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 (QnA) 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 (QnA) communities. One well known QnA 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 QnA 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 [55]. 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, 51, 55, 67]. One of the requirements is that the questions should be of good quality [58, 63, 62]. 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, 60].

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 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] [56]. There are several others who have used the same dataset [1, 2, 20, 38, 41, 52, 64, 73]. 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, 51, 67, 73], 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 [69]. 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 compared to some of the work done by others[5]. 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 would therefore have been excluded[6]. 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, meaning that some features that provide a good accuracy score for SO may have no impact in other communities.

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.

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

[5] Wang et al. [70] used 63,863 unique questions, Anderson et al. [2] used 28,722 questions and Treude et al. [67] used 38,419 questions.

[6] The minimum document frequency was set to 0.01, meaning it ignores words that appear in less than 1% of the 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 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)

SO was started in late 2008, and has since grown into a large community. Before SO was launched, the creators of SO ensured a critical mass (domain experts) would be available by inviting developers and programmers [51, 67]. For every question and answer, users can give a vote, which is a measurement of its quality [51]. If you like a question, up-vote ut, if its wrong, down-vote it. Questions and answers can also be sorted based on their vote score (see Figure 1). When doing analysis of questions, this is an excellent way to get the best (or the worst) questions. In addition to answers being ranked by votes, the user asking a question can also select one questions as the "accepted answer" (see Figure 2).

Figure 1: Questions sorted by vote. Questions with an accepted answer is marked with a green background.

Getting votes and having your answer selected as the accepted gives the users reputation. Reputation is an award given to the users for their participation, in addition to earning Badges (similar to achievements in games). The reputation controls how much freedom the user has on SO, everything from posting a comment to becoming a moderator. Reputation can also be used to award bounties. Bounty can be seen as a currency system, where the user trade parts of their reputation to get a satisfactory answer. URL can also be used to easily access questions on SO, as they use keywords in their links. In addition, Google uses their crawlers every 10 second to have the latest updates from in their search engine [17]. You also have full access to all data posted on SO and SE, which can be accessed either online (through their API) or offline (available from StackExchange [56]).

Figure 2: Example of a question on Stack Overflow[1]

2.1.1 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, 73], and [39, 73] 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][2]. In the course 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 [51, 38, 67, 57, 61].

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[3] 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,

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

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

[3] Yang et al. [73] 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)".

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 QnA system (or Internet in general), these are what are commonly referred to as "Trolls" [15, 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 [59]. 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[4].

2.1.2 Stack Overflow (SO) and reputation

Many QnA 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. [67] defines an unanswered question to be a question where no accepted answer is chosen[5]. 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).

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 [61]. Movshovitz-Attias et al. [38] found that users earn more reputation by providing good answers rather than good questions[6]. 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. [73] found that the activity level of a user is not equal to knowledge,

[4] 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.

[5] 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.

[6] 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.

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[7] questions.

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).

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. In the context of learning, questions are used for evaluating the students knowledge, or help them learn something new [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. [53] 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 [53].

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

[7] 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 [73, p. 273].

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. [23] 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 [29] also listed "Whether" and "Discrepancy"[8]. 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[9] [23, 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 [74]. The impact of N-gram is also related to the size of the text being analysed. Zhang and Lee [75] 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'?). [23] 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

[8] "Questions that reflect confusion over a perceived inconsistency." [29, p. 5]

[9] 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) [21, 32, 44, 74].

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 language.

Grammatical properties can be extracted by using Part of Speech (POS), e.g. by using Natural Language Toolkit (NLTK)[10], 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 [23] 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, 26], to identify languages, or filing documents based on content [26]. Documents can belong in more then category, and since categories can overlap they must be treated as a binary classification problem [25] . 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 (QnA)

QnA 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 QnA system, users wants the answer straight away instead of having to read through several documents. In addition, QnA sites allows users to search for questions in the same way they would ask another human (natural language[11]), and there are also different types of QnA sites. Do-

[10] 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 [65, See Section 2.3].

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

main specific QnA focuses on a specific topic (e.g. SO) and open domain where everything goes. QnA 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. [74] 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 [24] could not use TF-IDF based paragraph retrieval, because the paragraphs 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. [72] built an online QnA system for tourism, which consisted of question analysis, information retrieval and answer extraction. Since rule-based approaches 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. If a rule-based approach were to be used in this thesis, it would require creation of categories that were specific to programming.

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 [75] 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, 28, 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:

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

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].

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

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 [28]. 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 [66]. Equation 2.5 - 2.7 shows the kernel functions (K) for polynomial, radial basis function (RBF) and sigmoid (taken from [28, 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 make 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 [66].

