TRINITY COLLEGE DUBLIN

Classifying the Quality of Questions and Answers From Stack Overflow

by

Geoffrey Hodgins

A dissertation submitted in partial fulfillment for the
degree of Masters of Computer Science

in the department of
Integrated Computer Science
Computer Science and Statistics

Submitted to the University of Dublin, Trinity College, May 2016

Declaration of Authorship

I, Geoffrey Hodgins, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signed: ____

Date: ____

Summary

This dissertation aimed to discover indicators of quality, and to use this knowledge to correctly classify the quality of questions and answers from Stack Overflow. The proliferation of technical questions and answers on Q&A websites such as Stack Overflow means there is more information available than ever. However, the ease of publishing such information also tends to mean the quality varies significantly. The job of moderating Stack Overflow is left to the community, which is an extremely time-consuming job for what are essentially volunteers. Stack Overflow performs some basic quality analysis and automation, but this is an area where improvement would have many benefits to not only Stack Overflow, but many other domains where the quality of text is important.

Posts from Stack Overflow are labelled with their respective quality levels, which are inferred from other attributes in the data such as the score of a post. These labels allow the extracted features to be used to predict the quality of new and unseen posts. These features are extracted using several techniques such as readability indexes, basic counts, character n-grams, and sentiment analysis. Readability indexes attempt to estimate the complexity of a text. Basic counts find how many times something occurs in the text, such as code tags or words. Character n-grams generate features by counting the frequency of character sub-strings in the text, eg. “the” $\rightarrow$ [“t”, “h”, “e”, “th”, “he”, “the”]. Sentiment analysis estimates how positive or negative the emotion is, and how objective or subjective the text is. Machine Learning techniques are leveraged to discover empirical evidence and anecdotal insight. Supervised machine learning algorithms train on the extracted features of quality labelled questions and answers. Specifically, Random Forests are used due to their white-box nature, which allows the internal workings to be interpreted, analyzed, and visualized.

Promising quality classification performance and interesting insights are found. These results indicate that many of the features found strongly correlate with the quality of questions and answers, and are statistically significant in all cases tested, showing that a noticeable difference or pattern is found. These results come in several forms. Tables of performance metrics showing the recall, precision, and F_1 score for each class for question and answer quality classification are useful for comparing the performance of the classifiers. The F_1 scores for question and answer quality prediction were 0.45 and 0.42 respectively, which is a significant improvement over a baseline prediction rate of 0.25 for a 4 label classification. However, the performance for the two good quality labels is far better than the performance for the two bad quality labels. This leads to further analysis into why this might be and why the lack of deeper content validation is a prime suspect for this inaccuracy. The behaviour of the classifiers is visualized in the form of confusion matrices, and histograms showing how important individual features are in correctly predicting the quality. The tables containing distribution statistics of the features shows how they vary and correlate across the quality classes for questions and answers.

Based on these results, a number of areas for further work are outlined that would greatly benefit this research area. These suggestions include improving tools for technical text data, and content analysis. An argument could be made that the content of a question or answer is the most important influence on the perceived quality of a question or answer by a person. However, this kind of functionality is by no means easily implemented with the current open tools, and is more akin to the cutting edge analysis performed by IBM's Watson. To achieve truly accurate quality analysis in the future, major research will need to focus on validation of the information contained in a question or answer. However, regardless of how advanced these techniques become, quality will always remain a subjective notion that varies from person to person. This means that achieving perfect performance is not plausible or probable. In an ideal system, the accuracy of the quality classifications will be comparable to that of a subject matter expert. The work performed in this dissertation is a step along the way to this becoming a reality.

TRINITY COLLEGE DUBLIN

Abstract

Integrated Computer Science
Computer Science and Statistics

Masters of Computer Science

Classifying the Quality of Questions and Answers From Stack Overflow

by Geoffrey Hodgins

This dissertation aims to discover indicators of quality, and to use this knowledge to correctly classify the quality of questions and answers from Stack Overflow. The proliferation of technical questions and answers on Q&A websites such as Stack Overflow means there is more information available than ever. However, the ease of publishing such information also tends to mean the quality varies significantly. The job of moderating Stack Overflow is left to the community. Stack Overflow performs some basic quality analysis, but this is an area where improvement would have many benefits to not only Stack Overflow, but many other domains where the quality of text is important.

Machine Learning techniques are leveraged to discover empirical evidence and anecdotal insight. Specifically, Random Forests are used due to their white-box nature, and features such as readability indexes and similarity measures were engineered from raw text data. Promising quality classification performance is found, along with interesting insights from analyzing the Random Forest classifier. Based on these results, a number of areas for further work are outlined that would greatly benefit this research area. These suggestions include improving tools for technical text data, and content analysis.

Acknowledgements

I would like to thank my supervisor, Dr. Carl Vogel, for his guidance throughout this dissertation. He has helped shape its direction from when it was merely a rough idea in my head, to the implementation that it is now. My parents, Victor and Olwen, deserve a huge amount of credit for supporting me through the last five years of college, and the entirety of my life. I would also like to thank Lillian Hayes, for her constant support and stream of food and coffee.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What Is the Purpose Of This Dissertation?

This dissertation aims to discover indicators of quality, and to use this knowledge to correctly classify the quality of questions and answers from Stack Overflow. Stack Overflow is a popular questions and answers (Q&A) website (a Stack Exchange Q&A community) where users exchange knowledge about various topics related to programming. As of May 2016, there are more than 5.5 million users. These users have asked 12 million questions, supplied 19 million associated answers, and left 48 million comments on these posts. About 73% of the questions on Stack Overflow are answered. [StackExchange, 2015a] [StackExchange, 2015b] Whilst this figure is quite impressive, why are 27% of the questions on Stack Overflow not answered? [Asaduzzaman et al., 2013] What features of questions attract users to answer them? Why do some questions receive more answers and why are some answers of a higher quality? There are many questions here that are of interest that may be answered by delving into the data. The core of this dissertation is to explore quality, and how it varies in questions and answers and in the process, discovering insight into the subjective measures of quality in an individual community such as Stack Overflow. The central question of this dissertation is "Can the quality of a question or answer be predicted at the time of creation using just its text?".

1.2 Why Classify Question and Answer Quality?

The quality of the content provided by Q&A websites varies, and ranges from high-quality to low-quality. This low-quality content can also be abusive or dangerously ignorant/misleading in addition to merely being a poor question or answer. The job of moderating Stack Exchange Q&A's is left to the community and its moderators, and is an extremely time-consuming job for what are essentially volunteers. Stack Overflow performs some basic quality analysis and automation, but this is an area where improvement would have many benefits to not only Stack Overflow, but many other domains where the quality of text is important. This dissertation delves into the problem of analyzing and classifying the quality of questions and answers. Through this, it is possible to discover what affects a question's or answer's quality, and how the process of quality monitoring can be at least partially automated. Additionally, the techniques should generalize across multiple domains including other Stack Exchange Q&A communities.

Stack Overflow is rapidly growing both in the size of its community and the amount of posts it maintains. As the community continues to grow, the rate of new posts will only increase. The Stack Exchange Data Explorer, a query was run outputting how many new posts (questions and answers) have been created per month in 2014 and 2015. [Hodgins, 2015a] The output of this query is shown in table 1.1 on page 3, illustrating how difficult it is to monitor the quality of each and every post.

There are many applications for the techniques that are part of this research. Q&A forums could use these techniques to analyze the quality of the questions and answers as they are posted in order to identify and stop low quality (possibly abusive or spam) posts from being created. These techniques could be used to analyze the quality of technical documentation for software, if built into a continuous integration environment for software engineering, they could reject commits unless the documentation meets a certain quality standard. Journals could check that submitted papers meet the style and quality standards before being manually read by an editor. A search engine could even use these techniques as an additional quality metric for indexing online content.

Year	Month	QuestionCount	AnswerCount
2014	1	259862	366526
2014	2	260462	352897
2014	3	289103	375051
2014	4	274175	348076
2014	5	258159	313942
2014	6	238820	285422
2014	7	255455	312292
2014	8	236526	288915
2014	9	242139	291649
2014	10	252152	299617
2014	11	244719	288977
2014	12	228868	277938
2015	1	241315	287328
2015	2	242689	287014
2015	3	274107	317014
2015	4	266915	331192
2015	5	261572	321582
2015	6	265432	319613
2015	7	274544	330984
2015	8	255120	307888
2015	9	252880	301136
2015	10	267426	312286
2015	11	256791	294466
2015	12	246929	293671

Table 1.1: Count of New Questions and Answers Per Month in 2014 and 2015

1.3 How Successful Was This Dissertation?

This dissertation results in promising quality classification performance, indicating that many of the features found strongly correlate with the quality of questions and answers. The results are statistically significant in all cases tested, showing that a noticeable difference or pattern is found. These results are evaluated, generating several interesting insights. The benefits and limitations of the techniques used are discussed, and alternative approaches are suggested to circumvent them. The content of this dissertation is a significant foundation for others to learn from and build upon. Regardless of how advanced these techniques become, quality will always remain a subjective notion that varies from person to person. Ideally, the accuracy of the quality classifications would be comparable to that of a subject matter expert. While the accuracy of quality classification is far from perfect, the empirical and anecdotal evidence presented in this document shows that it is a significant step in the right direction. Therefore, this dissertation can be deemed a success, as it has performed and evaluated its core goals.

1.4 Roadmap

Section 1, Introduction starting page 1, just explained what the purpose of this dissertation is and why this is something that is useful to do.

Section 2, Related Works starting page 5, lays a foundation of work and knowledge in this domain that this dissertation builds on.

Section 3, Methodology starting page 9, gives an overview of the data that is the heart of this dissertation and discusses the important techniques used throughout this dissertation in order to accomplish the stated goals.

Section 4, Implementation starting page 42, discusses the implementation details of the methods and techniques listed above. This section includes concrete details about how the analysis was carried out and the system was built.

Section 5, Results starting page 53, shows the results generated by the implementation and analyzes the significance and outcomes of them.

Section 6, Evaluation starting page 62, evaluates and discusses how successful the dissertation was, both in terms of empirical results and in general.

Section 7, Conclusion starting page 73, summarizes the purpose, arguments, and results of the dissertation.

Chapter 2

Related Works

This section discusses related works to this dissertation. Some share the goal of analyzing a question and/or an answer's quality, whereas others do not share the same goal but find insight into an important element of our task. These are just some of the related works important to this dissertation, there are many insightful papers in the last several years in this space showing that it has become an area of much interest in both academia and industry. This dissertation hopes to build on the discoveries of the these papers, so that useful insights can be provided to others asking similar questions in the future.

2.1 Classifying Question Quality

Ponzanelli et al. [2014a,b] analyze Stack Overflow questions in order to predict quality at the time of creation. The first paper concentrates on understanding what metrics influence the quality of a question. Decision trees are utilized to do this as their output can be easily interpreted, allowing the authors to gain intuition about question quality. However, the Decision tree is succeeded by the use of a genetic algorithm for the purpose of classification. The second paper focuses on detecting low quality posts, and also uses a genetic algorithm. The full data dump as of September 2013 containing 5,648,975 questions was filtered and categorized into 4 separate classes; Very good, Good, Bad, and Very bad. Very Good/Good questions had to have an accepted answer, not be closed or deleted, and have a positive ranked score. Bad questions had to have a negative score, and Very bad questions were closed or deleted. Questions with the score of 0 are assumed to have not attracted enough interest from the community to evaluate and classify their quality with the information available. After the filtering, 1,262,959 questions were left in the data set across the 4 categories. Three feature metric sets are used; Stack Overflow, readability, and popularity metrics. Stack Overflow metrics include simple metrics such as URL/Email count and body length. Readability metrics include more complex structural and readability metrics such as word and sentences count, percentage of code, and Automated Reading Index (ARI). Popularity metrics include metrics such as reputation of the author and number of badges, including specifically question and answer badges. A linear quality function was learned for each metric set using genetic algorithms implemented with the framework JGAP. Popularity and readability metrics are found to be the most effective. However, readability is found to be effective alone and therefore seems to be highly correlated with quality. This dissertation utilises a subset of these predictor metrics with a focus on textual metrics at time of posting as a solid foundation for analyzing the quality of not just questions, but also answers.

2.2 Insight into Successful Answers

Presentation quality, time and social factors are all commonly studied predictors of question and answer success on Q&A platforms. Calefato et al. [2015] suggests that affective factors also strongly influences an answers success. Affective factors are emotional factors which can negatively or positively affect learning. The paper investigates how Stack Overflow users can increase the chance of getting their answer accepted, focusing on actionable factors that can be acted upon by users when writing an answer and making comments. The actionable factors in the model include presentation quality, affect, time and social. Each of these factors has predictor variables used to filter low quality posts. Presentation quality is important in order to comply with the community standards for answers. Presentation predictors include length, uppercase ratio, code snippets, and URL count. Affect predictors include sentiment metrics (positive vs negative polarity) and emotion intensity. Care must be taken with sentiment in the technical domain however, as many sentences that a technical user would not judge to be negative are predicted as such. Time predictors include arrival order, and time taken to answer since the question was posted. Expert users tend to be the fastest contributors, as there is a high probability that the first answer will be accepted. Additionally, the longer the wait time for an answer the less likely it is for an answer to be eventually accepted. Reputation predictors include the author's reputation, and their number of badges. The results of this study recommends potential contributors to be prompt in replying, to comply with the Stack Overflow presentation quality standard by including code snippets and URLs for providing contextual information, to avoid a negative attitude towards information seekers, and to follow-up in the comments with a positive discussion. Recommended further work from this study is to investigate to what extent emotions and personality traits shown in questions influence the answering behavior and to evolve sentiment detection for domain-specific applications such as a technical Q&A platform like Stack Overflow. In our work, the important predictors identified above can be used to aid in predicting both question and answer quality.

2.3 Quality Code, Quality Question

Duijn et al. [2015] analyze code fragments from Stack Overflow questions in order to improve the classification of high and low quality questions. Stack Overflow guidelines state the best questions contain a bit of code but not entire software programs, ideally just enough code for others to reproduce the problem. Analyzing the code fragments in isolation achieves similar performance to classification based on a wider set of metrics. They combine the code-to-text ratio, and code only metrics such as code readability. By using roughly 30 metrics questions were classified as either 'good' or 'bad' with an accuracy of approximately 80%. The metrics used for classification are a combination of readability metrics, metrics based on the constructs found during qualitative analysis (such as the use of "==" in Java, a common mistake when comparing objects), and the number of errors reported by a formatting style checker. Three classification algorithms were tested; decision tree, logistic regression, and random forest. In order to determine the most important features they used the feature importance of the random forests algorithm, metrics correlations, and created a decision tree analysis. Important code metrics related to question quality are those most important for general code readability, such as length of the lines, whitespace occurrence or number of formatting errors. The Pearson correlations also show that certain constructs should be avoided, such as print line statements that have a relatively high negative correlation with score, indicating that these subtract value from the code fragment. This paper shows incentive to analyse code contained in Stack Overflow questions and answers, in addition to all the other prediction metrics mentioned previously.

2.4 Emojis and Hedging Affecting Perception of Quality

Vogel and Sanchez [2012, 2013, 2015] delve into how hedging can influence the success of posts and how to discover and annotate hedges, including on Q&A platforms. They provide evidence "that forum posts using hedges are more likely to get high ratings of their usefulness". Hedges are "linguistic expressions whose contribution to sentence meaning is a modulation of the accuracy of the content they embed". This means that authors can manipulate readers interpretation of their question or answer by using these linguistic hedges, particularly singular first person epistemic phrases. Examples of singular first person epistemic phrases are; I think, I dont know, I know. Examples of words conveying non-phrasal hedging are; appear, seem, sometimes, suggest, unclear, think. Significantly, hedges are not just linguistic terms but can also be in the form of "emoticons" or "emojis". An example of an emoji is a "smiley", which is a stylized representation of a smiling humanoid face and looks like ":)". Emojis have become an increasingly common and important part of communicating in online communities, and are powerful indicators of sentiment. "Posts with no hedges are the ones awarded least kudos." This means it is important to take this into account when predicting the quality of a post, as it is essentially the perceived quality of the post being predicted, which hedges affect.

