”. In: *Natural Language Engineering* 1.1 (), pp. 000–000.

[31] B. Loni, S. H. Khoshnevis, and P. Wiggers. "Latent semantic analysis for question classification with neural networks". In: *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. Dec. 2011, pp. 437–442. DOI: 10.1109/ASRU.2011.6163971.

[32] Vanessa Lopez et al. "Is Question Answering Fit for the Semantic Web?: A Survey". In: *Semant. web* 2.2 (Apr. 2011), pp. 125–155. ISSN: 1570-0844. DOI: 10.3233/SW-2011-0041. URL: http://dx.doi.org/10.3233/SW-2011-0041.

[33] Jitendra Maan. "Social business transformation through gamification". In: *arXiv preprint arXiv:1309.7063* (2013).

[34] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 142–150. ISBN: 978-1-932432-87-9. URL: http://dl.acm.org/citation.cfm?id=2002472.2002491.

[35] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*. Vol. 1. Cambridge University Press, 2008. ISBN: 978-0521865715.

[36] Juan Manuel. *'verbose' argument in scikit-learn*. 2015. URL: http://stackoverflow.com/questions/29995249/verbose-argument-in-scikit-learn (visited on 05/27/2016).

[37] Kevin Markham. *Efficiently searching for optimal tuning parameters*. 2015. URL: http://blog.kaggle.com/2015/07/16/scikit-learn-video-8-efficiently-searching-for-optimal-tuning-parameters/ (visited on 05/27/2016).

[38] Dana Movshovitz-Attias et al. "Analysis of the reputation system and user contributions on a question answering website: Stackoverflow". In: *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*. IEEE. 2013, pp. 886–893.

[39] S. M. Nasehi et al. "What makes a good code example?: A study of programming Q amp;A in StackOverflow". In: *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. Sept. 2012, pp. 25–34. DOI: 10.1109/ICSM.2012.6405249.

[40] Rodney D Nielsen et al. "A taxonomy of questions for question generation". In: *Proceedings of the Workshop on the Question Generation Shared Task and Evaluation Challenge*. 2008.

[41] D. Posnett et al. “Mining Stack Exchange: Expertise Is Evident from Initial Contributions”. In: *Social Informatics (SocialInformatics), 2012 International Conference on*. Dec. 2012, pp. 199–204. DOI: 10.1109/SocialInformatics.2012.67.

[42] Radim Rehurek. *Practical Data Science in Python*. 2014. URL: http://radimrehurek.com/data_science_python/ (visited on 05/07/2016).

[43] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson Education, 2013. ISBN: 978-1292024202.

[44] Yutaka Sasaki. “Question answering as question-biased term extraction: a new approach toward multilingual QA”. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2005, pp. 215–222.

[45] Emil Schutte. *Autocomplete from Stack Overflow*. 2016. URL: https://emilschutte.com/stackoverflow-autocomplete/ (visited on 05/10/2016).

[46] Scikit-learn. *3.2. Grid Search: Searching for estimator parameters*. 2016. URL: http://scikit-learn.org/stable/modules/grid_search.html (visited on 05/07/2016).

[47] Scikit-learn. *4.2. Feature extraction*. 2016. URL: http://scikit-learn.org/stable/modules/feature_extraction.html (visited on 05/07/2016).

[48] Scikit-learn. *Choosing the right estimator*. 2016. URL: http://scikit-learn.org/dev/tutorial/machine_learning_map/index.html (visited on 04/05/2016).

[49] Scikit-learn. *Sample pipeline for text feature extraction and evaluation*. 2016. URL: http://scikit-learn.org/stable/auto_examples/model_selection/grid_search_text_feature_extraction.html (visited on 05/07/2016).

[50] Scikit-learn. *scikit-learn Tutorials*. 2016. URL: http://scikit-learn.org/stable/tutorial/index.html (visited on 05/07/2016).

[51] Scikit-learn. *sklearn.feature_extraction.text.CountVectorizer*. 2016. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (visited on 05/07/2016).

[52] Scikit-learn. *Working With Text Data*. 2016. URL: http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html (visited on 05/07/2016).