Joachims [25] 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.

2.4 Artificial Intelligence (AI) Methods

There are other methods and algorithms that could be applied for question classification. Echihabi and Marcu [12] used a noisy-channel model to retrieve answers based on the question sentence, and Heie et al. [21] used it for answer extraction. Loni et al. [31] compared Back-Propagation Neural Networks (BPNN) and SVM using latent semantic analysis, where BPNN performed better than SVM. Bloehdorn and Hotho [6] combined Boosting and semantics, which got good results, but the results depended on the parameter settings. Zhang and Lee [75] compared Nearest Neighbors, Naive Bayes, Decision Tree, Sparse Network of Winnows and SVM, and found that SVM was the best. They also included suggestions for improvement by adding a tree kernel to weigh up for syntactical differences which could reduce errors by 20%. Figueroa and Neumann [14] experimented with Genetic Algorithms to find the answer type for questions, but answers were mostly returned as unigrams instead of as a sentence.

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

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).

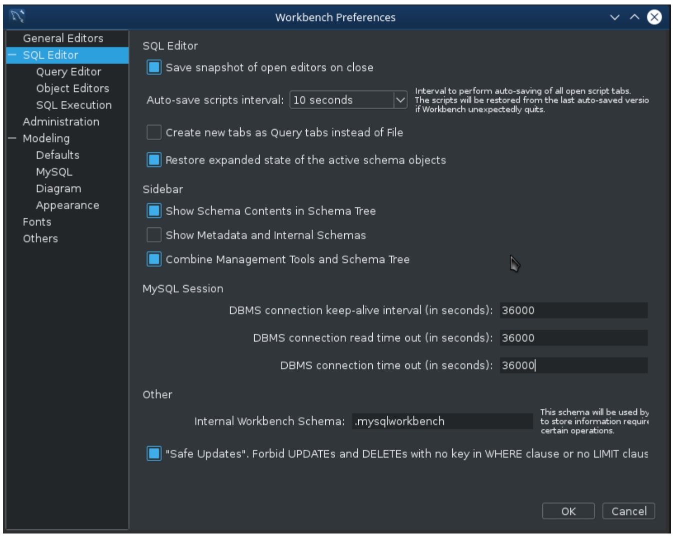

Figure 3: MySQL Workbench: Setting timeout values to avoid connection loss

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 find the tables, even if they still were in the /home/mysql/ folder. It is therefore advisable to

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. 47. 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 Extracting and storing questions for training

To be able to store all the retrieved columns and the belonging rows without creating object classes, the pandas.DataFrame[5] was used. One problem was that if changes were made and the data needed to be updated, the training files had to be re-created from the database. Because of the size of the data set and the fact that the source code was hosted on GitHub, I was hesitant to store the unprocessed training data to file. However, when loading 20,000 samples from the database with a 'WHERE' parameter, things tend to take time. Therefore, a CSV file was created using pandas.DataFrame.to_csv[6]. At first the questions were cleaned of their HTML content, but this would cause issues when detecting code samples and links. Therefore the unprocessed data set was updated to also include the HTML, and is therefore identical to the data in the database.

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/.

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

3.3 Question processing

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 Eloff [13]).

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 [50] [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[7] and bs4[8] (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>[9] (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[10], because the lxml parser failed. The solution was to replace the lxml parser with bs4 and just change the content of

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

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

[9] 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".

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

the <code> tag to the value 'has_codeblock'.

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. This will not have an effect on any rare words, given that this would require the words to appear in less than 200 questions. In most cases these will mostly be dummy words (e.g. words that are used as examples; "abc", "wwwww", "123abc", etc) or language specific terms, which is not of interest. The value for min_df is also specific for this thesis, and if a larger sample size was used, this value would perhaps need to be adjusted. The min_df can be represented either as a percentage (0.0, 0.99) or a specific number (0, n).

3.4 Feature sets, attributes and processing

Since I wanted a balanced data set for both high and low quality questions, the limit was set to 10,000 for both. To be able to retrieve an equal amount for both, the threshold for low quality questions had to be set to -5[14]. As can be seen in Table 1 (created by using code snippet in Listing 3.6), the average vote score was -7.

Listing 3.5: 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
```

[11] http://stackoverflow.com/questions/856307/wordwrap-a-very-long-string.
[12] http://www.nltk.org/
[13] http://pythonhosted.org/pyenchant/
[14] Originally, it was at -10, but this only returned 683 questions.