Chapter 3

Methodology

The Methodology section discusses the Stack Overflow data and important techniques used throughout this dissertation. The primary step necessary for this dissertation is to aggregate training and testing data to analyze and learn from. The data must also be in a format and storage medium that allows easy and flexible interaction and analysis. Methodology directly related to the Stack Overflow data such as where to find the data, what is in it, why to use it, managing it, and insights into the data is covered in the Data section. Next, machine learning techniques will be utilised to extract features, to generate a model using supervised machine learning techniques, and to use this model to perform classification. The methods used to do this are covered in in the Machine Learning section starting on page 19. The results of these methods will then be analyzed to gain insight into how indicative they are of quality of questions and answers from Stack Overflow, and how accurate the predictions of the model are on new and unseen data. Methodology for performing this analysis is covered in the Methods of Evaluation section starting on page 37.

3.1 Data

The Stack Overflow dataset of posts is central to this dissertation. Therefore this section is solely dedicated to the following:

- Where to Find the Dataset?
- What is in the Dataset?

- Why Use this Dataset?
- Managing the Data
- Insights Into The Data

3.1.1 Where to Find the Dataset?

The Stack Overflow dataset can be found on the Internet Archive's website. [StackExchange, 2015c] The Internet Archive is a non-profit that was founded to build an Internet library. Its purposes include offering permanent access for researchers and others to historical collections that exist in digital format. Importantly for this dissertation, they maintain a dump of all user-contributed content on the Stack Exchange network. Each site is formatted as a separate archive consisting of XML files zipped via 7-zip using bzip2 compression. Each site archive includes Posts, Users, Votes, Comments, PostHistory and PostLinks. This archive appears to currently update the dataset every 2 months or so. There are specific attribution requirements associated with this data, in that the author of the content and Stack Overflow must be digitally and visually accredited for it. This dissertation focuses on the Stack Overflow Posts dataset, and was downloaded on 03/01/2016. Since then, an update has been made to this dataset. This update occurred on the 01/03/2016 and updates the dataset with content from January and February 2016.

3.1.2 What is in the Dataset?

The StackOverflow dataset contains vast amounts of real world data from the StackOverflow StackExchange Q&A domain. As mentioned previously in the introduction, as of May 2016 there are more than 5.5 million users who have asked 13 million questions, supplied 19 million answers, and left 48 million comments on these posts. [StackExchange, 2015a] The first and last posts contained within it are from and 31/07/2008 03/01/2016 respectively.

The dataset's schema is documented in the Stack Exchange Data Explorer and on a "meta.stackexchange.com" post. Meta Stack Exchange is where users of StackExchange Q&A domains can discuss bugs, features, and support issues that affect the software powering all 155 Stack Exchange communities. [StackExchange, 2015d] Table 3.1 Schema of

Key	Description
Id	Unique identifier for each question or answer
PostTypeId	$1 = Question$, $2 = Answer$
AcceptedAnswerId	Id of the accepted answer for a question
ParentId	Id of the question an answer is associated with
CreationDate	Datetime of the post creation
Score	Number of $Upvotes - Downvotes$ for a post
ViewCount	Times the post was viewed
Body	Text of the question or answer (HTML)
OwnerUserId	User Id of the post
LastEditorUserId	User Id of the last editor of the post
LastEditorDisplayName	User display name of the last editor of the post
LastEditDate	Datetime of the most recent edit to the post
LastActivityDate	Datetime of the last action on the post
Title	Title of a question (null if answer)
Tags	Associated tags of the question, eg. $\langle Java \rangle$, $\langle Android \rangle$, $\langle MachineLearning \rangle$, etc
AnswerCount	Number of answers for the question (null if no answers)
CommentCount	Number of comments on post
FavoriteCount	Number of times the post has been favorited
ClosedDate	Datetime when the post was closed (null if the post is open)
CommunityOwnedDate	Datetime when the post was community wikied

Table 3.1: Schema of the Posts dataset

the Posts dataset found on page 11 shows the schema of the Posts dataset, and briefly describes each attribute.

Unfortunately, deleted posts are not included in the Posts.xml file from the archive. Certain information from deleted posts can be obtained from the "PostsWithDeleted" table, which can be accessed at "http://data.stackexchange.com/stackoverflow/" [Stack-Exchange, 2015b]. However, when a post is deleted only a subset of its attributes are stored. In particular, the "body" attribute is not present and therefore information about deleted questions is not particularly useful to this dissertation. When a post is deleted, the Id, PostTypeId, ParentId, CreationDate and DeletionDate are the only attributes present. Even though the important information is missing from these posts to be part of the quality analysis themselves, some of the information provided could lead to interesting analysis. It would be interesting to understand whether a question or answer is more likely to be deleted (Number of deleted questions or answers divided by Number of questions or answers), and whether it can be predicted if a post will be deleted or not. Deleted answers have a ParentId attribute which points to the question they answered. The number of deleted answers for a question may help to infer the quality of a question.

3.1.3 Why Use this Dataset?

Within the Stack Overflow dataset is a treasure trove of technical data, with massive amounts of user generated questions and answers to learn from. The quality of the dataset is high (important attributes are not missing) due to the tireless work of moderators and administrators, although real world data will always tend to have noise. Analyzing real world data is far more interesting, and telling of how effective techniques and implementations really are. Carefully controlled data, common in research settings, will often highlight the best of the technique whereas real data will make the technique deal with things like noise and missing information. The technical text domain also adds a lot of complexity to natural language processing, as many of the existing tools were developed with social media platforms such as Twitter in mind. The Stack Overflow data set is increasingly used in machine learning research, which allows comparisons to be made about the applicability of techniques to the data. This data set shows the applicability of the methods to a specific real-world technical-domain data problem, and also to a well studied data set that allows other researchers to compare metrics and develop intuitions about the techniques.

There are many hidden questions and answers in the data. As mentioned previously in the introduction, about 73% of the questions on Stack Overflow are answered. Immediately, the question "why are 27% of the questions on Stack Overflow not answered?" appears. Other questions like "what features of questions attract users to answer them?", and "why do some questions receive more answers" This dissertation attempts to uncover and answer interesting questions such as this by delving into the data. However, the core of this dissertation is to explore the quality of posts, and how it varies in questions and answers from Stack Overflow. The rest of this section will explain how this is enabled by the Stack Overflow dataset.

Attributes such as “Score” and “ClosedDate” allows the inference of quality, even though there are limitations and issues with this (discussed in 3.1.5 Insights Into The Data). Therefore by setting varying ranges of conditions on these attributes, the continuous scale can be transformed into seperate quantised sets that represent different levels of quality. Ponzanelli et al. [2014a] states that a large amount of variance in quality is seen within a set when questions are binned to two quality levels of “bad” and “good”, so much in fact that four bins are used instead. Therefore, the approach in this dissertation is to partition both questions and answers into four quantized categorical quality levels. These categorical levels of quality are “very good”, “good”, “bad”, and “very bad” for the purposes of this dissertation. This means that there are millions of posts from which to discover what affects a question’s or answer’s quality in a technical domain. If more work is done in this area, an accurate process for automated quality analysis and similar real-world tasks could be engineered. Additionally, the techniques should generalize across multiple domains including other Stack Exchange Q&A communities.

3.1.4 Managing the Data

Since Stack Overflow is by far the most popular domain of the Stack Exchange network, its associated files are separate rather than combined into a single archive to limit size limitations as much as possible. The file with Stack Overflow’s posts data is“stackoverflow.com-Posts.7z”. Figure 3.1 on page 14 shows the 7zip listing command. The compression/encoding information, size of the archive, and the size that the files will be once inflated is shown by this command. Figure 3.2 on page 14 shows the 7zip extract command inflating the archive, extracting the uncompressed “Posts.xml” file.

The extraction took roughly 23 minutes to complete on a laptop (4th Generation i7, SSD), and the 8GB compressed archive inflated to a 40GB xml file called “Posts.xml”. This quantity of data can be difficult to handle with traditional tools on a laptop. XML is quite verbose and is not an optimal format for analytics and machine learning. This leads to the process of extracting a subset of the data and transforming it into a preferable format, sometimes known as “Extract Transform Load” (ETL). Initially, this process was performed using various Python scripts that would accomplish specific filtering or transforming tasks. This method worked ok until several scripts later there was a mess to manage and more advanced operations were taking too much engineering

```
\$ 7z l stackoverflow.com-Posts.7z
Listing archive: stackoverflow.com-Posts.7z

--
Path = stackoverflow.com-Posts.7z
Type = 7z
Method = BZip2
Solid = -
Blocks = 1
Physical Size = 8512952500
Headers Size = 122

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2016-01-04 16:27:37 ....A  42327180776   8512952378  Posts.xml
------------------- ----- ------------ ------------  ------------------------
                           42327180776   8512952378  1 files, 0 folders
```

FIGURE 3.1: 7zip listing command showing more details about the archive

```
\$ 7z e stackoverflow.com-Posts.7z

Processing archive: stackoverflow.com-Posts.7z

Extracting  Posts.xml

Everything is Ok

Size:       42327180776
Compressed: 8512952500
```

FIGURE 3.2: 7zip extract command inflates the compressed archive

time. Remembering the running order became tedious, and performing operations that were simple in databases, such as joins, were needed. Performing these operations in a dynamic (slower) language such as Python, and without indexes lead to some painful performance. This prompted the switch to using a database. Utilizing a database table with several indexes massively improved the process of managing the data.

Postgres [PostgreSQL, 2016] was identified as an ideal database due to its proved performance for large amounts of data and easy to use interfaces. While creating, populating and managing tables in a database is additional overhead, it provides flexibility and power for managing the data. Queries can be created to dynamically pull subsets of the data with various conditions in a flexible manner. Alternatively, managing the data with flat files and Python is initially low overhead (quick and dirty) but in the long run proves difficult to manage and unwieldy as the preprocessing tasks grow. Indexes are a common way to enhance database performance. An index allows the database server to find and retrieve specific rows much faster than it could do without an index, but indexes also add overhead to the database system as a whole, so they should be used sensibly. Without indexes, a simple query such as "`SELECT title FROM posts_table WHERE id = 4;`" would result in a full table scan row by row in order to get a single title from a post with id 4, which is extremely inefficient for tables as they grow and get larger. However, by creating an index like so, "`CREATE INDEX posts_table_id_index ON posts_table (id);`", Postgres can look up the index for the id rather than scanning the entire table. The reason there is overhead for indexes, is that the index needs to be updated as the table information changes, and there is also a storage overhead to the index. PostgreSQL provides several index types, each having their own strengths and weaknesses, including B-tree, Hash, GiST, SP-GiST, GIN and BRIN. B-tree and Hash indexes are two of the most commonly used indexes. B-trees can handle equality and range queries ("<", ">", "=", and combinations of) on data that can be sorted into some ordering. Constructs equivalent to combinations of these operators, such as BETWEEN and IN, can also be implemented with a B-tree index search. Also, an IS NULL or IS NOT NULL condition on an index column can be used with a B-tree index. Hash indexes can only handle simple equality comparisons, but are extremely efficient at doing so. Therefore, it is important to understand what kind of queries columns will be used in, in order to choose the correct kind of index. If a column is only used for equality conditions, than hash will be the fastest. However, if a column is used in more complex queries containing range conditions and the like, a more flexible index will be needed such as a B-tree index.

A Python script was developed in order to correctly parse the Posts.xml file and insert it into Postgres, creating the necessary indexes. This script is covered in more detail in the Implementation section starting on page 42. Once the data is loaded into the database and the indexes are built, queries will be flexible and efficient. SQL queries can be created in order to extract the quality subsets of data ("very good", "good", "bad", and "very bad" for both questions and answers) into CSV files, making them easy and efficient to read in Python using common tools. These datasets can be used as aggregated training and testing data for a supervised learning algorithm to discover what affects a question's or answer's quality. Several attributes are used to filter the

Quality	Question Rules		Answer Rules
Very Bad	$score < 0$	Closed	$score <= -2$
Bad	$score < 0$		$score = -1$
Good	$0 < score <= 6$	Accepted Answer	$0 < score <= 6$
Very Good	$7 <= score$	Accepted Answer	$7 <= score$

TABLE 3.2: Rules for Partitioning Question and Answer Quality

Quality	Question Count	Answer Count
Very Bad	28611	4958
Bad	126436	25655
Good	404099	974005
Very Good	12534	21256

TABLE 3.3: Distribution of Question and Answer Quality

posts into these four quality datasets. Table 3.2 Rules for Partitioning Question and Answer Quality on page 16 shows the unique rules for both questions and answers in order to achieve this partitioning. Posts are filtered by a number of attributes to generate these sub-datasets; "CreationDate" to extract posts from a particular timeline, "PostTypeId" to extract either questions or answers, "LastEditDate" to filter if a post has been edited, "Score" to find posts in the designated range of $Upvotes - Downvotes$ for the respective quality partition, "AcceptedAnswerId" to check if a question has an accepted answer, and "ClosedDate" to check if a post is open or closed. Table 3.3 Distribution of Question and Answer Quality on page 16 shows the distribution of the quality classes when paritioned using the rules defined in table 3.2. Edited posts are removed as the quality changes overtime, leading to an underlying movement in the quality that is measured while the score is an aggregation across all of the iterations of the post. If edited posts were included, this could lead to noisy and misleading data where the score and real quality of the post are "out of sync".

3.1.5 Insights Into The Data

When it comes to Machine Learning, there is very much a "garbage in, garbage out" mentality, in that it learns from the data and assumes it is perfectly true. If the training data is noisy and misrepresentative of what the model will be predicting, then the training data was garbage and the end result model's predictions will be garbage. Therefore it is important to understand the data involved in the machine learning well to ensure that it is suitable for the task. In this dissertation, the dataset contains real posts from

Stack Overflow. As long as correct sampling is performed, the training data will be perfectly representative of the data to be classified. However, the dataset does not actually contain a quality label for which to partition on. This can be worked around by inferring a quality label using several of the contained attributes as was mentioned in the previous section. This is not without its problems, as these rules are fairly arbitrary heuristics developed as the result of experience using Stack Overflow. Rules such as good and very good questions having accepted answers and positive scores are mostly common sense in their reasoning, and that the differentiate between a bad and very bad question is not its score but that is has been closed. Some of these rules/ranges are born out of the desire to obtain a sufficient number of samples in each category, not scientific reasoning. What this potentially leads to is the creation of noisy subsets of data, where the actual quality that an experienced user or moderator of Stack Overflow would assign to a post does not match the inferred quality of the post. Therefore, training on this will lead to incorrectly learned rules for predicting quality, leading to incorrectly classifying unseen questions and answers.

Since the “Score” attribute of posts is significantly used in order to derive a quality measure, it is important to understand it. Score is the normalized version of $upvotes - downvotes$ from the “Votes” table. It can be negative or positive, corresponding to whether the post has more upvotes or downvotes. The query listed in figure 3.3 shows that the most negative score for a post is -125, and the most positive score is 11955. The mean is 0.93 and the distribution of scores is a positively skewed normal distribution. This means that the most common scores are in the center around the score of 0, but that as the normal distribution pans out the positive direction has more scores than the negative direction. The popularity of topics, authors, and other related factors will affect the score of a post. The more attention a post has, the more viewers there are to either upvote or downvote it. This means that the more popular a post is, the more likely it is to make it into one of the two extreme quality categories. The following PostgreSQL query listed in figure 3.4 calculates the correlation of a post’s viewcount and its score for the years 2014 and 2015. The correlation returned was 0.50795, which is a significant amount of correlation in the positive direction. This means that as viewcount increases, there is a strong correlation in the increase of score. There is a dual-action effect here though, in that high-quality posts should receive more attention, but that posts with attention should receive more quality indicating votes.

```sql
SELECT score, COUNT(score)
FROM posts
WHERE
    creationdate >= timestamp '2014-01-01 00:00:00' and
    creationdate < timestamp '2016-01-01 00:00:00'
GROUP BY score
ORDER BY score
;
```

Figure 3.3: PostgreSQL query for score distribution

```sql
SELECT corr(viewcount, score)
FROM posts
WHERE
    creationdate >= timestamp '2014-01-01 00:00:00' and
    creationdate < timestamp '2016-01-01 00:00:00'
;
```

Figure 3.4: PostgreSQL query for correlation between viewcount and score

The subjective nature of quality is another particularly important influence on the analysis of this data due to the fact that putting a value on quality is open to interpretation and will vary from person to person, even between extremely experienced users or moderators (perhaps more so!). It is important to consider what quality is in different domains and communities, and therefore it is important to consider what quality is in this context. The following are some hypotheses of what might affect quality, and will be investigated further later in this dissertation. Readability is important as questions must be clear and understood in order to receive clear and correct answers. Correspondingly, answers must be understandable in order for the original poster and others to gain information from it. As Stack Overflow is a programming community Q&A, there will often be questions about code, therefore good questions and answers may involve code snippets in order to replicate or give examples. However, they should not contain whole programs as they will be difficult to follow and take considerably more time to debug. It can also annoy some in the community as if a question essentially asks for help, and then copy-pastes all their code in it is seen as being lazy and wanting someone else to do their work for them. The sentiment of a post could also be an important indication of quality, as insulting or rude behaviour will be seen negatively. The length of a post can indicate the amount of content and effort that has gone into a post, and therefore its quality.