[53] Madhavi Sewak et al. *Finding a Growth Business Model at Stack Overflow, Inc.* 2010. URL: https://web.stanford.edu/class/ee204/Publications/Finding%20a%20Growth%20Business%20Model%20at%20Stack%20Overflow.pdf (visited on 05/07/2016).

[54] Logan Short, Christopher Wong, and David Zeng. "Tag recommendations in stackoverflow". In: (2014).

[55] Louisa M. Slowiaczek et al. "Information selection and use in hypothesis testing: What is a good question, and what is a good answer?" In: *Memory & Cognition* 20.4 (1992), pp. 392–405. ISSN: 1532-5946. DOI: 10.3758/BF03210923. URL: http://dx.doi.org/10.3758/BF03210923.

[56] SpaceMachine.net. *DATASETS OVER ALGORITHMS*. 2016. URL: http://www.spacemachine.net/views/2016/3/datasets-over-algorithms (visited on 04/24/2016).

[57] Joel Spolsky. *Stack Overflow Launches*. 2008. URL: http://www.joelonsoftware.com/items/2008/09/15.html (visited on 05/06/2016).

[58] Inc. StackExchange. *Stack Exchange Data Dump*. 2016. URL: https://archive.org/details/stackexchange (visited on 04/05/2016).

[59] StackOverflow.com. *Badges*. 2016. URL: http://stackoverflow.com/help/badges (visited on 04/05/2016).

[60] StackOverflow.com. *How to Ask*. 2016. URL: http://stackoverflow.com/questions/ask/advice? (visited on 04/05/2016).

[61] StackOverflow.com. *Privileges*. 2016. URL: http://stackoverflow.com/help/privileges (visited on 04/05/2016).

[62] StackOverflow.com. *What does it mean if a question is "closed" or "on hold"?* 2016. URL: http://stackoverflow.com/help/closed-questions (visited on 04/05/2016).

[63] StackOverflow.com. *What is reputation? How do I earn (and lose) it?* 2016. URL: http://stackoverflow.com/help/whats-reputation (visited on 04/05/2016).

[64] StackOverflow.com. *What topics can I ask about here?* 2016. URL: http://stackoverflow.com/help/on-topic (visited on 04/05/2016).

[65] StackOverflow.com. *What types of questions should I avoid asking?* 2016. URL: http://stackoverflow.com/help/dont-ask (visited on 04/05/2016).

[66] Clayton Stanley and Michael D Byrne. "Predicting tags for stackoverflow posts". In: *Proceedings of ICCM*. Vol. 2013. 2013.

[67] Ewan Klein Steven Bird and Edward Loper. *Categorizing and Tagging Words*. 2015. URL: http://www.nltk.org/book/ch05.html (visited on 05/06/2016).

[68] Textprocessing.com. *Stemming and Lemmatization with Python NLTK*. 2016. URL: http://text-processing.com/demo/stem/ (visited on 05/07/2016).

[69] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press (Elsevier), 2009. ISBN: 978-1-59749-272-0.

[70] C. Treude, O. Barzilay, and M. Storey. "How do programmers ask and answer questions on the web? (NIER track)". In: *Software Engineering (ICSE), 2011 33rd International Conference on*. May 2011, pp. 804–807. DOI: 10.1145/1985793.1985907.

[71] user2991243. *What is "Verbose" in scikit-learn package of Python?* 2014. URL: http://stats.stackexchange.com/questions/153823/what-is-verbose-in-scikit-learn-package-of-python (visited on 05/27/2016).

[72] Bogdan Vasilescu. *Academic papers using Stack Exchange data*. 2012. URL: http://meta.stackexchange.com/questions/134495/academic-papers-using-stack-exchange-data.

[73] Alexander Wissner-Gross. *Datasets Over Algorithms*. 2016. URL: https://www.edge.org/response-detail/26587 (visited on 04/24/2016).

[74] Jinzhong Xu, Yanan Zhou, and Yuan Wang. "A Classification of Questions Using SVM and Semantic Similarity Analysis". In: *Internet Computing for Science and Engineering (ICICSE), 2012 Sixth International Conference on*. Apr. 2012, pp. 31–34. DOI: 10.1109/ICICSE.2012.49.