```
print(temp_df.groupby("label").describe())
# prints all selected columns
print(temp_df.groupby("label").describe(include='all'))
```

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).

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.

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[15]). 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 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

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

contains indicator of word synonyms for homework[16], 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[17] 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[18]). 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. Attached tags are 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'++. Since all the tags were in lower-case, the questions also had to be converted to lower-case to

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

ensure proper matching.

Listing 3.6: 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)
```

3.5 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 3.7 and 3.8. These are the default values used in Scikit-learns tutorials [46, 49] ([23, 34, 75] 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.5, Figure 8 p. 51). Since the sample size in this thesis was only 20,000, the expectation was that SVC would present better predictions.

Listing 3.7: 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 3.8: 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.

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

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

There are two options for exhaustive search in Scikit-learn: GridSearchCV and RandomizedSearchCV. GridSearchCV was selected because it matches all parameters, and as as stated in [37], RandomizedSearchCV can give lower scores.

Before running the grid search, the questions were split into two parts, a training set and a test set by using train_test_split[19]. 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 used for classification tasks [28]. 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.

3.6 System menu

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. For this, argparse was selected. 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. Argparse 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. The system menu is shown in Figure 4.

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

```
PS C:\Users\KnutLucas> cd .\Documents\GitHub\IMT4904_MasterThesis_Code\
PS C:\Users\KnutLucas\Documents\GitHub\IMT4904_MasterThesis_Code> python .\main.py
Program started Tue May 31 10:49:49 2016


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')
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)

Enter option followed by arguments (if any). Enter h to show options: _
```

Figure 4: The system menu

4 Experimental setup and Results

4.1 Experimental setup

A total of six features were detected and converted from each question. These features were code samples[1], hexadecimal, numerical, synonyms for homework and tags (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[2] (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[3].

To be able to compare if a feature would have an impact, the creation of classifier models was divided into four 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. In addition, a training set was also created by selecting only the questions containing the given feature. That way, one would also have a comparison of the features accuracy for the question in which they appeared. A new classifier model based on the unprocessed data set was created for each feature using grid search. This was done to account for the number of question related to each of the singular features.

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

[2] For a visualization of how bad it was, see Figure 6 and 7 in Appendix A.4, p. 50. 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/*.

[3] 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.

In the third part, a comparison was made between the unprocessed classifier model and the classifier model based on all the features[4], using GridSearchCV to select the best parameters. Porter stemming was also included in one of the models using all the features, to see what effect stemming would have.

In the last part, SGD was used to create three classifier models. The first two was based on the unprocessed data set, and the last used all the features. The difference between these is that one of the unprocessed models used grid search with all losses, and the other two had loss='log'. The reason for setting loss='log', is because this allows the model to also present not only a prediction, but also a probability to the user. The confusion matrices for each classifier model is presented in Appendix A.3, 48.

4.2 Results

4.2.1 Comparison of features based on unprocessed data set and all questions

The results for the first and second part of the training is shown in Table 3. 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% (403 misclassified as bad, 401 misclassified as good). The two features that sticks out the most is Numerical and Tags. Numericals accuracy score is 80.55% (375 misclassified as bad, 401 misclassified as good). Tags accuracy score is almost 4% lower than the unprocessed (467 misclassified as bad, 486 misclassified as good).

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

4.2.2 Comparison of features based on unprocessed data set and occurrence

An overview of how many questions contained each feature is given in Table 4. The number used for training and evaluation is shown in Table 5. The results are

[4] Two features were excluded, 'has_external' and 'has_assignment'.

presented in Table 6. Numerical now has a worse accuracy than the unprocessed, but 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 5, only 30% of the training data are good questions. When also looking at the confusion matrix (Appendix A.3.2, p. 49), it predicts more questions as bad.

	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 4: The number of questions containing the given feature.

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

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

4.2.3 Comparison of classifier for unprocessed and stemmed

In addition to comparing the combination of all features against the unprocessed classifier, stemming was also included.

Stemming

There were two stemmers that were considered, Porter and Lancaster. Lancaster is based on Porter, but is more aggressive, and Lancaster replaced words which was not even of the same root[5]. Therefore, Porter was used as the stemmer.

Results

The expectation was that stemming would enhance the prediction and increase the prediction accuracy, but this was not the case. The stemmed feature detector only achieved an accuracy score of 75.97%, which is 3.93% lower than the unprocessed, and 3.15% lower than the non-stemmed classifier. When comparing the stemmed feature detector against the unprocessed and the other two feature detectors including all features, it has the highest misclassification[6]. However, the non-stemmed feature detector that was trained based on grid-search, performs better than the feature detector using the unprocessed parameters. This is because it is better at predicting bad questions.