If the datetime attributes of the dataset were timezoned, then the time of posting may indicate some interesting things about a post. However, no timezone information is included so attributes such as CreationDate do not indicate a local time of posting for the author. The time which a post is created could indicate the quality of a post. For example, a question asked at 4am local time may be strewn with errors and confusing to a reader in comparison to a question asked at 2pm. This would also serve to give some location information away about the poster, which is not ideal for privacy concerns or other political factors.

In corpus linguistics, a hapax legomenon (hapax/es), is a word that occurs only once within a context. This context could be in an entire language, in the works of an author, or in a single text. The term is sometimes incorrectly used to describe a word that occurs in just one of an author's works, even though it occurs more than once in that work. Hapaxes are quite common, as predicted by Zipf's law, which states that the frequency of any word in a corpus is inversely proportional to its rank in the frequency table. For large corpora, about 40% to 60% of the words are expected to be hapax legomena. In the fields of computational linguistics and natural language processing (NLP), especially corpus linguistics and machine learning, it is common to disregard hapaxes (and sometimes other infrequent words), as they are likely to have little value for computational techniques. This disregard has the added benefit of significantly reducing the memory use of an application, since by Zipf's law many words are hapaxes. However, the number of hapaxes is likely to be even higher in technical data like the Stack Overflow posts due to the explosion in number of nouns describing computer science related things. Topics will often have topic specific terminology, and as there are many separate topics present in the data, this will lead to many hapaxes.

3.2 Machine Learning

The research in this dissertation is based in the field of Machine Learning and Natural Language Processing. This section covers the related methodology utilized to achieve the goals of this dissertation; to learn what features are indicative of quality, and to predict the quality of questions and answers. The first example of a machine learning program was created in 1952 by Arthur Samuel. It played the board game checkers, learning the best moves to make in various scenarios from past games. Then in 1957 Frank

Rosenblatt invented the Perceptron, an example of which is shown in figure 3.6 on page 22, and in 1967 pattern recognition was implemented with a type of algorithm called the nearest neighbour. These events marked the beginning and significant early development of Artificial Intelligence and Machine Learning. The 1990's then brought the advent of a statistical data-driven approach that has led to the Big Data revolution. Machine Learning comes in various forms; Supervised, Semi-Supervised and Unsupervised. The core difference between these is whether the input data has labels to train from. For example, unsupervised learning algorithms do not take labelled input and train from it, they often are given unlabelled data to cluster into logical clusters or groups. The next section discusses supervised learning as it is utilized in this dissertation.

3.2.1 Supervised Learning

Supervised Learning methods take labelled data as input. For example, input might be a dataset containing information about people such as their age, gender, and a boolean value stating whether they bought a product. This boolean value can be used as what is known as a target label, and enables an algorithm to learn to predict the value of it for new and unlabelled information by finding patterns in the other attributes. For example, the supervised learning algorithm may learn that males under the age of 35 but over 21 are very likely to buy the product. Thus, when new and unlabelled data that states a person is 28 and male is to be predicted, the algorithm will predict that they will buy the predict. Sometimes the raw attributes will result in lacklustre predictions. Feature engineering (also known as feature extraction) can be used to create new and useful features to learn from, often greatly increasing prediction accuracy. Feature engineering is discussed further in 3.2.3 Feature Engineering and Extraction starting on page 26, however a simple example will be given here following the above context. If instead of the age of a person, the date of birth in the format of "DD/MM/YYYY" as an unstructured string was given, this would make this attribute quite difficult for an algorithm to learn from. Each combination of day, month, and year would be seen as a separate random string rather than a combined ordering. However, useful features can easily be engineered from this data. The most obvious feature is that the age can be calculated from this date of birth attribute and used in the training process. Once all the desired features are present, the data must be transformed into a machine understandable representation, as the learning algorithm won't understand the significance of things like words. More on data representation is covered in 3.2.2 Data Representation on page 22. Once the data is in an appropriate representation, a supervised learning algorithm will be able to take the labelled data as input and learn how to predict a chosen target variable. Once the training has occurred and generated a model, often referred to as fitting the model, it can be used to make predictions on unlabelled data. This process is shown in figure 3.5 on page 21.

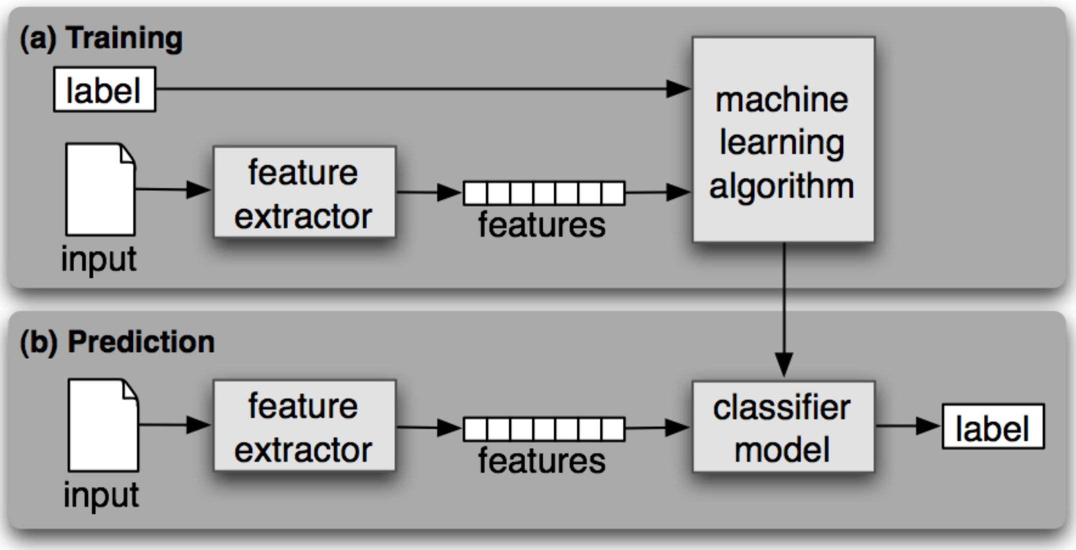


FIGURE 3.5: The supervised learning process. [NLTK, 2015]

Classification is a more specific type of prediction, which is essentially the task of predicting one or more classes/categories rather than a continuous number, which is known as regression. This dissertation focuses on the classification of text documents into four categories of quality, also known as labels or classes. Classification of text documents is also sometimes coined as document classification. The documents to be classified may be text documents, images, music or many others, but when the type of document classification is not explicitly stated then text classification is generally implied. Supervised document classification is when examples of the correct classification for documents are given to train on. Multiclass classification is when multiple classes can be assigned in the same prediction, but are not used in this dissertation. An example of when multiclass classification is useful is in topic extraction. Documents can have multiple topics within them, so limiting the prediction to labelling it with a single topic class would immediately be incorrect. Predicting a single class is often referred to as hard classification. When classification can predict levels of membership to multiple classes it is often referred to as soft classification. Figure 3.6 on page 22 shows a perceptron updating its

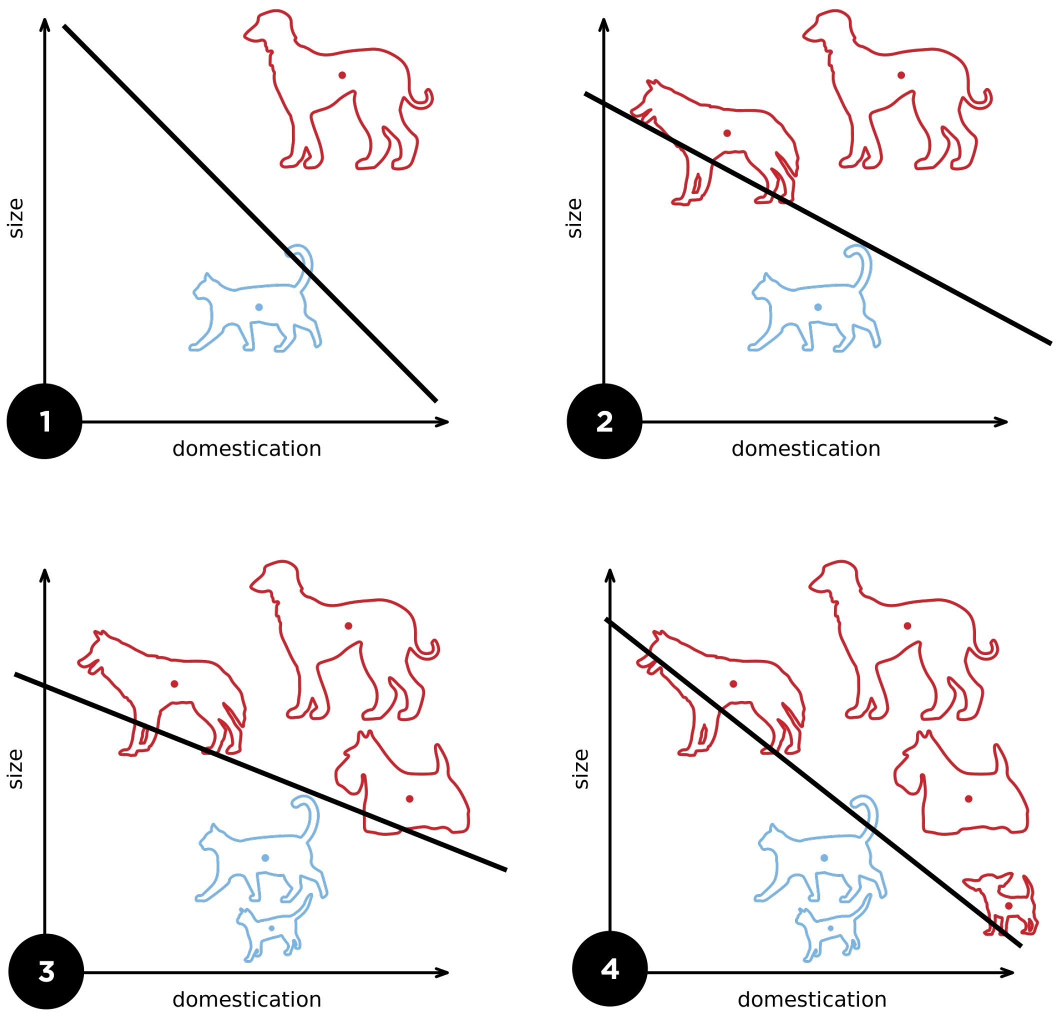


FIGURE 3.6: Perceptron updating its linear boundary as training examples are added Goodspeed [2015]

linear boundary as training examples are added as part of a classification of dogs or cats. It learns how to weight the attributes "size" and "domestication" in order to classify whether a new unlabelled animal is a dog or cat.

3.2.2 Data Representation

In order for algorithms to learn from data and make prediction on data, it must be in a machine understandable representation. Feature vectors are a common way to structure input to a machine learning algorithm. Feature vectors are an n-dimensional vector of numerical features that represent an object. Machines do not understand human concepts by default. Machines will not understand the significance of words or what they mean in the same way a native speaker would. Machines are not be able to look at an image and identify objects from it as people do. Therefore this information must be encoded for a machine in a representation that it understands, and that representation is in numbers, binary to be precise. Obviously, we do not look at things like paintings as an array of numbers, but that is exactly how it must be (and is) represented for a computer, with each pixel of an image represented by a number. Feature names (which are most likely initially encoded as strings) are mapped to indices. Take the example from the previous section, the feature names (dog, cat) will be represented as the numbers (1, 2). For inspection purposes when performing tasks such as visualization, these indices can be mapped back to their string representation. For the feature values themselves, there are two main types of data that need to be represented; numerical and categorical. This section also covers how text data is in turn represented in the form of numerical and categorical features.

Numerical data is already represented in a format amenable to machine learning, except for performing the mapping of its feature names. Although, similarly to other types of data or perhaps even more so, performing some preprocessing or normalization on the raw numerical data will lead to far better results.

There are two types of categorical variables; nominal and ordinal. A nominal categorical variable has no intrinsic ordering to the categories, and tends to be the default type when talking about categorical data in general. For example, hair color is a categorical variable having a number of categories (blonde, brown, brunette, red, etc), and there is no inherent ordering. An ordinal categorical variable has a clear ordering of the categories. An example of this is a mapping of temperatures to labels such as very hot, hot, cold, and very cold. Even though these can can order these from lowest to highest, the spacing between the values may not be the same across the levels of the variables.

The quality categories (very good, good, bad, very bad) used in this dissertation are also an example of of a categorical variable, or more specifically as an ordinal categorical variable, but for the purposes of this dissertation this distinction should not make too much of a difference. Categorical information does not have an obvious numerical representation, as categories are a list of unique labels that each represent something. This something could be a range of numbers, a colour, the size label on clothing, a species of animal, a type of flower, etc. For instance, three labels in the category of animals could be [dog, cat, horse].

A common method for turning categorical information into useful features for machine learning is called One-Hot encoding. One-Hot encoding creates a new boolean feature (value of the feature is either 1 or 0) for each distinct category label, if a specific category label is present than its feature value is 1, 0 if not. Taking the previous example, this means that 3 new features would be created with either the value of 1 or 0 to represent the labels dog, cat and horse. There are several other types of categorical encoding used, although somewhat less common. [McGinnis, 2015] Ordinal coding assigns an integer to each category, creating a category-integer mapping similar to the feature name indices mapping mentioned above. Ordinal coding implies an order to the variable that may or may not actually exist. Binary encodes the categories as ordinal, but then converts the integers into binary code and performs one-hot encoding on the digits. This encodes the data in fewer dimensions than one-hot but with some distortion of distance. Other methods such as Sum and Helmert compare means of the dependent variable for the different categories of levels, but they are not used in this dissertation.