[75] Jie Yang et al. "User Modeling, Adaptation, and Personalization: 22nd International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings". In: ed. by Vania Dimitrova et al. Cham: Springer International Publishing, 2014. Chap. Sparrows and Owls: Characterisation of Expert Behaviour in StackOverflow, pp. 266–277. ISBN: 978-3-319-08786-3. DOI: 10.1007/978-3-319-08786-3_23. URL: http://dx.doi.org/10.1007/978-3-319-08786-3_23.

[76] Show-Jane Yen et al. "A support vector machine-based context-ranking model for question answering". In: *Information Sciences* 224 (2013), pp. 77–87. ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/j.ins.2012.10.014. URL: http://www.sciencedirect.com/science/article/pii/S0020025512006792.

[77] Dell Zhang and Wee Sun Lee. “Question Classification Using Support Vector Machines”. In: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*. SIGIR ’03. Toronto, Canada: ACM, 2003, pp. 26–32. ISBN: 1-58113-646-3. DOI: 10.1145/860435.860443. URL: http://doi.acm.org/10.1145/860435.860443.

A Appendix

A.1 Acronyms

AI Artificial Intelligence. 3, 26

BOW Bag of Words. 10

IDE Integrated Development Environment. 18, 26, 35

IR Information Retrieval. 11

ML Machine Learning. iii, 2, 3, 26

NLTK Natural Language Toolkit. 11, 18, 52

POS Part of Speech. 11

QA Question-Answering. iii, 1, 7, 11, 12

QC Question Classification. 2, 9, 12

SE Stack Exchange. iii, 1–3, 5, 8, 14, 31, 33

SGD Stochastic Gradient Descent. 22, 30

SO Stack Overflow. iii, 1–3, 5–8, 11, 21, 25, 26, 31, 35

SVC Support Vector Classification. 22, 30, 33

SVM Support Vector Machines. ii, iii, 2, 3, 12, 13

A.2 MySQL Database

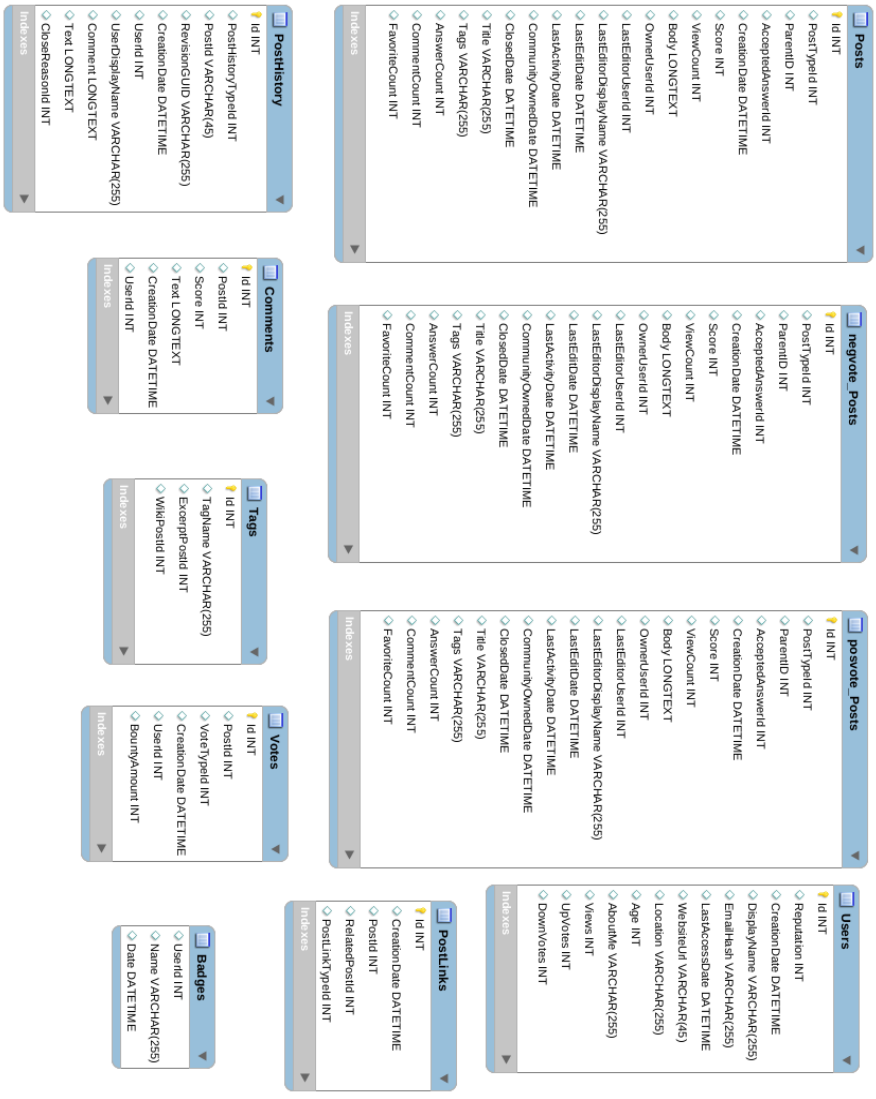

Figure 4: MySQL Database used for dataset

A.3 Tex.Stack Exchange: Data set and Features

	Amount	Oldest	Newest	Vote (lowest)	Vote (highest)
Votes < 0	93	22.08.2010	10.09.2014	-14	-1
Votes $= 0$	5,078	26.07.2010	14.09.2014	0	0
Votes > 0	65,919	18.08.2008	14.09.2014	1	448
All questions	71,090	18.08.2008	14.09.2014	-14	448

Table 10: Overview of the questions in the Tex.StackExhange (August 2015 data set)