	Score	Gamma (γ)
All features (stemmed)	75.97%	$1e^{-03}$ (0.001)
All features (no stemming)	79.12%	$1e^{-03}$ (0.001)
All features (unprocessed)	79.07%	$1e^{-04}$ (0.0001)
Unprocessed	79.90%	$1e^{-04}$ (0.0001)

Table 7: Comparison of the classifier for the raw (unprocessed) questions vs. questions with all features. Classifier: SVC, Kernel=RBF and C=1000.

4.2.4 Comparison of Support Vector Classification (SVC) and Stochastic Gradient Descent (SGD)

Table 8 shows a comparison between unprocessed and all features (stemmed) using the SGD classifier. A full exhaustive search for all losses was also ran on the unprocessed, but this is omitted because it resulted in the same values as for loss='log'.

SVC did perform slightly better than the SGD, which can be seen from comparing Table 7 and 8. The classifier for the unprocessed SVC achieved a score of 79.90% vs. SGD which achieved 79.87% (only 0.3% difference). When comparing the feature detector containing all features (where both were stemmed), SVC

[5] E.g. the words "user" and "using" both became "us" with Lancaster, and with Porter it became "user" and "use".

[6] Based on the confusion matrices displayed in Appendix A.3, p. 48.

achieved a score 75.97% and SGD achieved 75.55% (0.42% difference).

Based on the confusion matrices, if one only were interested in predicting bad questions, SGD would be a better option, given that it has the lowest misclassification. However, as seen in Table 8, it uses a max document frequency of 0.5, meaning it excludes words appearing in over 50% of the questions.

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

4.2.5 Applicability

Since this thesis was mainly focused on programming questions and SO, one could ask the question whether or not this is a closed domain system. The answer is that it is an open domain system for SE. SE uses the same database format for all their data, which is listed in the ReadMe.txt included in the data set[7].

To see if the system was expandable, the data for Tex.StackExchange[8] was extracted and added to a database. Given the amount of data available in SO vs. the amount of data in Tex.StackExchange, which would be more predictive[9]? However, this also helps prove a point in relation to whether or not data matters when training a classifier. The results show that SO is more predictive than Tex.StackExchange. Tex.StackExchange got an accuracy score of 99%, but it misclassified all 15 bad questions as good (and in total, it only contains 93 bad questions).

[7] It is also included in the source code repository on GitHub. The database tables were also based on the ReadMe.

[8] The Tex.StackExchange data is based on a data set that was downloaded in August 2015, but the newest question in the data set is from 2014.

[9] One critique is that the number of questions is not comparable, as shown in Table 9.

	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 9: 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 10: 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 11: 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 12: Classification report for Tex.StackExhange (August 2015 data set).

5 Discussions

5.1 Data set and Question selection

The dataset was retrieved from Stack Exchange Archive [56], 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 13, 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 13: Overview of the questions in the Stack Overflow dataset.

Since the threshold values were simply selected, there are three alternatives that 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%) [19]. The equation to calculate the quartiles are shown in Equation 5.1 - 5.3.

$$Q_1 = \frac{(n+1)}{4} \tag{5.1}$$

$$M = \frac{2 \cdot (n+1)}{2} \tag{5.2}$$

$$Q_3 = \frac{3 \cdot (n+1)}{4} \tag{5.3}$$

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 [27, 54, 71].

5.2 Can Stack Overflow (SO) be used to measure question quality?

SO has strict guidelines for their questions, they should be straight to the point and be on topic (programming related) [8, 22]. Their main user base is targeted at professionals and experts, and questions receive vote based on their usefulness. If the question is considered bad, it will receive down-votes, and it can also be closed or deleted. However, for some question there is what I would call a bias factor. If enough people have the same problem, then it will automatically become a good question. Not necessary because of the questions quality, but because enough users have the same problem (e.g. Bugs, IDE behaviour, tweaks, etc).

For bad questions, there are some common denominators. They question has already been asked, the question is school related or show indications that they have not tried to find a solution. The code examples in the bad questions have often syntax errors, are badly formatted, or contain more code than needed to answer the question. The opposite can also be true, where they only present one line of code and ask "Why does not this work?".

To measure the quality of a question, there are various features one could use. Sewak et al. [51] uses votes as a measurement for questions and answers. Users with high reputation could be used as a measurement for expertise [38]. Questions that are marked as on hold, closed, duplicate, etc., can be considered as bad questions. However, for this to be true, one would need to take in into account the number of votes, since duplicate questions can still receive up-votes. There is also the possibility of filtering, through the Tags attached to the questions, allowing you to select those relevant to a specific category or language. Can SO be used to measure question quality? For a programming related domain, the answer is yes.