In order to use machine learning algorithms on text data, it must first be transformed into a machine understandable representation. The bag-of-words model is a popular simplifying representation that represents text as the multi-set of its words, disregarding grammar and word order but maintaining multiplicity. It is essentially a term-frequency vector with its dimensions equal to the number of unique words in the data, each entry the number of occurrences of a particular word in a document. It embodies the essentialism and reductionist view of text data in that it reduces a text document down to the individual objects that represent it, and does so in an isolated manner without any concept of relationship between objects. The bag-of-words model is commonly used in methods of document classification, where the occurrence of each word is used as a feature for training a classifier. This occurrence can be represented in multiple ways, in that it could either exist or not exist, or could be a form of the number of times it occurred. The bag-of-words approach is not the only representation, there are other representations that try to model similarity-based information and other kinds of relationships from the data. Text documents are commonly represented by the bag-of-words model, however not all of the words are discriminative or characteristic which essentially becomes noise. Documents can be represented using a richer feature set of term frequencies, named entities and term pairs. Term Frequencies is the complete vocabulary set of the document corpus after the stop-words removal and words stemming operations. Named entities includes names of people, organizations, locations, etc. Term pairs takes only only those term associations which have statistical significance for the document corpus to maintain a compact feature set. Word co-occurrence matrices can also be used to describe how words occur together. This captures the relationships between words, which is powerful when trying to identify relationships between documents that contain these words. Many techniques can be used to extract and infer interesting information from a piece of text. For example a useful weighted bag-of-words feature representation can be generated by transforming a bag-of-words using term frequency inverse document frequency (TF-IDF). TF-IDF weights the values based on the product of each word occurrence in each text (term frequency) and the inverse of the word occurrences across texts (inverse document frequency). TF-IDF finds the words that have a more significant effect on what category the document is in, and also removes the issue of uneven document length and frequency of categories in the training sets. The term frequency (TF) component of TF-IDF is a measure of the frequency of a term in a document. [Luhn, 1957] The simplest term frequency $tf(t, d)$ is the raw frequency of a term in a document, which is the number of times that term t occurs in document d. If we denote the raw frequency of t by $f_(t, d)$, then the simple term frequency scheme is $tf(t, d) = f_(t, d)$, where tf is term frequency. Other measures of frequency include include boolean, log scaled, and augmented frequency. Boolean is when $tf(t, d) = 1$ if t occurs in d and $tf(t, d) = 0$ otherwise. Log scaled $tf(t, d) = 1 + \log(f_(t, d))$, or zero if $f_(t, d) = 0$. Augmented frequency is when the raw frequency is divided by the maximum raw frequency in the document, such that $\mathrm{tf}(t, d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d}: t' \in d\}}$. The inverse document frequency (IDF) component is a measure of whether the term is common or rare across all documents. [Sparck Jones, 1988] It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient. The equation for this is $\mathrm{idf}(t, D) = \log \frac{N}{1+|\{d \in D: t \in d\}|}$ where N is the total number of documents in the corpus $N = |D|$, $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears. Combining these two components gives term frequency inverse document frequency (TF-IDF) is calculated as $\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$. A high value of TF-IDF is reached by a high term frequency and a low document frequency of the term in the whole collection of documents. The TF-IDF function filters out very common words due to the IDF's log function. As a term appears in more documents the ratio inside the log will approach 1, causing the IDF to approach 0. TF-IDF is a dot product between TF and IDF, resulting in the same. Character N-Grams is like the bag-of-words representation, but instead can be viewed as a bag-of-characters representation. [Cavnar et al., 1994] Instead of splitting on word boundaries and using them (or n-grams of them) as features, it uses n-character splits as features. The primary advantage of this approach is that it is ideally suited for text coming from noisy sources, such as user generated content with many non-natural language aspects to it such as technical data with code and markup, as is in Stack Overflow posts. This dissertation makes heavy use of Character N-Grams as they prove to be very effective, and are discussed further in the next section, Feature Engineering and Extraction.

3.2.3 Feature Engineering and Extraction

An important part of the process of supervised learning is feature engineering. Feature engineering uses domain knowledge of the data to generate new features that are not part of the raw data. Feature engineering often improves the accuracy of the classification, however it is both difficult and expensive in terms of man-hours, or compute cycles if automated in some way. There is also a danger of over-engineering and over-fitting a model through excessive feature engineering. A simple example of an engineered feature would be calculating the count of words per sentence from a piece of raw text. It is important to use features that accurately predict the quality of a question or an answer to make accurate predictions. Table 3.4 on page 27 shows the features that were extracted in order to classify quality in this dissertation and a description of each. The rest of this section discusses how some of these features are extracted focusing on text similarity, sentiment analysis, and readability indexes.

The text similarity measures between two pieces of text, whether it be title-body or body-body similarity, use Term Frequency Inverse Document Frequency (TF-IDF) to create the term-frequency vectors, and cosine similarity to calculate a similarity between the two vectors. TF-IDF has been discussed already, when explaining how to represent textual data. Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.. The cosine

Feature	Description
Body Length	Count of characters of the post (HTML)
Spelling Error Count	Count of spelling errors in body
Email Count	Count of e-mail addresses in post
Url Count	Count of URLs in the post
Uppercase Percentage	Percentage of uppercase characters
Lowercase Percentage	Percentage of lowercase characters
Spaces Count	Count of spaces
ARI	$4.71 * \frac{characters}{words} + 0.5 * \frac{words}{sentences} - 21.43$
Flesch Reading Ease	$206.835(1.015 * \frac{words}{sentences})(84.6 * \frac{syllables}{words})$
Flesch-Kincaid Grade	$(0.39 * \frac{words}{sentences}) + (11.8 * \frac{syllables}{words}) - 15.59$
Gunning Fog Index	$0.4(\frac{words}{sentences} + PercentageComplexWords)$
SMOG Index	$\sqrt{(complexwords * \frac{30}{sentences})} + 3$
Coleman Liau Index	$(0.0588 * \frac{characters}{words}) - (0.296 * \frac{words}{sentences}) - 15.8$
LIX	$\frac{words}{periods} + \frac{longwords*100}{words}$
RIX	$\frac{longwords}{sentences}$
Sentiment	Polarity [-1.0, 1.0], [Negative, Positive]
Subjectivity	[0.0, 1.0], [Objective, Subjective]
Lines of Code	Count of lines between tags $\langle code \rangle$
Code Percentage	Percentage of a posts lines that are code
Number of Code Tags	Count of $\langle code \rangle$ tags
Number of P Tags	Count of $\langle p \rangle$ tags
Character N-Grams	"the" $\rightarrow$ ["t", "h", "e", "th", "he", "the"]
Title Length	Count of characters in the title of a question
Title-Body Similarity	Similarity of a question's title and body
Is Title Capitalized	True if the title begins with a capital letter
Q&A Body Similarity	Similarity of an answer's and its associated question's bodies

Table 3.4: Engineered Features Used for Predicting Quality

similarity of two documents ranges from 0 to 1, since the term frequencies (TF-IDF weights) cannot be negative.

Readability measures are ways of predicting the complexity of a piece of text. Research has shown that the two main factors that affect the difficulty of reading and comprehending text are lexical and syntactical difficulty. [Klare, 2000] Lexical difficulty refers to how difficult or rare the words are, as words that are simpler or short are often more easily understood. Lexical difficulty is often predicted using word length in characters or syllable count, but sometimes also using word lists mapping words to their difficulty. The former methods have an obvious weakness, in that not all long words or polysyllabic words are difficult. Syntactical difficulty refers to how difficult the sentences are, as longer, more complicates sentences will be less comprehensible than short and simple sentences. Syntactical difficulty can be predicted using measures such as the average number of words (or hard/easy words) per sentence. These formulas are often validated by comparing the consistency of their predictions with each other, or comparing them with an outside measure of readability such as an experts opinion (an experienced teacher) on a texts readability level.

Many readability indexes approximate a U.S. grade level to understand the text. A breakdown of grade levels in the U.S with expected ages is shown in table 3.5 on page 29. Flesch Reading Ease is one of the few which does not do this, but even its output can be converted into an approximate grade using a simple table as shown in table 3.6 on page 30. There are many factors that affect how the output of a readability formula is interpreted. The most important of these is who the target audience is. Specifically, what age and education level, what is the text domain, and what is the purpose of the reader. For example, a piece of text designed to help a child learn to read should be very simple. Conversely, a Masters student who wishes to further their knowledge in Machine Learning may want a text filled with lots of useful information and domain specific language, which naturally will score a high complexity prediction from readability indexes. In the former example, a piece of text that is predicted as simple may not be appropriate for the Masters student in this particular situation, but a simpler blog post on the same topic might be at another time or for a more casual learner. Therefore, whether the text is complex or simple is not necessarily always bad or good, but depends on the context. The context of Stack Overflow will vary from question to question, but complex questions and answers can be expected due to it be

Age (Years)	Education level
5-6	Kindergarten
6-7	First Grade
7-8	Second Grade
8-9	Third Grade
9-10	Fourth Grade
10-11	Fifth Grade
11-12	Sixth Grade
12-13	Seventh Grade
13-14	Eighth Grade
14-15	Ninth Grade
15-16	Tenth Grade
16-17	Eleventh grade
17-18	Twelfth grade
18-22	College

Table 3.5: Expected Age for Each US Grade Level

a Q&A website for programmers on domain specific topics. Next, some examples of readability indexes used in this dissertation will be discussed.

The Automated Readability Index (ARI) was published by E. A. Smith and R. Senter in November 1967 [Senter and Smith, 1967]. The research was done in association with Aerospace Medical Research Laboratories of the US Airforce. The ARI was devised to enable measures of readability to be calculated as the text was typed. Impulses from a typewriter would activate counters which record the number of letters, words and sentences contained in the passage. From this, the average word length and average sentence length are calculated. These variables can be used to compute a readability index that reflects the difficulty of the text. The formula for the Automated Readability Index is $4.71 * \frac{characters}{words} + 0.5 * \frac{words}{sentences} - 21.43$, where $\frac{characters}{words}$ calculates the average number of characters per word and $\frac{words}{sentences}$ calculates the average number of words per sentence. The advantage of relying on the number of characters rather than syllables is that they are far easier to count for computers. Like many other readability indexes, ARI outputs an estimate of the grade level that the text would be appropriate for.

Flesch Reading Ease is considered as one of the oldest and most accurate readability formulas. Rudolph Flesch developed this formula in 1948. The Flesch Reading Ease

Flesch Reading Ease	Education level	Difficulty
90-100	5th Grade	Very Easy
80-89	6th Grade	Easy
70-79	7th Grade	Fairly Easy
60-69	8th and 9th Grade	Standard
50-59	10th to 12th Grade	Fairly Difficult
30-49	College	Difficult
0-29	College Graduate	Very Confusing

TABLE 3.6: Flesch Reading Ease Mapping

Formula is a simple approach to assess the grade-level of the reader. This formula is best used on school text, but is used by many US Government Agencies, including the US Department of Defense. [Kincaid et al., 1975] Flesch Reading Ease can be calculated using the formula $206.835(1.015*\frac{words}{sentences})(84.6*\frac{syllables}{words})$. $\frac{words}{sentences}$ calculates the average sentence length, and $\frac{syllables}{words}$ calculates the average number of syllables per word. Flesch Reading Ease outputs a number between 0 and 100, the higher the number the easier the text is to read. The table 3.6 on page 30 contains a mapping of output to difficulty labels.

Flesch-Kincaid Grade improves upon the Flesch Reading Ease Readability Formula. Rudolph Flesch and John P. Kincaid are co-authors. In 1976, the US Navy modified the Reading Ease formula to produce a grade-level score by applying the Flesch Grade-Scale formula, or the Kincaid formula. The formula for the Flesch-Kincaid Grade is $(0.39 * \frac{words}{sentences}) + (11.8 * \frac{syllables}{words}) - 15.59$, where $\frac{words}{sentences}$ calculates the average length of sentences and $\frac{syllables}{words}$ calculates the average number of syllables per word. Like other grade level based readability indexes, a score of x means that the text is suitable reading for a student in year x of grade-school in the American school system. Theoretically, the lowest grade level score could be -3.4, but this is highly unlikely in practice as there are no real passages where every sentence consists of a one-syllable word.

The Gunning Fog Index was developed by Robert Gunning, who was a graduate from Ohio State University. Gunning published the formula in a book in 1952 called “The Technique of Clear Writing”. [Gunning, 1952] Gunning observed that most high school graduates were unable to read. His opinion was that the problem primarily stemmed from a writing problem rather than a reading problem, that newspapers and business documents were full of "fog" and unnecessary complexity. The core principle of the Gunning Fox Index is that sentences should be short and simple, rather than long and complicated sentences. The formula for Gunning Fox Index is $0.4(\frac{words}{sentences} + PercentageComplexWords)$, where $\frac{words}{sentences}$ calculates the average words per sentence, and the percentage of complex words is the percentage of words that are made up of three syllables or more but are not proper nouns, not combinations of easy words or hyphenated words, or two-syllable verbs made into three with -es and -ed endings. However, this can be seen as one major flaw with the Gunning Fox Index as it assumes that all multi-syllabic words are difficult, which is not necessarily true.

The SMOG Index was developed by G Harry McLaughlin in 1969 in an article, "SMOG Grading A New Readability Formula in the Journal of Reading". [Mc Laughlin, 1969] The formula for the SMOG Index is $\sqrt{(complexwords * \frac{30}{sentences}) + 3}$, where complex words is the count of words with three or more syllables.

The ColemanLiau Index was developed by linguists Meri Coleman and T. L. Liau. [Coleman and Liau, 1975] The formula was to help the U.S. Office of Education calibrate the readability of all textbooks for the public school system. Similarly to the Automated Readability Index, but unlike most of the other grade-level predictors, the ColemanLiau relies on characters instead of syllables per word. Instead of using syllable/word and sentence length indices, Meri Coleman and T. L. Liau understood that counting characters is far simpler for computers than counting syllables or sentence length. According to Coleman, "There is no need to estimate syllables since word length in letters is a better predictor of readability than word length in syllables.". The formula for the Coleman Liau Index is $(0.0588 * \frac{characters}{words}) - (0.296 * \frac{words}{sentences}) - 15.8$, where $\frac{characters}{words}$ calculates the average number of characters per word and $\frac{words}{sentences}$ calculates the average number of words per sentence.

The "Lasbarhetsindex" (LIX) is a Swedish readability measure developed by Swedish scholar Carl-Hugo Bjrnsson to calculate the difficulty of text in Swedish comprehensive school. However, it has often proven to work across many foreign texts leading it to be used on languages such as English. The formula for LIX is LIX $\frac{words}{periods} + \frac{longwords*100}{words}$, where $\frac{words}{periods}$ calculates the average number of words per period, colon or capital first letter, and $\frac{longwords*100}{words}$ calculates the percentage of long words out of all the words,

LIX	Text Difficulty
20	Very Easy
30	Easy
40	Medium
50	Difficult
60	Very Difficult

TABLE 3.7: LIX Norms for Swedish for interpreting text difficulty

where long words are considered words of more than 6 letters. The equal weighting of these variables contributes to its computational efficiency, and the calculation should be consistent as long words are calculated by character length, not syllables. It is potentially this reason that it bypasses the problem of counting syllables, that it works well across multiple languages. Bjrnsson created a table of norms for Swedish to interpret text difficulty from Lix scores, this table 3.7 can be found on page 32. RIX is another readability index, and merely simplifies the LIX formula to $\frac{longwords}{sentences}$, which calculates the average number of long words per sentence, where long words are words of more than 6 letters as per LIX.

3.2.4 Preprocessing

Preprocessing is an important step in the data mining process. The phrase "garbage in, garbage out" holds true in many machine learning processes. Data often contains issues such as out-of-range values, impossible data combinations, missing values, and too much variance. Depending on the algorithm used, these issues could cause wildly inaccurate and therefore misleading results. If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. Preprocessing includes cleaning, normalization, and transformation, of the raw data.

Feature scaling is a method used to standardize the range of independent variables or features of data and is also known as data normalization. One method of feature scaling is to scale the features to be within a minimum and maximum value, this is called min-max scaling. The equation for min-max scaling is $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$. Another similar method is to scale features so that the absolute values are less than a maximum value, which is known as max absolute scaling.

In computational linguistics, lemmatisation is the algorithmic process of determining the lemma for a given word. A lemma is the base form of a word, for example words such as “written”, “writing”, and “writes” can all be lemmatized to “write”. Since the process may involve complex tasks such as understanding context and determining the part of speech of a word in a sentence it can be a hard task to implement a lemmatiser. A simpler and similar technique is stemming, the difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications. For example, a stemmer might have simple rules to remove word endings such as “ing”, “ed”, or “s”.

3.2.5 Training and Testing Strategies

Once the data has been preprocessed and is in the form of feature vectors, it can be used to train and test supervised machine learning models. Training and testing data must be kept separate, as the test data is supposed to be completely new and unseen when the prediction accuracy of the learning model is tested. If there is data leakage of the test data into the learning process, the performance metrics will be artificially boosted. There are many different training and testing strategies that can be used to accomplish this.

Splitting the data into a train-test split in a specific ratio, such as 70%/30%, is the simplest of these strategies. This separates the data so that the training phase learns from the learning data alone, and not the testing data. This means that the test data is completely unseen, and therefore provides a realistic test of how the model will perform on real-world new and unseen data.