	Unprocessed	Features
Score	0.99	0.99
C	1	1
Kernel	Linear	Linear

Table 11: Comparison of raw data set (unprocessed) and feature detectors for Tex.StackExhange (August 2015 data set). Vote score was < 0 for bad and $> +7$ for good.

Actual	Predicted: -1	Predicted: 1
-1	0	15
1	0	2004

Table 12: Confusion Matrix for Tex.StackExhange.

	Precision	Recall	F1-score	Support
-1	0.00	0.00	0.00	15
1	0.99	1.00	1.00	2004
avg / total	0.99	0.99	0.99	2019

Table 13: Classification report for Tex.StackExhange (August 2015 data set).

A.4 Confusion Matrices for Stack Overflow

A.4.1 Confusion matrices for unprocessed and all feature detectors

Table 14: Confusion Matrix for unprocessed data set and all feature detectors using the same parameters.

Table 15: Unprocessed dataset

Actual	Predicted: -1	Predicted: 1
-1	1591	403
1	401	1605

Table 16: All features

Actual	Predicted: -1	Predicted: 1
-1	1558	436
1	401	1605

Table 17: Code blocks

Actual	Predicted: -1	Predicted: 1
-1	1547	420
1	413	1593

Table 18: Hexadecimal

Actual	Predicted: -1	Predicted: 1
-1	1591	403
1	401	1605

Table 19: Homework

Actual	Predicted: -1	Predicted: 1
-1	1590	404
1	400	1606

Table 20: Links

Actual	Predicted: -1	Predicted: 1
-1	1548	410
1	416	1590

Table 21: Numerical

Actual	Predicted: -1	Predicted: 1
-1	1591	403
1	375	1631

Table 22: Tags

Actual	Predicted: -1	Predicted: 1
-1	1508	486
1	467	1539

Table 23: Confusion Matrix for all features, with and without stemming.

Table 24: With stemming

Actual	Predicted: -1	Predicted: 1
-1	1558	436
1	525	1481

Table 25: Without stemming

Actual	Predicted: -1	Predicted: 1
-1	1567	427
1	408	1598

Table 26: Confusion Matrix for the SGD classifier, with loss='log'.

Table 27: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	1610	384
1	594	1412

Table 28: All features with stemming

Actual	Predicted: -1	Predicted: 1
-1	1607	387
1	418	1588

A.4.2 Confusion matrices for singular feature detectors - occurrence only

Table 29: Confusion Matrix for singular feature detectors, only for questions containing it.