5.3 Feature and classifier results

Singular features

When it comes to the singular features, it was overall Numerical that achieved the highest and Tags that achieved the lowest score when compared to the unprocessed data set. The problem with Tags is that it uses two feature types, where 'has_external_tags' was not represented for the classifier using all features. Furthermore, when looking at Table A on p. 46 which lists only questions containing the given feature, the feature detector for Tags appear in 19,967 questions. The feature detector using all features appear only in 17,558, which means that at least 2,409 questions contains the external tags. What should have been done was to have trained an additional model, with just the feature for 'has_attached_tag'. If external tags should be included as feature, a filtering mechanism would be needed. The easiest solution would be to run a word test, ignore all English words and save the rest to file.

The same can be said for Homework, since Homework also uses two features, 'has_homework' and 'has_assignment'. A solution for this is already offered by SO. All the site tags that are listed, also list their related synonyms. The solution would then be two combine N-gram and semantic similarity, where one would check word pairs (tag synonyms are listed with hyphens, e.g. "assignment-operator"). However, the question is also how relevant this feature is. From Table 4, one can see that the Homework feature is only represented in 374 questions. If you remove all the false-positives for assignment, it might be less than 200 questions that contain it, meaning it would be excluded (because min_df excludes less than 1%).

Hexadecimal is only present in 160 questions, which means it will be excluded by min_df. Hexadecimal is therefore neither representative, nor a good feature.

The feature for code samples has a better prediction when used over questions containing it, then it does when used for all questions. It is also better at classifying bad questions, which could indicate that questions including a lot of code samples is not considered a good question[1].

All features and stemming

A more accurate comparison on the effect of stemming would could have been achieved by also creating a stemmed classifier for the unprocessed dataset. However, the results from comparing the non-stemmed against the stemmed shows that stemming decreases the predictability. This could be related to the length of the questions, because words that were previously excluded is now included (which a comparison against unprocessed could have shown). One improvement that was shown, was that the non-stemmed improved its accuracy vs. the the one using the unprocessed classifiers parameters. This not only shows that the parameters have an effect on the accuracy, but also that if each singular feature detector had been trained using grid search, the results would not have been comparable.

[1] 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".

Support Vector Classification (SVC) and Stochastic Gradient Descent (SGD)
As expected, SVC was better at prediction than SGD, but only with 0.42% (for the unprocessed data set classifiers). This could indicate that if one were to increase the sample size to 30,000 or higher, SGD could be a better classifier than SVC. However, one problem with SGD is that it excludes words that appear in more than 50% of the questions (max_df). This could be related to it filtering out code samples and dummy words, since the classifier using all features uses up to 75% of the words. For the SVC, the max document frequency was set to ignore more than 95% of the words in a question (because I was more interested in the less used words). Which means that an improvement for the SVC would be to allow it to also select the max document frequency to see whether or not this would improve its accuracy.

5.4 Limitations and other issues

One potential issue is that it is the development version (v0.18.dev0) of Scikit-learn and not the stable version (v0.17.1) that was used. Due to various installation issues, the development version from GitHub was used. In the development version, a lot of changes have been added, where modules have been moved and function parameters removed (making a lot of tutorials outdated). The question became at one point whether or not a switch should be made from using the development version into the stable version. However, 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. 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. As previously mentioned, the Tags and Homework should have had an additional classifier training, using only the words for Homework and the tags attached to the question. The grid searches have only been used over a short set of parameters, which mean that the parameters it selected is not the best ones. Instead, it selected those because it could not go higher (or lower) for the given parameter. There is neither any comparison of feature detectors for the SGD, as was done with the SVC. This was excluded due to the time it takes to train each classifier.

Tex.StackExchange does not contain enough questions to be on equal comparison terms with the SO data set. However, it was selected based on the expectation that it would not contain an equal amount of data. For Tex.StackExchange the score had to be set to greater than zero to even have enough bad questions to compare with. Which leads to the score retrieval. When retrieving questions based on vote score, there is no sorting or selecting of the best or worst. It just selects the values that are greater or lower then the set value. This means that if you were to select all the best and worst (sorted on vote score), you could perhaps get better accuracy score.

One of the more peculiar issues (which I do not have an explanation for), was when the proper 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.

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[2]. 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 [18]. To ensure that progress was made, and that the 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, 68]. 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[3].

[2] 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.

[3] 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`.

6 Conclusion

6.1 Conclusion

SO is today a well known QnA community for programmers, and has existed since late 2008. SO can be seen as a gamification system, that awards users for their participation by giving out reputation and badges. If the question (or answer) that is posted is considered of good quality by the community, it gets upvoted. SO has a strict policy on what type of questions are allowed to be posted, and it was therefore interesting to see if this could be used to measure question quality.