Cross-validation is a more robust model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. K-fold cross-validation randomly partitions the data into k equal sized folds. Of the k folds, a single fold is retained as the validation data for testing the model, and the remaining $k - 1$ folds are used as training data. The cross-validation process is then repeated k times across all of the different unique permutations of the folds. The k results from the folds can then be averaged to produce a single estimation. The advantage of this method over the traditional train-test split method is that all observations are used for both training and validation, and each observation is used for validation exactly once. The process of rounds reduces variability and therefore obtains a more reliable performance metric. Stratified k-fold cross-validation is a slight modification, in that the folds are selected so that, in the case of classification, each fold contains roughly the same proportion of each class label. This means the samples from a balanced dataset will also be balanced, which is ideal for prediction problems.

3.2.6 Decision Trees

Decision Trees (sometimes known as Classification Trees when used for classification) are attractive because amongst other data mining methods, decision trees have various advantages. They are simple to understand as they can quite easily visualized and graphed, to the extent that they are considered a white-box model meaning that the results are easily explained due to the inherent boolean logic. They require little data preprocessing such as normalisation and are efficient and robust in use. In fact, tree based learning algorithms are very unique in the trait of being scale-invariant. Pruning is a technique that can be used to fix inherent problems such as overfitting, not many algorithms have an easy post-processing fix for this. Figure 3.7 illustrates a decision tree that decides whether someone will buy a product based on age, education status, and credit rating. They have been studied extensively in the past several decades, and are used in practical applications. [Quinlan, 1986] [Ho, 1995] Decision Trees are trees in which non-leaf nodes are labeled with the input features, the arcs coming from a node labeled with a feature are labeled with each of the possible values of the feature, and each leaf of the tree is labeled with a class or a probability distribution over the classes. Learning occurs by splitting the source set into subsets based on an attribute value. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees (TD-IDT) is an example of a greedy

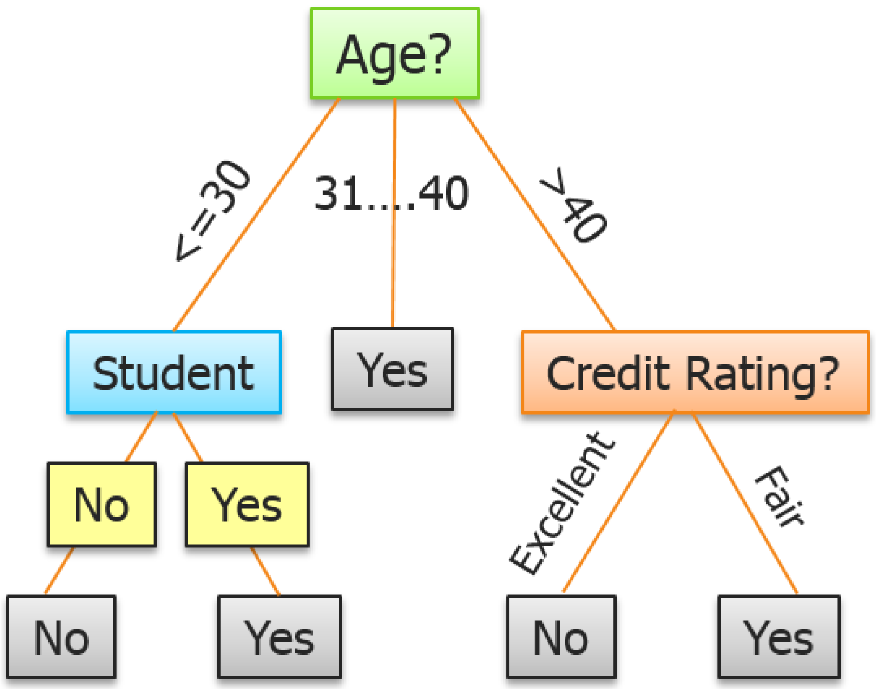


FIGURE 3.7: Illustration of a Decision Tree - “Will they Buy The Product?”

algorithm, and it is by far the most common strategy for learning decision trees from data.

Node impurity is a common method for selecting splits in a Classification Tree. It is calculated by looking at the total decrease in node impurities from splitting the variables averaged over all the trees, and uses Gini impurity. However, this method is biased towards variables with more categories. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. It reaches its minimum (zero) when all cases in the node fall into a single target category. To compute Gini impurity for a set of m items, suppose $i \in \{1, 2, ..., m\}$, and let f_i be the fraction of items labeled with value i in the set. Gini impurity can be computed by summing the probability f_i of each item being chosen times the probability $1 - f_i$ of a mistake in categorizing that item, $I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i)$.

As a result of using this method of split selection, decision trees have the ability to estimate how important features are in the classification process. The relative rank (depth) and frequency of a feature in a tree can be used to estimate feature importance with respect to the predictability of the target variable, which in this dissertation is quality. Features at the top of the tree split a larger fraction of the input samples than features further down the tree, and therefore can be seen as more important to the prediction than the lower splits. Every time a split of a node is made on variable m, the Gini impurity criterion for the two descendant nodes is less than the parent node. Adding up the Gini decreases for each individual variable over all trees in the forest gives a fast variable importance that is often very consistent with the permutation importance measure.

3.2.7 Random Forests

The general method of Random Forests was first proposed by Ho [1995]. Decision trees are attractive classifiers for several reasons, in particular their high execution speed. But trees derived with traditional methods often cannot be grown to arbitrary complexity for possible loss of generalization accuracy on unseen data. The limitation on complexity usually means suboptimal accuracy on training data. Essentially, the method is to build multiple trees in randomly selected subspaces of the feature space. Trees in, different subspaces generalize their classification in complementary ways, and their combined classification can be monotonically improved. Therefore, a Random Forest classifier uses a number of decision trees in order to improve the classification rate.

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges as such to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields favorable error rates that are robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting. Internal estimates are also used to measure variable importance. If Gini node impurity is used across all of the trees in the Random Forest, the feature importance estimate that can be calculated from this will become more reliable as variance is reduced, even more so as the number of trees is increased. These ideas are also applicable to regression. [Breiman, 2001] A visualized example of a Random Forest is shown in figure 3.8 on page 37.

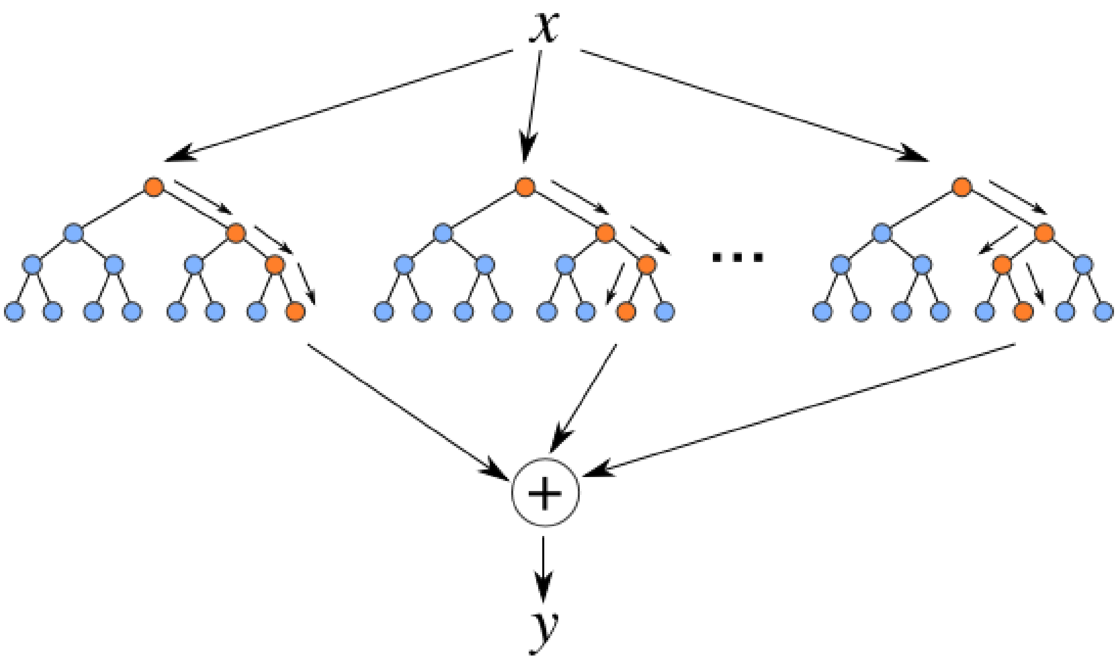


FIGURE 3.8: Visualized Example of a Random Forest

3.3 Methods of Evaluation

This section covers methods that be used to evaluate the implementations in order to compare how they perform with each other and other implementations. Computational complexity is discussed as an important validation to ensure that an implementation is tractable. Then performance measures for machine learning algorithms are discussed, these are important to discover the optimal techniques to solve the problem of predicting question and answer quality.

3.3.1 Performance Measures

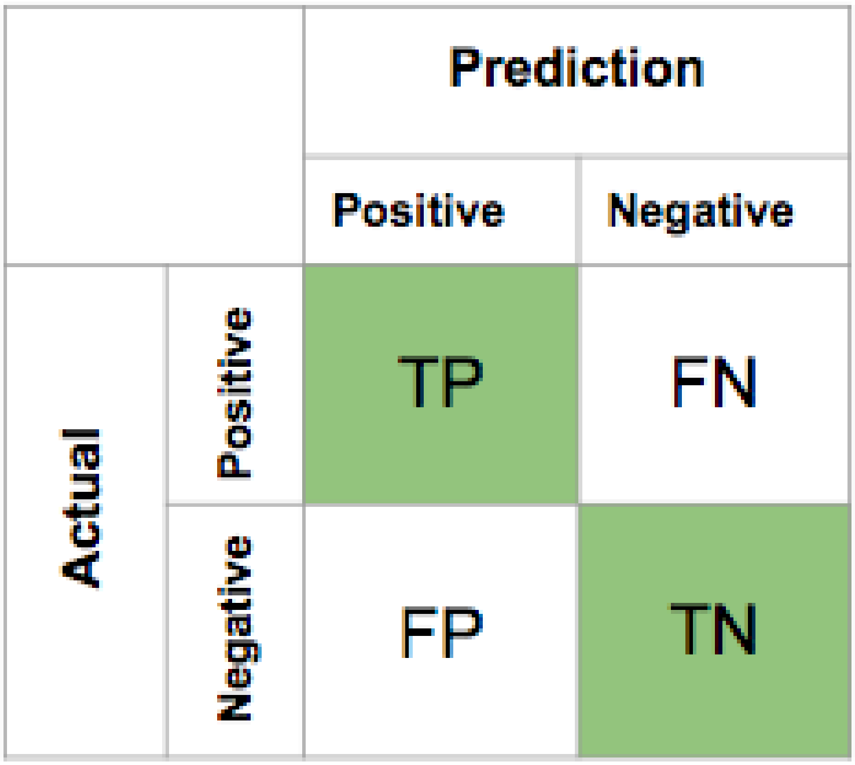
It is important to define how the performance of the algorithms used will be measured so that empirical results can be recorded and compared. There are many metrics to choose from, each with their own advantages and disadvantages. Additionally, some are more appropriate for different types of machine learning. This dissertation focuses on the classification branch of machine learning, and therefore metrics for classification will be focused on.

Many classification metrics are functions of four base metrics; True Positive, False Positive, True Negative, and False Negative. These metrics can be explained using an example of a binary classification with classes True and False. True positives are when True is predicted and True is the actual class. False positives are when True is predicted but False is the actual class. True negatives are when False is predicted and False is the actual class. False negatives are when False is predicted but True is the actual class.

Recall, precision and F_1 score are commonly used metrics that build upon these base metrics. Recall is the fraction of the documents that are relevant to the query that are successfully retrieved. The formula for recall is $\frac{Relevant \cap Retrieved}{Relevant}$, or $\frac{TP}{TP+FN}$. However, merely by returning every document a recall of 100% can be achieved. Therefore, it is useful to also calculate the number of non-relevant documents returnd. Precision is the fraction of retrieved documents that are relevant to the query. The formula for precision is $\frac{Relevant \cap Retrieved}{Retrieved}$ or $\frac{TP}{TP+FP}$. F_1 score combines recall and precision as the square of the geometric mean divided by the arithmetic mean. The formula for F_1 score is $2 * \frac{Precision*Recall}{Precision+Recall}$ or $2 * \frac{TP}{2TP+FP+FN}$. When recall and precision are close, it is approximately an average of two, therefore roughly an equal combination of them. F_1 score is a special case of the general F_β measure F_β measure $F_\beta = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$. Two other commonly used F measures are the F_2 measure, which weights recall higher than precision, and the $F_{0.5}$ measure, which puts more emphasis on precision than recall. As suggested by this, recall and precision tend to be considered as a trade off.

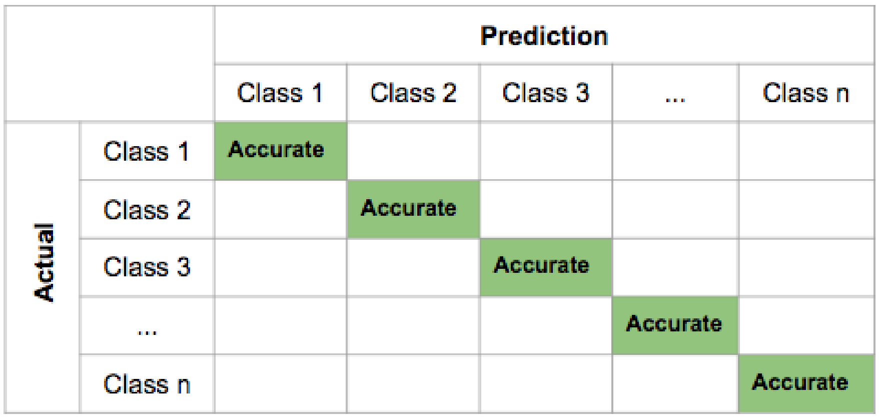
A confusion matrix is a table layout that visualises the performance of a classification model by displaying the actual and predicted points in a nxn grid, where n is the number of classes in the classification. The X-axis is the label that the model predicted, and the Y-axis is the true label. Confusion matrices use either a count in or colour intensity of each square to indicate the number of percentage of samples predicted in a grid. The confusion matrix for a binary ($n = 2$) classification is shown in figure 3.9 on page 39. The confusion matrix for a multiclass classification is shown in figure 3.10 on page 39. [WSO2, 2016] Confusion matrices allows which points are correctly classified and which points are incorrectly classified to be identified. The points in the grids with matching actual and predicted classes are the correct predictions. The green grids in the previously mentioned figures are the grids of correctly classified points. Ideally, all other grids should have zero points.

Feature Importances of the random forest can be visualized using histograms. In the



		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

FIGURE 3.9: Example of Binary Classification Confusion Matrix



		Prediction				
		Class 1	Class 2	Class 3	...	Class n
Actual	Class 1	Accurate				
	Class 2		Accurate			
	Class 3			Accurate		
	...				Accurate	
	Class n					Accurate

FIGURE 3.10: Example of Multiclass Classification Confusion Matrix

results chapter, feature importances diagrams have the name of each feature listed along the Y-axis, the importance of feature is on the X-axis, and all of the feature importances sum to 1. It provides an intuitive visualization of what the most important features are when predicting quality of questions and answers.

Bradley [1997] discusses evaluating machine learning algorithms using ROC (Receiver Operating Characteristic). ROC curve is commonly used for the evaluation of supervised machine learning algorithms. The Area Under the ROC Curve (AUC) is commonly used as a measure of classifier performance. AUC exhibits a number of desirable properties when compared to overall accuracy; increased sensitivity in Analysis of Variance (ANOVA) tests, a standard error that decreases as both AUC and the number of test samples increases, decision threshold independent invariant to a priori class probabilities, and it gives an indication of the amount of work done by a classification scheme giving low scores to both random and one class only classifiers. AUC actually represents the probability that a randomly chosen positive example is correctly rated ranked with greater suspicion than a randomly chosen negative example. The paper concludes with the recommendation that AUC be used in preference to overall accuracy when single number evaluation of machine learning algorithms is required.