Table 30: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	786	202
1	215	768

Table 31: Code blocks

Actual	Predicted: -1	Predicted: 1
-1	793	195
1	218	765

Table 32: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	21	0
1	6	5

Table 33: Hexadecimal

Actual	Predicted: -1	Predicted: 1
-1	21	0
1	6	5

Table 34: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	52	4
1	8	11

Table 35: Homework

Actual	Predicted: -1	Predicted: 1
-1	50	6
1	7	12

Table 36: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	95	56
1	28	337

Table 37: Links

Actual	Predicted: -1	Predicted: 1
-1	87	64
1	30	335

Table 38: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	1044	110
1	247	404

Table 39: Numerical

Actual	Predicted: -1	Predicted: 1
-1	1043	111
1	256	395

Table 40: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	1559	426
1	398	1611

Table 41: Tags

Actual	Predicted: -1	Predicted: 1
-1	1487	498
1	442	1567

Table 42: Unprocessed

Actual	Predicted: -1	Predicted: 1
-1	1284	387
1	342	1499

Table 43: All features

Actual	Predicted: -1	Predicted: 1
-1	1268	403
1	329	1512

A.5 Various screenshots

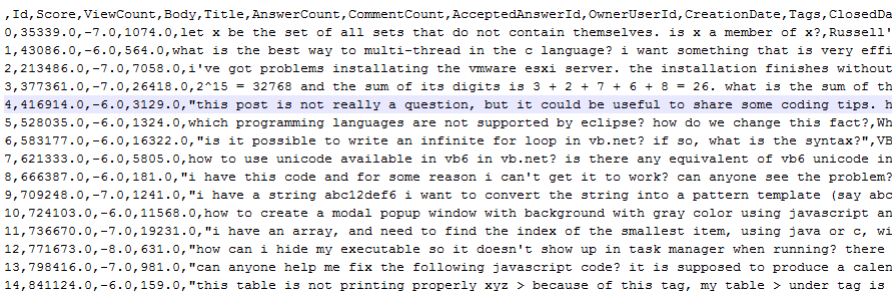

Figure 5: Question without external tags detected.

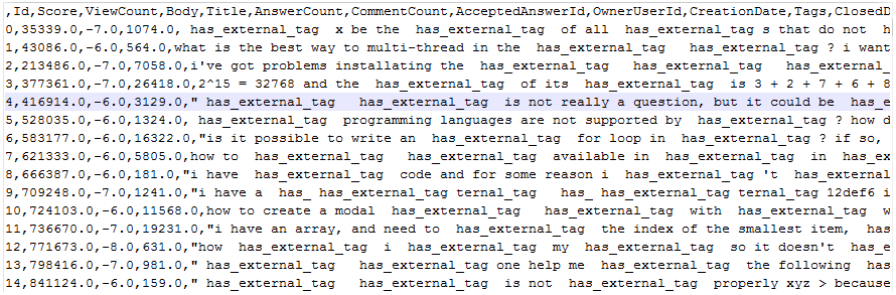

Figure 6: Question with external tags detected.