In this thesis, a data set containing all information posted within the SE community was acquired. The data set (downloaded in March 2016) for SO contains a total of 11,203,031 questions. Two classifier types were tested, SVC and SGD on a total of 20,000 questions. The testing was done on both the unprocessed questions and questions containing one or more feature detectors. The classifier using SVC had the highest accuracy score, where the Numerical feature got an accuracy score of 80.55%, but only when trained on all questions. The unprocessed got second place, with an accuracy score of 79.90%. However, when training the classifiers only on questions that contained the given feature, numerical got a lower score than the unprocessed. Which could mean that the feature only did better for all questions, because it had more questions (and not feature occurrences) to learn from.

The classifier using SGD achieved a slightly lower score than SVC, with an accuracy score of 79.87%. This could indicate that the SGD would perform better for a larger data set, if the number of questions was increased to 30,000 or more. The comparison of the confusion matrices for the singular feature detector also show that it is easier to predict bad questions, rather than good.

6.2 Further work

Further development of the system would include analysis of the code, to check for syntax errors. Sentiment analysis could also be of interest, since SO does not want greetings or gratefulness in their questions [8]. For the sentiment analysis, symbol checking would also be included (e.g. '?', '!', emoticons, etc). Version numbering is one of the features that were excluded, due to the complexity[1]. It

[1] A version number can contain numbers, letters and symbols, in addition to including a "v." or "version", or product name.

would also be interesting to see what results the SGD would give when training a classifier on each of the singular features.

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] Abdessamad Echihabi and Daniel Marcu. “A Noisy-channel Approach to Question Answering”. In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. ACL ’03. Sapporo, Japan: Association for Computational Linguistics, 2003, pp. 16–23. DOI: 10.3115/1075096.1075099. URL: http://dx.doi.org/10.3115/1075096.1075099.

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

[14] Alejandro G Figueroa and Günter Neumann. “Genetic algorithms for data-driven web question answering”. In: *Evolutionary computation* 16.1 (2008), pp. 89–125.

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

[16] 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/.

[17] 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).

[18] 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).

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

[20] 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.

[21] 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.

[22] 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).

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

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

[29] 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.

[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. *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).

[51] 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).

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

[53] 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.

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

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

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

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

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

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

[60] 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).

[61] 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).

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

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

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

[65] 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).

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

[67] 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.

[68] 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).

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

[70] Shaowei Wang, David Lo, and Lingxiao Jiang. "An Empirical Study on Developer Interactions in StackOverflow". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. SAC '13. Coimbra, Portugal: ACM, 2013, pp. 1019–1024. ISBN: 978-1-4503-1656-9. DOI: 10.1145/2480362.2480557. URL: http://doi.acm.org/10.1145/2480362.2480557.

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

[72] 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.

[73] 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.

[74] 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.