3.3.2 Computational Complexity

Efficiency of these algorithms is critical due to the tendency to be performed in conjunction with large amounts of data. Understanding the resource needs of the algorithm when it comes to CPU and memory in particular is important, and also how long it takes to run the algorithm. An algorithm could be the most accurate method there is to cluster data, but if it is intractable to run it on anything bigger than a toy data set than it will never be used. Garey and Johnson [1979] discusses the theory of NP-Completeness, and a distinction is made between two particular causes of intractability. The first is that the problem is so difficult that the solution is exponential, and the second is that the solution is required to be so extensive that it cannot be described with an expression having length bounded by a polynomial function of the input length. This second type of intractability can be regarded as a signal that the problem is not defined realistically, because we are asking for more information than we could ever hope to use. It is the first kind of intractability here that is often faced when analysing text data, as this is inherently difficult. For example, graph based methods are NP-Complete problems, and are common in Machine Learning. Many classical machine learning algorithms, such as Support Vector Machines (SVM) and Logistic Regression, assume simpler models to make them tractable (ie. that they run in polynomial time, $O(p(n))$ for some polynomial function p, where n is used to denote the input length). Random Forests were chosen because they can map some interesting and complex relations, while remaining computationally tractable.

3.3.3 Statistical Significance in Machine Learning

Yeh [2000] discusses statistical significance testing for metrics like recall, precision and balanced F-score. This process is a necessary part of empirical machine learning and natural language processing. Unfortunately, many commonly used tests often underestimate the significance and so are less likely to detect differences that exist between different techniques. This underestimation comes from an independence assumption that is often violated. Thankfully, there are some useful tests that do not make this assumption, including computationally-intensive randomization tests that are utilized in this dissertation.

Ojala and Garriga [2010] proposed a test they call "test 1". This tests whether the classifier has found a significant class structure, or in other words a real connection between the data and the class labels. Test 1 does this by permuting the target labels of the data, training and testing on different folds many times randomly. This should simulate a "coin toss" prediction for the target label, which can then be compared with the engineered solution to calculate statistical significance.

Chapter 4

Implementation

This section details how the above methodology can be implemented in order to accomplish the goals of this dissertation; predicting the quality of questions and answers, and gaining insight into indicative features of quality. There are two main functional components to the implementation of this dissertation; data management and machine learning. The data management component involves obtaining, storing, processing, and interacting with the data. Much of these tasks were discussed in the Methodology section, but more detailed implementation aspects of these tasks are discussed here. This involves using the Postgres database and how the data is parsed into it from the archived XML file with the Python program "PostsXML2Postgres.py". The more specific aspects of the SQL queries used in this dissertation are also covered. The machine learning component uses the quality datasets that are created using the data management component of the dissertation, and outputs results of how successful the classification was and insights such as which features were most important for doing so. There are three main programs that are used in the Machine Learning component; "QuestionQualityAnalysis.py", "AnswerQualityAnalysis.py", and "StackOverflowTextAnalysis.py". Several other Python programs were written in order to do complementary analysis, such as calculating the feature distribution statistics across questions and answers, or calculating word counts across posts in order to find hapaxes.

The source code for this dissertation can be found in the following Github repository:

- https://github.com/ghodgins/stackoverflow-quality-analysis [Hodgins, 2015b]

4.1 Data Management

The first task is obtaining the dataset itself, and was explained in the Methodology section. The Stack Overflow dataset can be found on the Internet Archive's website. [StackExchange, 2015c] Once downloaded, the archive can be inflated to obtain the "Posts.xml" file, which is a 40GB file containing millions of posts in XML format. This is not particularly easy to work with, as it is a huge file in a less than ideal format. A Python script was developed in order to correctly parse the Posts.xml file and insert it into a database, also creating indexes to make queries efficient and flexible. PostgreSQL was chosen as the database because it is a powerful and open source database that has more than 15 years of active development, a proven architecture, and simple to use. Postgres has earned a strong reputation for reliability, data integrity, and correctness in both academia and industry. It is also highly scalable in the sheer quantity of data it can manage, meaning it perfectly suits applications looking to leverage it for pure data management purposes. There are active PostgreSQL systems in production environments that manage in excess of 4 terabytes of data. [PostgreSQL, 2016]

"PostsXML2Postgres.py" utilizes the ElementTree XML library and "psycopg2" to parse and insert the data into the database table. However, before data can be inserted into a table, it must first be created. The program creates the table and appropriate indexes using the SQL listed in figure 4.1 on page 44, and figure 4.2 on page 45 respectively. The ElementTree XML library was chosen for the XML parsing functionality because it is a simple and efficient API for parsing and creating XML data. ElementTree will store every parsed XML node in memory in order to build the document tree, but this will not scale to parsing a 40GB file on a reasonably provisioned machine. The ElementTree "iterparse" interface parses an XML section into an element tree incrementally, and reports what is happening on each event. Therefore, the ElementTree "iterparse" class was chosen, but previously parsed nodes were freed as it went which resulted in a memory efficient "streaming style" parser. This implemented function is listed in figure 4.3 generate_parsed_xml function using ElementTree's iterparse interface on page 45. This was used to parse 40GB from the Posts.xml file and insert batches of posts of a 1000 posts at a time into the Postgres database. This was done to avoid storing lots of posts in memory at once, and to prevent the database table from being overloaded with inserts. "psycopg2", a Python-PostgreSQL Database Adapter, was used to perform the

```sql
DROP TABLE IF EXISTS Posts CASCADE;
CREATE TABLE Posts (
    Id                    int PRIMARY KEY   ,
    PostTypeId            int not NULL      ,
    AcceptedAnswerId      int               ,
    ParentId              int               ,
    CreationDate          timestamp not NULL,
    Score                 int               ,
    ViewCount             int               ,
    Body                  text              ,
    OwnerUserId           int               ,
    LastEditorUserId      int               ,
    LastEditorDisplayName text              ,
    LastEditDate          timestamp         ,
    LastActivityDate      timestamp         ,
    Title                 text              ,
    Tags                  text              ,
    AnswerCount           int               ,
    CommentCount          int               ,
    FavoriteCount         int               ,
    ClosedDate            timestamp         ,
    CommunityOwnedDate    timestamp
);
```

Figure 4.1: SQL CREATE TABLE For Posts Table

inserts. It is a wrapper for "libpq", which is the official PostgreSQL client library. The "mogrify" functionality correctly transforms and binds Python values for PostgreSQL queries, making the process robust and seamless for a Python developer. Figure 4.4 on page 45 lists a run of the Python script used to import the "Posts.xml" file into the "Posts" table in the Postgres "stackoverflow" database. This process took approximately 65 minutes in total to complete. Once the data has been loaded into the table, SQL queries can be used to manage the data in an extremely powerful and flexible way.

Now that the data has been loaded into the table, the data can be analyzed in interesting ways such as the queries described in 3.1.5 Insights Into The Data to calculate the distribution of score and the correlation between the score and view count of a post. More importantly, the quality subsets for questions and answers (very good, good, bad, and very bad) can be extracted from the table using SQL queries. For both of the following queries, "order by RANDOM()" is used to shuffle the posts, as they dataset comes ordered by date. If not shuffled, this might lead to a sampling error, say if a Python program took the first N posts from one of the outputted CSV files. Figure 4.5

```
CREATE INDEX posts_post_type_id_index ON Posts (PostTypeId)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_score_index ON Posts (Score)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_creation_date_index ON Posts (CreationDate)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_owner_user_id_index ON Posts (OwnerUserId)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_answer_count_index ON Posts (AnswerCount)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_comment_count_index ON Posts (CommentCount)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_favorite_count_index ON Posts (FavoriteCount)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_viewcount_index ON Posts (ViewCount)
    WITH (FILLFACTOR = 100);
CREATE INDEX posts_accepted_answer_id_index ON Posts (AcceptedAnswerId)
    WITH (FILLFACTOR = 100);
```

FIGURE 4.2: SQL CREATE INDEXES For Posts Table

```
def generate_parsed_xml(filepath):
    # xml reader iterable
    fxml = iterparse(filepath, events=("start", "end"))

    xml = iter(fxml)  # turn it into an iterator
    event, root = xml.__next__()  # get the root element
    for event, elem in xml:
        if(event == "end" and elem.tag == "row"):
            yield elem.attrib
            # clear the children nodes of the root to avoid memory consumption
        root.clear()
```

FIGURE 4.3: generate_parsed_xml function using ElementTree's iterparse interface

```
\$ python3.5 PostsXML2Postgres.py
Pre-processing...
Pre-processing took 0.024 seconds.
Processing data...
Table processing took 3271.920 seconds.
Post processing...
Post processing took 680.914 seconds.
```

FIGURE 4.4: Python Script Used to Import “Posts.xml” into Postgres Database

```
COPY (
    select id, parentid, creationdate, score, viewcount,
           title, tags, answercount, commentcount,
           favoritecount, closeddate, communityowneddate, body
    from posts
    where
        creationdate >= timestamp '2014-01-01 00:00:00' and
        creationdate < timestamp '2016-01-01 00:00:00' and
        posttypeid = 1 and
        lasteditdate is null and
        closeddate is [not] null and
        [acceptedanswerid is not null and]
        {SCORE_RANGE}
    order by RANDOM()
) TO '/tmp/questions-{QUALITY}.csv' WITH CSV HEADER;
```

Figure 4.5: PostgreSQL Query Used To Generate Question Quality Dataset

on page 46 lists the PostgreSQL query used to generate the question quality datasets. This query returns all of the questions from the years 2014 and 2015 that are unedited, and then varies depending on the variable "SCORE_RANGE", whether the condition for "closeddate" is set to filter for null or not null, and whether "acceptedanswerid" is included to be not null. The pattern of these variables follows the ruleset for question quality laid out in table 3.2 on page 16 in the 3 section. Figure 4.6 on page 47 lists the PostgreSQL query used to generate the answer quality datasets. This dataset contains not only data about each answer, but each answer's parent question too. Again, answers are filtered on whether they are from the years 2014 and 2015, unedited, and within the "SCORE_RANGE" that is set according to the aforementioned table 3.2 on page 16. However, the question that the answers are associated must also match all these conditions apart from the "SCORE_RANGE". Once these queries are run, there will be 4 datasets each for questions and answers corresponding to the four quality classes. These datasets can be used as training and testing data from the machine learning component of this dissertation.

4.2 Machine Learning Pipeline

The Machine Learning component of this dissertation handles the preprocessing, feature extraction, learning, prediction, and visualization functionality of this dissertation. The term "pipeline" is often used in this context because these functions can often be

```sql
COPY (
    select
        a.id as id,
        a.body as body,
        q.id as parentid,
        q.title as parenttitle,
        q.body as parentbody
    from (
        select id, parentid, body
        from posts
        where creationdate >= timestamp '2014-01-01 00:00:00' and
              creationdate < timestamp '2016-01-01 00:00:00' and
              posttypeid = 2 and
              lasteditdate is null and
              {SCORE_RANGE}
    ) a
    join (
        select id, title, body
        from posts
        where creationdate >= timestamp '2014-01-01 00:00:00' and
              creationdate < timestamp '2016-01-01 00:00:00' and
              posttypeid = 1 and
              lasteditdate is null
    ) q
    on (a.parentid = q.id)
    order by RANDOM()
) TO '/tmp/answers-parents-{QUALITY}.csv' WITH CSV HEADER;
```

FIGURE 4.6: PostgreSQL Query Used To Generate Answer Quality Dataset - Includes Parent Question

viewed as one atomic black box component, where data is input into the pipeline and output after an arbitrary number and kind of processing stages have occurred along the way. There are many technologies available that provide Natural Language Processing (NLP), Machine Learning, and visualization functionality that can be leveraged. Python is a very popular language for these purposes, with a burgeoning communing developing libraries for these purposes such as Numpy, Scipy, Matplotlib, TextBlob, BeautifulSoup, NLTK (Natural Language Toolkit), and Scikit-Learn. As mentioned in the introduction to this section, there are three main programs that are used in this component; "QuestionQualityAnalysis.py", "AnswerQualityAnalysis.py", and "StackOverflowTextAnalysis.py". StackOverflowTextAnalysis encapsulates the feature extraction functionality that is used to analyze Stack Overflow questions and answers. This class is used in the other two programs that analyze questions and answers respectively, in order to reduce

```python
def get_long_word_count(self):
    # compute value if not previously done
    if self.long_word_count is None:
        self.long_word_count = 0
        for word in self.get_words():
            if len(word) >= 7:
                self.long_word_count += 1

    # value previously computed, return it
    return self.long_word_count
```

Figure 4.7: "get_long_word_count" function as an example of the efficient implementation of the StackOverflowTextAnalysis class

code duplication. The class implements efficient and reliable methods for extracting the engineered features such as readability indexes, and character N-Grams. For each post that it analyzes, the necessary calculations will only be performed once, such that if multiple methods need to know how many words in the text have more than 6 characters it will only need to be calculated for the first call of the function, and then that function acts as a getter. Figure 4.7 on page 48 shows an example of how the class is written to accomplish this, specifically the function "get_long_word_count" which is used in multiple other functions as part of their calculations.

This dissertation heavily relies on Scikit-Learn [Pedregosa et al., 2011] and its related dependencies. It was developed as a production ready tool for machine learning applications, whereas NTLK and many other related libraries have their origins as a learning tool in academia. Scikit-Learn has incredibly useful abstractions, helper classes, and harnesses that provide much of the functionality a developer would have to implement using other tools.

The "CountVectorizer" class converts a collection of text documents to a matrix of token counts, where the tokens are words or characters, and the counts are the frequency of their occurrence. If an a-priori dictionary or an analyzer that does some kind of feature selection is not provided, then the number of features will be equal to the vocabulary size found by analyzing the data. Constructing an object with the CountVectorizer class that preprocesses the text (eg. lowercasing) and extracts character N-Grams is shown in figure 4.8 on page 49. This analyzer returns the frequency of the extracted character N-Grams, and can be used as features for predicting the quality of posts. The easiest

```
self.char_ngrammer = CountVectorizer(
    analyzer='char', ngram_range=(3, 4)
).build_analyzer()
```

FIGURE 4.8: Constructing a CountVectorizer analyzer that generates preprocessed character N-Grams of length 3 and 4 characters

```
{
    'sota_ari': 23.114285714285714,
    'sota_body_length': 145,
    'sota_body_text_length': 145,
    'sota_code_percentage': 50.0,
    'sota_coleman_liau_index': 31.241428571428568,
    'sota_email_count': 0,
    'sota_flesch_kincaid_grade': 97.0014285714286,
    'sota_flesch_reading_ease': 97.0014285714286,
    'sota_gunning_fog_index': 5.6571428571428575,
    'sota_is_title_capitalized': True,
    'sota_lines_of_code': 1,
    'sota_lix': 14.142857142857142,
    'sota_lowercase_percentage': 81.9672131147541,
    'sota_num_code_tags': 1,
    'sota_num_p_tags': 2,
    'sota_rix': 0.5,
    'sota_sentiment': 0.2,
    'sota_smog_index': 6.872983346207417,
    'sota_spaces_count': 17,
    'sota_spelling_error_count': 1,
    'sota_subjectivity': 0.2,
    'sota_title_body_similarity': 0.0,
    'sota_title_length': 59,
    'sota_uppercase_percentage': 2.459016393442623,
    'sota_url_count': 1
}
```

FIGURE 4.9: Dictionary representation of extracted engineered features

way to generate many different features was to create a Python dictionary with feature names and values aggregated, an example of which is shown in figure 4.9 on page 49.

However, Scikit-Learn's supervised learning algorithms (known as estimators) cannot use this format as input, but instead need a feature vector representation of the data. Thankfully there is a helper class for this called "DictVectorizer". DictVectorizer transforms lists of feature-value mappings (dictionary) to vectors that can be used with Scikit-Learn estimators. When feature values are strings, this transformer will do a binary one-hot coding. Features that do not occur in a sample will have a zero value in the resulting vector.