A.6 Scikit-learns roadmap - Choosing the right estimator

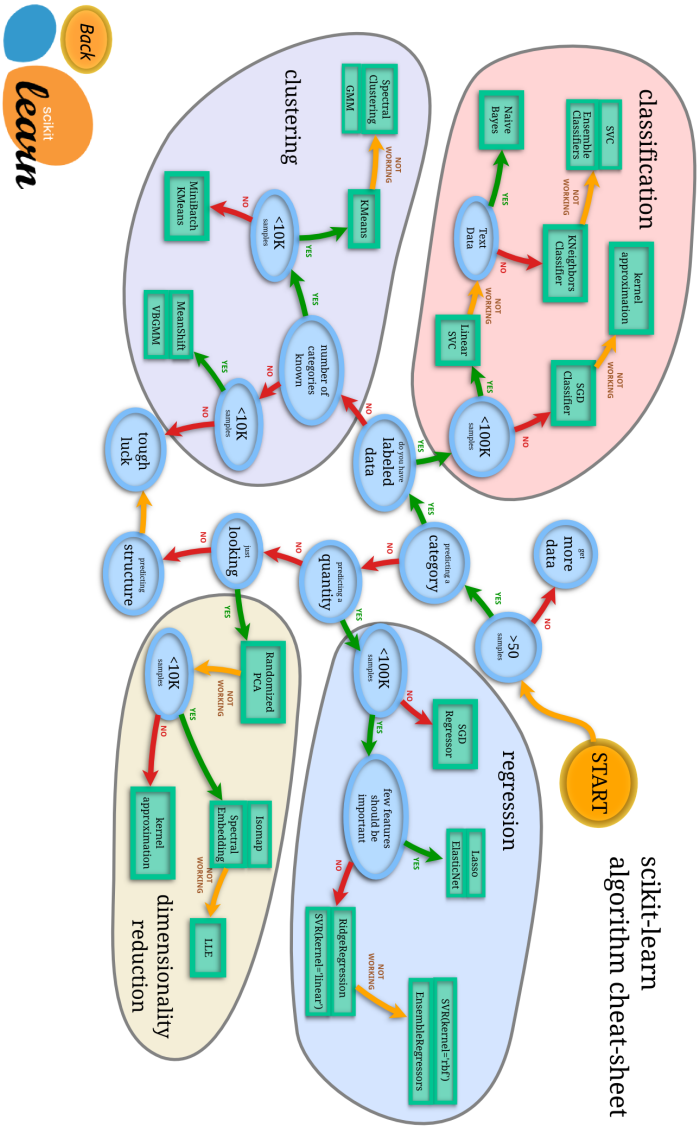

Figure 7: Choosing the right estimator [48].

A.7 Quick installation guide for Windows x64

Since Python is much more adapted to *nix systems then Windows, I decided to write a short guide on how to install Python 3.5 and Scikit-learn on Windows. This guide is provided as-is, and I make no guarantees that this will result in a functioning installation, but it should at least reduce the problems. A presumption here is that the operative system is Windows x64 (if you have x86, you can ignore all the x64 settings). Before installing anything, download the following:

- Python: https://www.python.org/downloads/windows/.
  Select the Python version with "Windows x86-64" in its name.
- CygWin: https://cygwin.com/
- MinGW: http://www.mingw.org/
- MySQL connector for Python (Git; only needed for running this project):
  https://github.com/mysql/mysql-connector-python.
- x64 version of Numpy and Scipy [7, 15]:
  http://www.lfd.uci.edu/~gohlke/pythonlibs/.
  The latest available versions at the time of writing is "*numpy-1.11.1rc1+mkl-cp35-cp35m-win_amd64.whl*" and "*scipy-0.17.1-cp35-cp35m-win_amd64.whl*".
- Scikit-learn (Git): https://github.com/scikit-learn/scikit-learn
- If you do not have a version of Microsoft Visual Studio installed, you need to install Visual Studio 2013 or newer (because compiler requires it)[1].

1. Install Python, update pip, and install these packages:
   pip install -U bs4 pandas nltk matplotlib cython nose nosetests
2. Install MinGW. Thereafter select "mingw32-base" under "MinGW Base System". In addition, under "MinGW Base System", select all belonging to Class "bin" and "dll"
3. Install CygWin. During installation, you will be asked what you want to install. Select all entries that contains "gcc", "mingw64", "make", "automake", "lapack" and "openblas". The GCC-compilers are used in combination with MinGW, because Scikit-learn needs Fortran, Lapack and OpenBlas for *make* to succeed.
4. Change to folder with the x64 version of Numpy and Scipy, and install them [7, 15]:
   pip install "numpy-1.11.1rc1+mkl-cp35-cp35m-win_amd64.whl"
   Verify installation: 1. *python*, 2. *import numpy*, 3. *numpy.__version__*.
   pip install "scipy-0.17.1-cp35-cp35m-win_amd64.whl" and verify using the same steps, swapping numpy with scipy.
5. Start CygWin (run as administrator), change to directory containing Scikit-learn and run the following commands:
   python setup.py build and *python setup.py install*.

[1] "vcvarsall.bat needed for python to compile missing from visual studio 2015": http://stackoverflow.com/q/33323172

6. If you want to use NLTK, you need to run *python*, *import nltk* and *ntlk.download()* to download the corpus.