[75] 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, 32

BOW Bag of Words. 9

IDE Integrated Development Environment. 23, 32

IR Information Retrieval. 10

ML Machine Learning. iii, 2, 3, 32

NLTK Natural Language Toolkit. 10, 18, 53

POS Part of Speech. 10

QC Question Classification. 2, 8, 11

QnA Question-Answering. iii, 1, 7, 10, 11, 36

SE Stack Exchange. iii, 1–3, 5, 14, 29, 36

SGD Stochastic Gradient Descent. 22, 26, 28, 29, 34, 36, 37

SO Stack Overflow. iii, 1–8, 11, 21, 29, 31–34, 36

SVC Support Vector Classification. 22, 28, 34–36

SVM Support Vector Machines. ii, iii, 2, 4, 11–13

A.2 MySQL Database

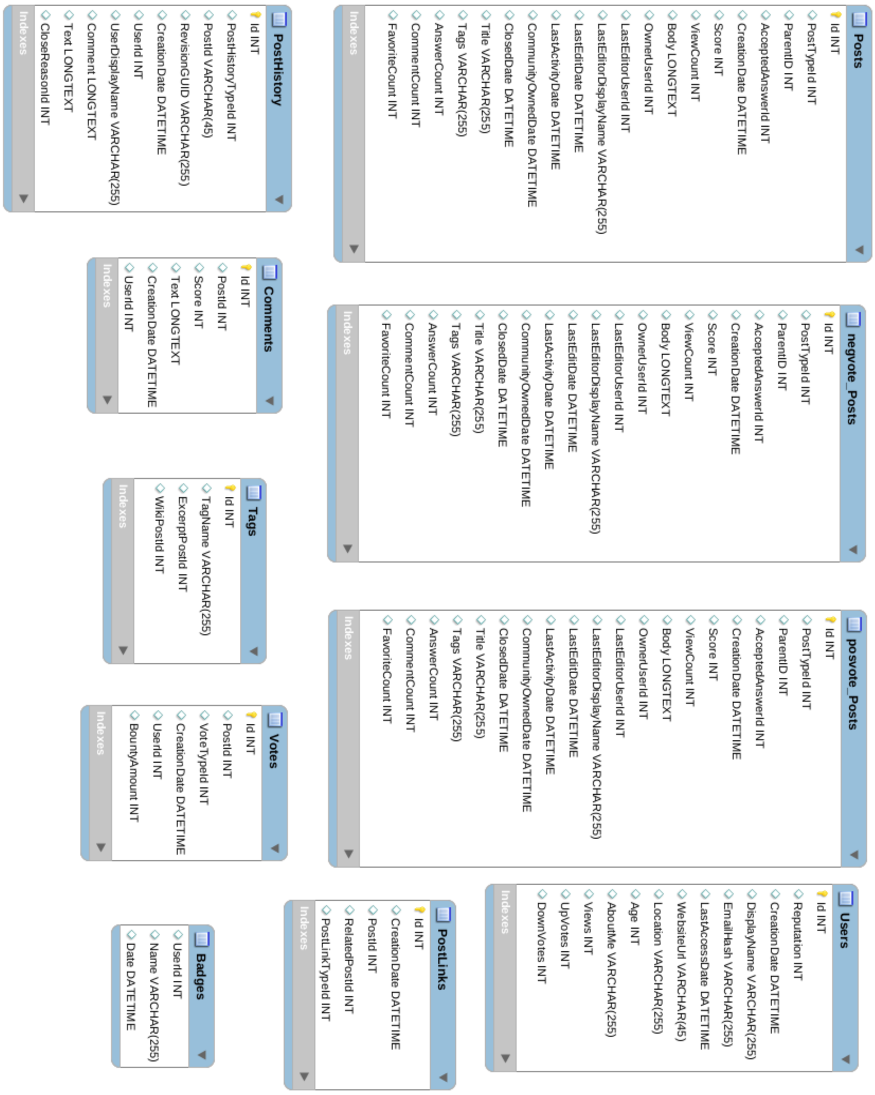

Figure 5: MySQL Database used for dataset

A.3 Confusion Matrices for Stack Overflow

A.3.1 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	1607	387
1	418	1588

Table 28: All features with stemming

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

A.3.2 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.4 Comparison of questions with and without ’has_external_tag’

```
,Id,Score,ViewCount,Body,Title,AnswerCount,CommentCount,AcceptedAnswerId,OwnerUserId,CreationDate,Tags,ClosedDa
0,35339.0,-7.0,1074.0,let x be the set of all sets that do not contain themselves. is x a member of x?,Russell'
1,43086.0,-6.0,564.0,what is the best way to multi-thread in the c language? i want something that is very effi
2,213486.0,-7.0,7058.0,i've got problems installating the vmware esxi server. the installation finishes without
3,377361.0,-7.0,26418.0,2^15 = 32768 and the sum of its digits is 3 + 2 + 7 + 6 + 8 = 26. what is the sum of th
4,416914.0,-6.0,3129.0,"this post is not really a question, but it could be useful to share some coding tips. h
5,528035.0,-6.0,1324.0,which programming languages are not supported by eclipse? how do we change this fact?,Wh
6,583177.0,-6.0,16322.0,"is it possible to write an infinite for loop in vb.net? if so, what is the syntax?",VB
7,621333.0,-6.0,5805.0,how to use unicode available in vb6 in vb.net? is there any equivalent of vb6 unicode in
8,666387.0,-6.0,181.0,"i have this code and for some reason i can't get it to work? can anyone see the problem?
9,709248.0,-7.0,1241.0,"i have a string abc12def6 i want to convert the string into a pattern template (say abc
10,724103.0,-6.0,11568.0,how to create a modal popup window with background with gray color using javascript an
11,736670.0,-7.0,19231.0,"i have an array, and need to find the index of the smallest item, using java or c, wi
12,771673.0,-8.0,631.0,"how can i hide my executable so it doesn't show up in task manager when running? there
13,798416.0,-7.0,981.0,"can anyone help me fix the following javascript code? it is supposed to produce a calen
14,841124.0,-6.0,159.0,"this table is not printing properly xyz > because of this tag, my table > under tag is
```