Now that the extracted features are represented in a compatible format for Scikit-Learns estimators, the data can be used for training and testing. There are various training

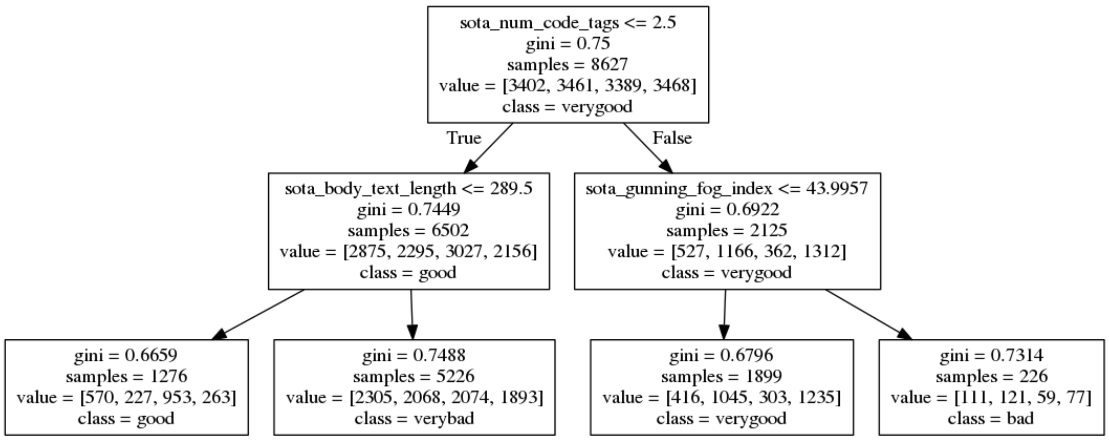
```
X_train, X_test, Y_train, Y_test = cross_validation.train_test_split(
        X, Y, train_size=0.7, random_state=31, stratify=Y)
```

FIGURE 4.10: Stratified splitting of the data into 70% training data, and 30% testing data

and testing strategies for which Scikit-Learn provides harnesses. Figure 4.10 on page 50 shows "cross_validation.train_test_split" being used to perform a stratified split of the data into training and testing subsets, where X notates the features data and Y notates the corresponding labels. This separates the data so that the training phase learns from the learning data alone, and not the testing data. This means that the test data is completely unseen, and therefore provides a realistic test of how the model will perform on real-world new and unseen data. K-Fold equally divides all the samples into k groups called folds. The estimator learns using $k - 1$ folds, and the fold left out is used for testing. This can be used in a rotating fashion so that it learns and tests for all of the unique permutations of the folds. Each fold is constituted by two arrays: the first one is related to the training set, and the second one to the test set. StratifiedKFold is a variation of k-fold which returns stratified folds, each set containing approximately the same percentage of samples of each target class as the complete set.

This dissertation used the Random Forests supervised learning algorithm, and leverages Scikit-Learn's "RandomForestClassifier" implementation. In contrast to the publication [Breiman, 2001] that the implementation is based on, the Scikit-Learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class. The parameters used were simple, between 100 and 10000 decision trees were used, out-of-bag samples were used to estimate the generalization error, the built in parallel feature was used, and the max depth of the tree was sometimes set to a value between 2 and 8. Figure 4.11 on page 51 shows a visualization of a decision tree from the random forest used to classify question quality. This example is from a model that had its max depth of trees set to 3, but good classification accuracy was still reached due to the number of decision trees training being increased. Smaller trees are faster to train, and therefore more decision trees can be trained in the same amount of time than if the trees were larger.

Normalization was implemented using the classes "MaxAbsScaler" and "MinMaxScaler", however as Random Forests are robust in that little pre/post-processing needs to be



sota_num_code_tags <= 2.5
gini = 0.75
samples = 8627
value = [3402, 3461, 3389, 3468]
class = verygood
True
False
sota_body_text_length <= 289.5
gini = 0.7449
samples = 6502
value = [2875, 2295, 3027, 2156]
class = good
sota_gunning_fog_index <= 43.9957
gini = 0.6922
samples = 2125
value = [527, 1166, 362, 1312]
class = verygood
gini = 0.6659
samples = 1276
value = [570, 227, 953, 263]
class = good
gini = 0.7488
samples = 5226
value = [2305, 2068, 2074, 1893]
class = verybad
gini = 0.6796
samples = 1899
value = [416, 1045, 303, 1235]
class = verygood
gini = 0.7314
samples = 226
value = [111, 121, 59, 77]
class = bad

Figure 4.11: Visualization of a Decision Tree from the Question Quality Random Forest

done, little difference was noticed and scaling was left out of the processing pipeline. The former scales each feature individually such that the maximal absolute value of each feature in the training set will be 1.0. The latter scales and translates each feature individually such that it is in the given range (parameter named "feature_range") on the training set. Therefore, if passed the tuple (0, 5), it will scale the features to a value between 0 and 5, however the value of this defaults to 0 and 1.

Scikit-Learn sklearn.cross_validation.permutation_test_score implements test 1 from the paper "Permutation Tests for Studying Classifier Performance" [Ojala and Garriga, 2010]. This test aims to discover if the classifier has found a significant class structure, or in other words if the classifier has found a correlation between the data and the class labels. It does this by permuting the target labels of the data, training and testing on different folds many times randomly. This should simulate a "coin toss" prediction for the target label, which can then be compared with the engineered solution to calculate statistical significance. The more rounds or permutations that this test is allowed to run for, the more accurate and reliable the results should be. However, the test is extremely computationally intensive as it does N rounds of training and testing on the supervised learning algorithm on all of the data.

In certain cases, NLTK was used for its preprocessing implementations, such as stemming and lemmatization. TextBlob was used to calculate the sentiment and subjectivity. BeautifulSoup is a powerful HTML and XML parser, and was used to process the posts in order to extract information such as how many code and paragraph tags existed. Numpy and Scipy were used as they are Scikit-Learn dependencies, and provide efficient data structures and processing routines. They are also extremely useful for arranging values and plots for visualization using tools such as Matplotlib, which was used to generate most of the graphs seen in the Results section.

Chapter 5

Results

This section lists the main results from this dissertation, and provides some insight and analysis into these results. Both tables and graphs are used to make the results data more digestible. The metrics and visualisations used here are explained in the Methodology section, starting on page 9. These results have been generated by what is described in the Implementation section starting on page 42. These results contain classification reports, confusion matrices, feature importances histograms and tables showing the distribution of features across the different quality categories for questions and answers. These results allow the the techniques such as the learning algorithms to be empirically judged and compared. The Evaluation section discusses these results and the rest of the dissertation in a more holistic manner. The classification of both quality and answers prove to be statistically significant results, with a worst case p value of 0.019 across all runs. Increasing the amount of data and permutations used in the significance test would improve the classification results and significance level even further, however computational resources and time are a finite resource. The baseline accuracy (coin toss) for this 4 label classification is 0.25. The following results were all generated using 4900 samples per quality level, making the total sample size 19600 questions or answers for each of the classifications. This sample size was chosen in order to keep the categories and amount of data consistent because it is roughly the smallest category size across all the quality distributions, as shown in figure 3.3 on page 16. The question quality classification was run with up to 12000 samples per class, but this would mean a comparison between classifying question and answer quality would not be fair.

	Precision	Recall	f1-score
verybad	0.38	0.23	0.28
bad	0.44	0.40	0.42
good	0.44	0.62	0.52
verygood	0.57	0.62	0.59
avg / total	0.46	0.47	0.45

Table 5.1: Classification Report for Question Quality

5.1 Predicting the Quality of Questions

Table 5.1 shows the question quality classification report. As mentioned in the Methodology, recall is the fraction of the documents that are relevant to the query that are successfully retrieved, precision is the fraction of retrieved documents that are relevant to the query, and F_1 score is a combination of the two. The “verygood” and “good” classes have the most successful classification rates, with an F_1 score of 0.59 and 0.52 respectively. The “bad” class prediction rates is where the accuracy begins to significantly tail off, with the worst accuracy occurring for the classification of “verybad” quality questions.

The confusion matrix graph shown in figure 5.1 shows how the predicted quality of questions compares to the ground truth of the questions in a visually intuitive way. The same tailing off can be seen along the diagonal in the confusion matrix as in the classification report. This graph provides a very intuitive look into the behaviour of the question quality model. It can be seen that many of the verybad quality questions are being misclassified as good quality questions. A hypothesis for this is that the surface level features such as readability indexes and the length of the text can separate the quality of the higher quality questions, but once the content is of a low quality it overrides any other factor. As this dissertation does not do any content level analysis, this will lead to misclassified questions.

The feature importance graph shown in figure 5.2 shows how important the contribution of each feature in the set has towards correctly predicting the quality of questions. This is one of the most useful outputs in terms of gaining insight into what features are indicative of quality, and what the subjective view of quality is in a community. Readability indexes appear to be strongly correlated with question quality as several of them appear near the top of the feature importances. The number of code tags (counted both as the

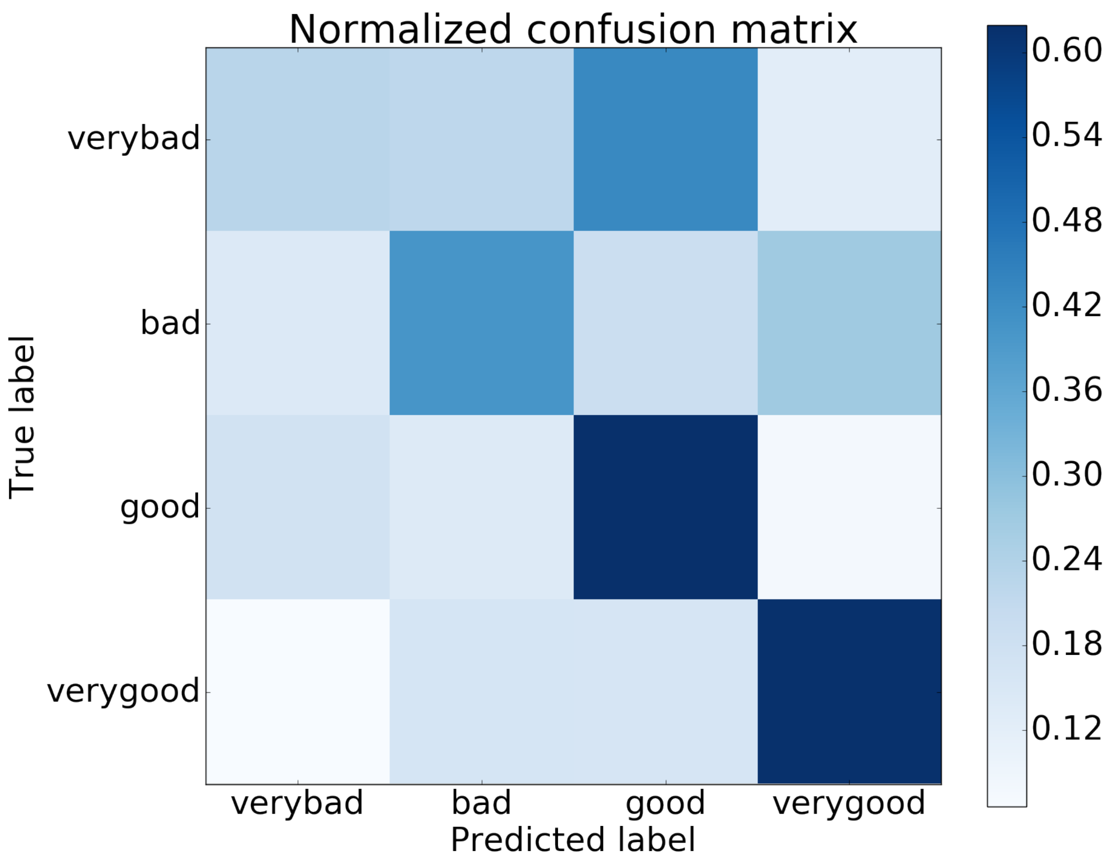


Figure 5.1: Normalized Confusion Matrix for Question Quality

"sota_num_code_tags" engineered feature and as character n-grams like "cod") and the body text length appear highly important.

Table 5.2 shows the distribution of the features (except for character n-grams) used to predict question quality, by calculating each ones median, mean and standard deviation. This table shows interesting patterns, such as the median, mean, and standard deviation for Flesch Reading Ease all going up as the quality decreases from verygood to verybad question quality.

5.2 Predicting the Quality of Answers

Table 5.3 shows the answer quality classification report. The behaviour of classifying the quality of answers is very similar to the behavior of classifying the quality of questions. The "verygood" and "good" classes have the most successful classification rates. The "bad" class prediction rates is where the accuracy begins to significantly tail off, with the worst accuracy occurring for the classification of "verybad" quality answers. The

	Very Good			Good			Bad			Very Bad		
Feature	Median	Mean	SD	Median	Mean	SD	Median	Mean	SD	Median	Mean	SD
ari	16.24	21.78	23.45	17.52	24.6	26.97	15.5	24.86	32.48	12.82	20.57	29.29
body length	836.5	1188.19	1395.63	881.5	1245.87	1379.66	639.5	1113.66	1701.6	468.0	805.89	1329.77
body text length	699.0	1028.56	1316.31	752.0	1100.11	1310.32	554.5	1008.88	1631.91	396.0	725.59	1264.07
code percentage	40.0	40.54	24.29	50.0	46.68	24.91	50.0	46.86	27.06	44.44	43.17	25.61
coleman liau index	12.05	13.15	7.45	11.45	12.29	5.91	10.86	11.73	6.01	10.05	10.53	5.44
email count	0.0	0.08	3.71	0.0	0.04	1.16	0.0	0.02	0.26	0.0	0.03	0.67
flesch kincaid grade	9.6	13.42	17.22	10.17	15.2	20.2	8.73	15.44	24.32	7.2	12.53	21.95
flesch reading ease	75.64	67.02	49.39	77.41	67.06	54.86	81.06	67.49	64.02	85.68	75.9	57.34
gunning fog index	15.95	19.97	17.82	17.07	22.41	21.15	15.38	22.45	25.61	13.25	19.34	23.24
is title capitalized	1.0	0.89	0.31	1.0	0.85	0.35	1.0	0.82	0.39	1.0	0.83	0.38
lines of code	5.0	14.67	27.46	9.0	18.67	29.74	5.0	20.01	44.94	1.0	13.8	35.93
lix	46.91	56.73	44.59	49.44	62.51	52.86	44.88	62.51	63.94	40.0	54.66	58.01
lowercase percentage	70.45	68.54	7.99	67.39	65.46	9.71	68.24	65.5	10.69	69.96	66.77	10.29
num code tags	2.0	2.9	3.69	2.0	2.45	2.67	1.0	1.37	1.92	1.0	1.01	1.5
num p tags	4.0	4.33	2.8	4.0	4.2	2.48	3.0	3.28	2.31	2.0	2.78	2.01
rix	5.0	7.14	10.35	5.25	7.79	10.69	4.25	7.44	11.67	3.5	5.87	10.18
sentiment	0.05	0.05	0.17	0.05	0.05	0.17	0.04	0.05	0.18	0.05	0.07	0.19
smog index	12.49	13.58	4.99	12.62	13.84	5.39	11.66	13.04	6.05	10.75	11.72	5.36
spaces count	125.0	198.85	318.75	139.0	232.92	320.11	103.0	232.28	446.13	74.5	167.83	394.17
spelling error count	9.0	20.27	42.47	11.0	22.88	45.87	8.0	22.38	52.97	5.0	13.96	36.31
subjectivity	0.44	0.44	0.19	0.44	0.44	0.19	0.41	0.41	0.22	0.41	0.4	0.23
title body similarity	0.25	0.27	0.15	0.21	0.24	0.14	0.21	0.24	0.16	0.23	0.25	0.17
title length	51.0	54.03	19.96	50.0	52.38	18.93	46.0	49.69	20.1	45.0	48.04	19.39
uppercase percentage	3.46	4.1	2.65	3.55	4.44	3.43	3.31	4.23	3.66	3.09	3.82	3.01
url count	0.0	0.76	1.45	0.0	0.6	1.35	0.0	0.45	1.58	0.0	0.29	0.88