Figure 6: Question without external tags detected.

```
,Id,Score,ViewCount,Body,Title,AnswerCount,CommentCount,AcceptedAnswerId,OwnerUserId,CreationDate,Tags,ClosedD
0,35339.0,-7.0,1074.0, has_external_tag  x be the  has_external_tag  of all  has_external_tag s that do not  h
1,43086.0,-6.0,564.0,what is the best way to multi-thread in the  has_external_tag   has_external_tag ? i want
2,213486.0,-7.0,7058.0,i've got problems installating the  has_external_tag   has_external_tag   has_external_
3,377361.0,-7.0,26418.0,2^15 = 32768 and the  has_external_tag  of its  has_external_tag  is 3 + 2 + 7 + 6 + 8
4,416914.0,-6.0,3129.0," has_external_tag   has_external_tag  is not really a question, but it could be  has_e
5,528035.0,-6.0,1324.0, has_external_tag  programming languages are not supported by  has_external_tag ? how d
6,583177.0,-6.0,16322.0,"is it possible to write an  has_external_tag  for loop in  has_external_tag ? if so,
7,621333.0,-6.0,5805.0,how to  has_external_tag   has_external_tag  available in  has_external_tag  in  has_ex
8,666387.0,-6.0,181.0,"i have  has_external_tag  code and for some reason i  has_external_tag 't  has_external
9,709248.0,-7.0,1241.0,"i have a  has_ has_external_tag ternal_tag   has_ has_external_tag ternal_tag 12def6 i
10,724103.0,-6.0,11568.0,how to create a modal  has_external_tag   has_external_tag  with  has_external_tag  w
11,736670.0,-7.0,19231.0,"i have an array, and need to  has_external_tag  the index of the smallest item,  has
12,771673.0,-8.0,631.0,"how  has_external_tag  i  has_external_tag  my  has_external_tag  so it doesn't  has_e
13,798416.0,-7.0,981.0," has_external_tag   has_external_tag one help me  has_external_tag  the following  has
14,841124.0,-6.0,159.0," has_external_tag   has_external_tag  is not  has_external_tag  properly xyz > because
```

Figure 7: Question with external tags detected.

A.5 Scikit-learns roadmap - Choosing the right estimator

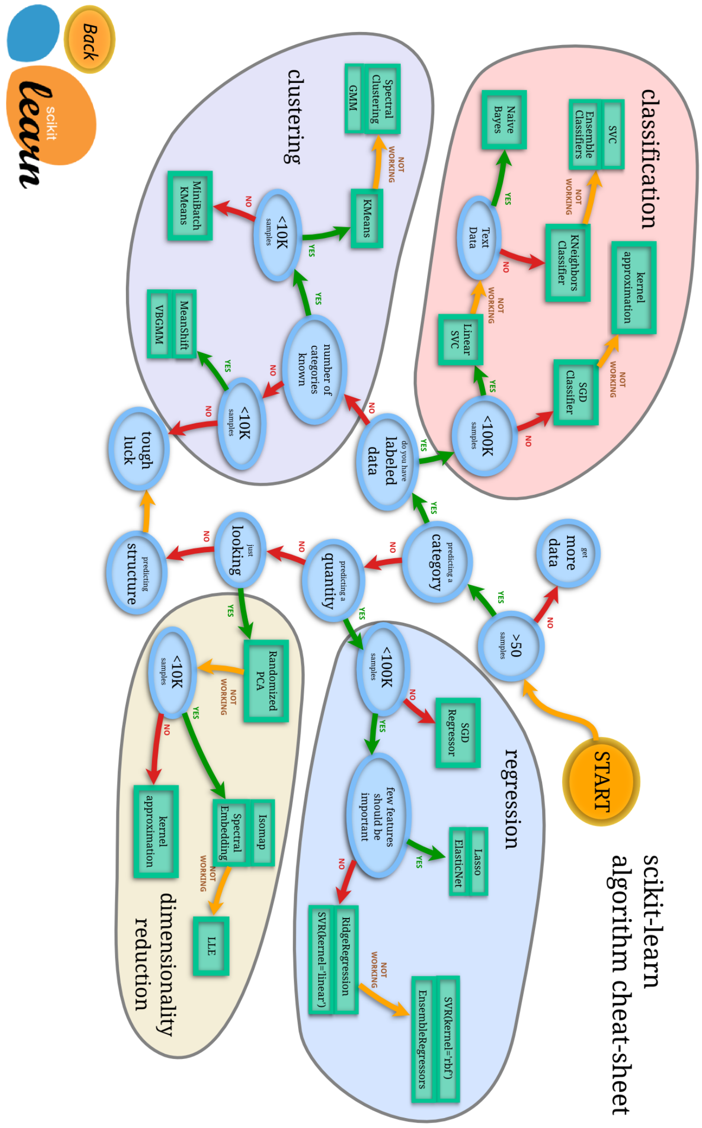

Figure 8: Choosing the right estimator [48].

A.6 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, 16]:
  `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, 16]:
   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.