TABLE 5.2: Question features Statistics - Median, Mean and Standard Deviation

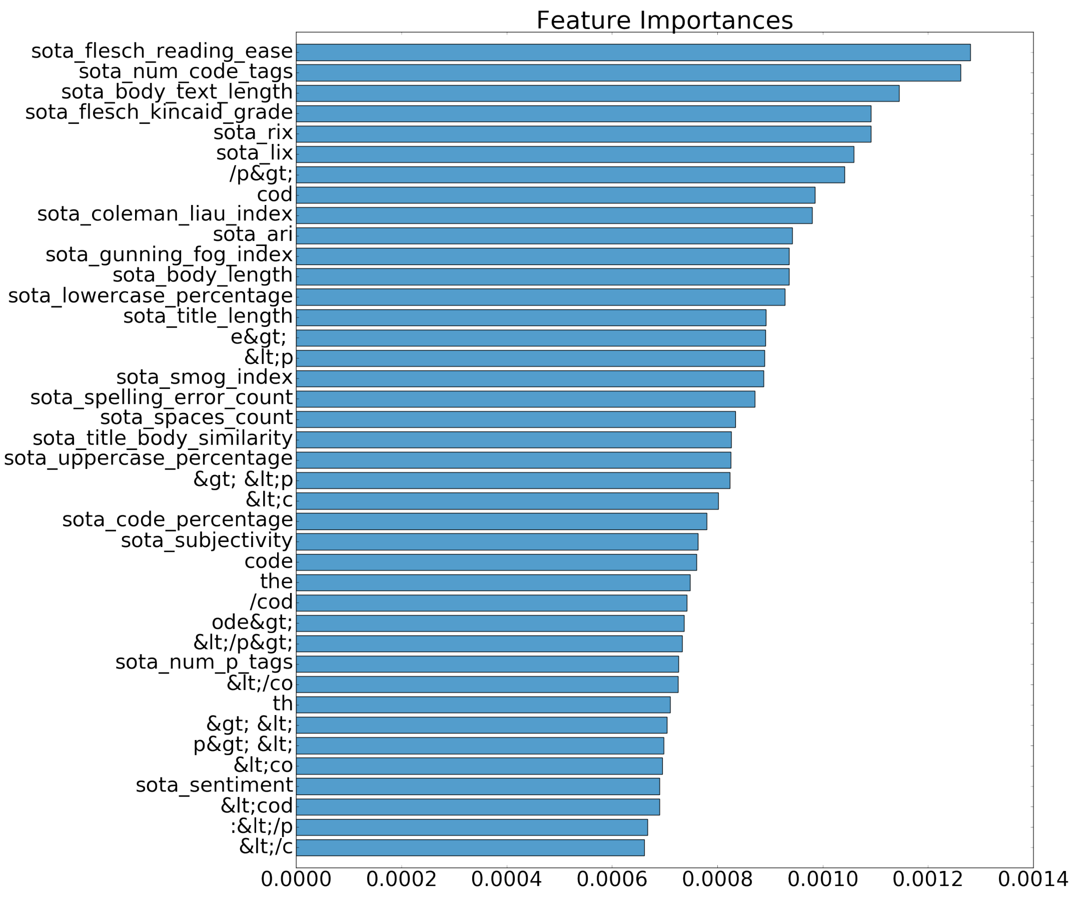


Figure 5.2: Feature Importances for Question Quality

	Precision	Recall	f1-score
verybad	0.34	0.23	0.27
bad	0.43	0.36	0.39
good	0.39	0.48	0.43
verygood	0.54	0.68	0.60
avg / total	0.43	0.44	0.42

Table 5.3: Classification Report for Answer Quality

minor difference here is that the tail off is more significant with answers than it is with questions. This may be due to the fact that it is clearer when an answers is wrong than a question being “wrong”. Following on from the same hypothesis as with questions, this dissertation is unable to verify content level quality and therefore the overriding incorrectness of the information will not be detected.

The confusion matrix graph shown in figure 5.3 shows how the predicted quality of answers compares to the ground truth of the answers in a visually intuitive way. The same tailing off of performance can be seen along the diagonal in the confusion matrix

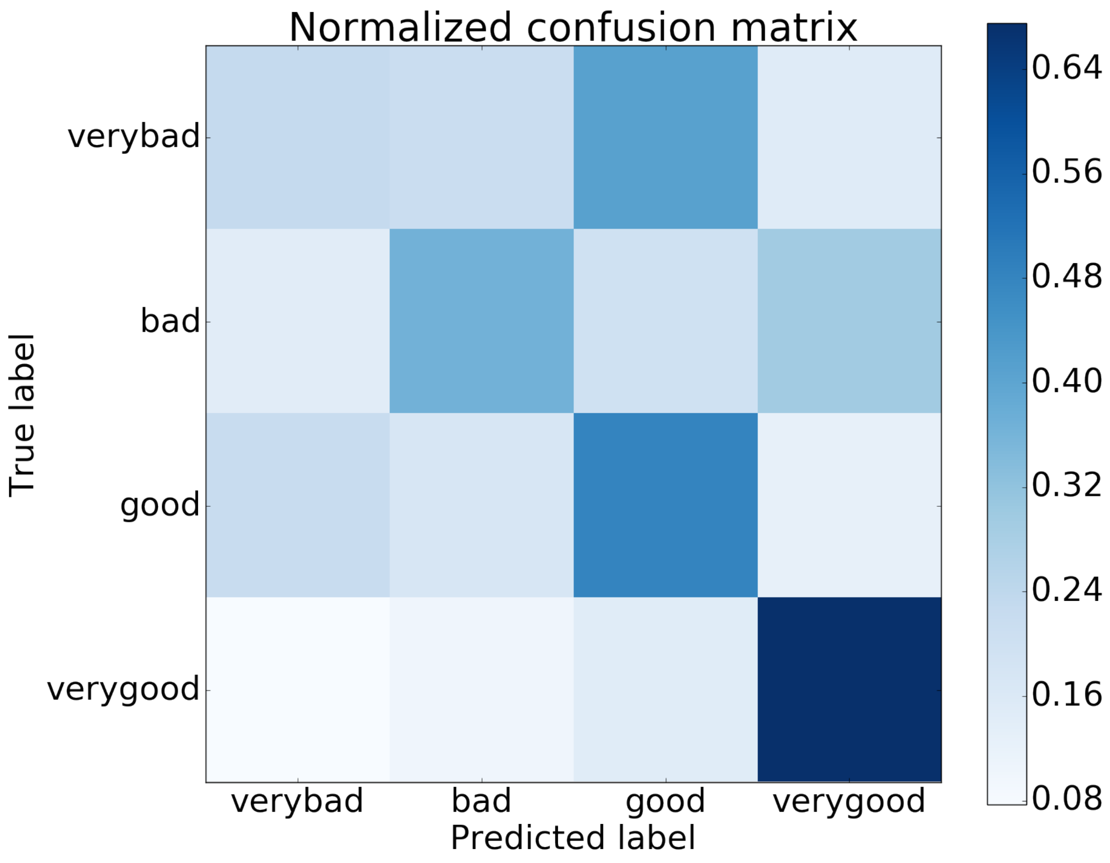


Figure 5.3: Normalized Confusion Matrix for Answer Quality

as in the classification report as the quality class moves towards verybad. This graph provides an intuitive look into the behaviour of the answer quality model. It can be seen that many of the verybad quality answers are being misclassified as good quality answers.

The feature importance graph shown in figure 5.4 shows how important the contribution of each feature in the set has towards correctly predicting the quality of answers. This is one of the most useful outputs in terms of gaining insight into what features are indicative of quality, and what the subjective view of quality is in a community.The length of the answer can be seen as an important feature of quality. An interesting feature shown as the 6th most important is "sota_answer_question_body_similarity", which calculates the word similarity between an answer's body and its associated question's body. There are multiple potential aspects of quality that this feature may be picking up on, however it is most likely that an answer is covering the questions asked. It may also be a sign that using similar, understandable, and common terminology leads to a higher scored answer. Alternatively, if an answer repeated the questions as a quoted list answer each one in order, this feature would also score highly. Readability indexes also appear highly here, as shown in figure 5.2 previously. Code and paragraph tags are also highly important as they also were with questions.

An interesting indicator of quality that the character n-grams extracted was whether “rel=nofollow” was present on links or not in the answer body. In summary, Stack Overflow puts this tag on links in questions and answers so that they will not benefit the linked page on search engines. [Bondy, 2015] This means that spammers have less incentive to spam their links, and also has the resulting effect that Stack Overflow pages will not be increasing the rank of its competitors’ sites. The Stack Overflow community pressured this to change though, and a policy began that high score and high profile posts would have these nofollow tags removed from their links. Therefore, this is a highly accurate indicator of high quality posts. However, it is cheating in the context of this dissertation, as the goal is to classify the quality of posts at the time of their creation. As this nofollow tag will always be present at the time of creation, this feature will not be helpful when always predicting newly posted questions and answers.

Table 5.4 shows the distribution of the features (except for character n-grams) used to predict answer quality, by calculating each ones median, mean and standard deviation. This table shows interesting patterns, such as the median and mean for body and body text length decreasing as the quality decreases from verygood to verybad answer quality.

Feature	Very Good			Good			Bad			Very Bad		
	Median	Mean	SD	Median	Mean	SD	Median	Mean	SD	Median	Mean	SD
answer question body similarity	0.16	0.19	0.14	0.16	0.19	0.15	0.13	0.17	0.15	0.13	0.17	0.15
ari	15.82	20.12	17.18	15.73	20.86	18.94	14.51	20.64	31.97	13.55	18.3	19.37
body length	509.0	754.09	883.0	490.0	673.61	683.65	328.5	490.26	708.58	291.0	428.95	604.91
body text length	392.0	613.24	791.58	382.0	547.41	620.05	257.0	408.01	654.86	227.0	358.91	572.54
code percentage	39.21	39.52	22.79	40.0	42.2	23.49	50.0	45.34	23.31	50.0	45.09	22.48
coleman liau index	12.17	12.7	6.4	11.85	12.27	6.01	11.56	12.42	7.31	10.95	11.64	6.92
email count	0.0	0.01	0.33	0.0	0.01	0.25	0.0	0.01	0.26	0.0	0.01	0.1
flesch kincaid grade	9.07	12.0	12.79	9.02	12.59	14.14	8.05	12.19	24.28	7.41	10.55	14.46
flesch reading ease	77.53	72.2	39.64	77.95	71.92	40.41	81.63	72.89	66.61	84.07	78.32	43.33
gunning fog index	15.48	18.88	13.45	15.43	19.59	15.15	14.17	18.99	25.36	13.62	17.42	15.35
lines of code	3.0	7.81	17.03	3.0	7.58	14.04	2.0	7.12	15.48	2.0	6.44	15.35
lix	45.62	53.85	33.59	45.27	55.24	37.74	41.75	53.61	63.37	40.53	49.65	38.44
lowercase percentage	70.9	68.51	9.3	70.9	68.0	10.28	70.56	67.59	10.41	70.65	67.66	10.44
num code tags	2.0	2.93	3.81	1.0	2.34	3.05	1.0	1.21	1.72	1.0	1.05	1.41
num p tags	2.0	3.06	2.58	2.0	2.65	2.0	2.0	2.07	1.63	2.0	1.95	1.54
rix	4.67	6.37	6.79	4.5	6.27	6.63	4.0	6.05	12.68	3.5	5.24	7.23
sentiment	0.03	0.06	0.2	0.03	0.07	0.19	0.0	0.07	0.21	0.0	0.07	0.21
smog index	12.49	13.03	4.73	12.49	12.96	4.65	11.25	12.29	5.53	10.75	11.66	4.81
spaces count	67.0	118.15	203.32	66.0	106.26	146.19	44.0	84.56	233.83	39.0	72.75	159.45
spelling error count	5.0	10.02	17.5	5.0	9.24	16.24	4.0	7.8	15.13	4.0	6.74	13.71
subjectivity	0.41	0.4	0.26	0.4	0.39	0.26	0.37	0.35	0.28	0.37	0.35	0.29
uppercase percentage	3.11	4.06	3.59	2.99	4.22	4.44	3.0	4.24	4.48	2.9	4.27	4.84
url count	0.0	0.75	1.31	0.0	0.7	1.23	0.0	0.5	1.12	0.0	0.42	1.01

Table 5.4: Answer Features Statistics - Median, Mean and Standard Deviation

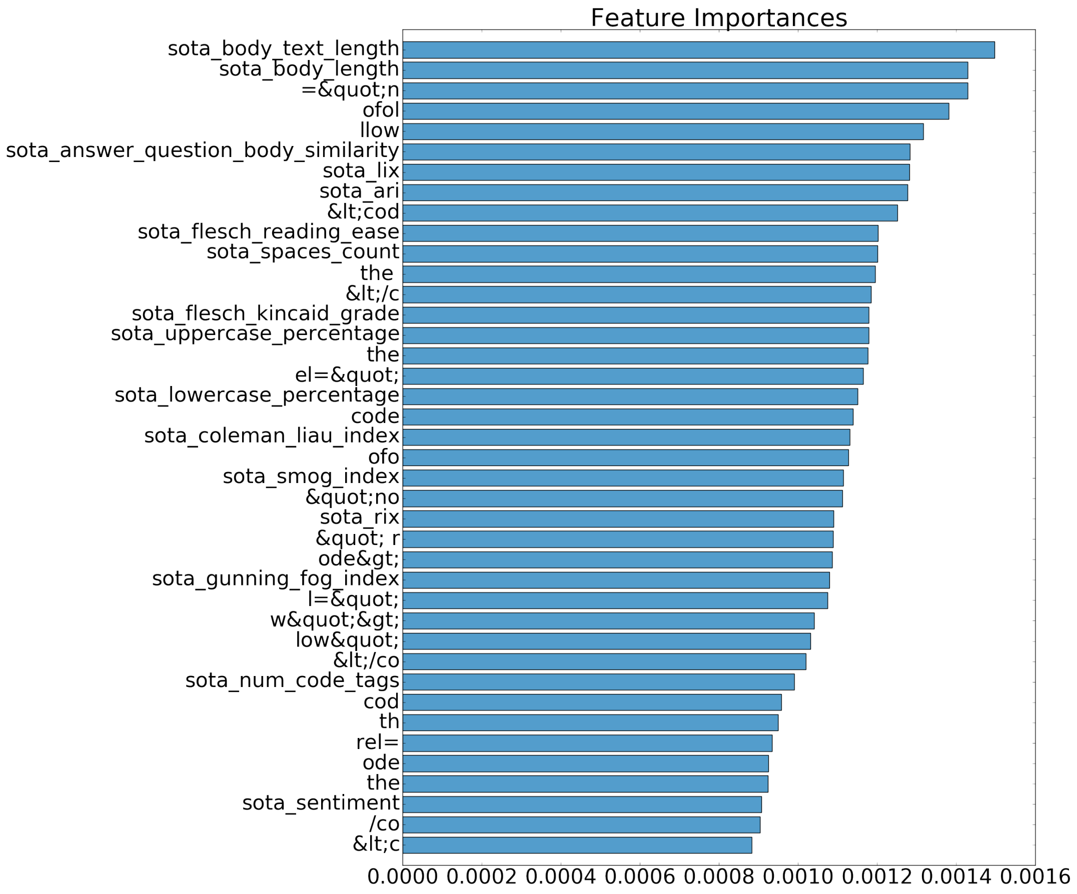


Figure 5.4: Feature Importances for Answer Quality

Chapter 6

Evaluation

This Evaluation section evaluates how successful this dissertation has been in terms of empirical results, anecdotal evidence, and the significance of these findings to academic and industry settings. The success with which the quality of questions and answers was classified is evaluated, followed by the evaluation of the chosen methodology.

Without research being applied to do something useful, can the findings really be significant or matter? This is particularly important in a field such as Machine Learning, which is founded in solving very practical problems such as email spam and financial markets. Ensuring that techniques and theory discovered in the process of machine learning research are communicated well and made useful to its intended application area is critical to making the research truly matter. The paper "Machine Learning that Matters" [Wagstaff, 2012] aims to start a discussion about how "machine learning research has lost its connection to problems of import to the larger world of science and society", and what changes need to be made to put the focus back on machine learning that matters. A hyper-focus on artificial benchmark data sets is one thing that distances research from real world application. While a novel idea may be superior to other methods on that particular data set, it may not necessarily transfer directly to real world data sets in specific problem domains. It is perfectly acceptable for a paper to test and report the results on these benchmark data sets, but perhaps more effort should be made on testing the research on real data to aid in the transfer of research to application. Another point from this paper is the hyper-focus on abstract metrics with the intention of allowing comparison of these metrics across domains. These abstract metrics ignore problem-specific details to allow cross domain comparison, however this is assuming that the performance across domains is comparable at all.

The example of this Wagstaff uses is that “80% accuracy on iris classification might be sufficient for the botany world, but to classify as poisonous or edible a mushroom you intend to ingest, perhaps 99% (or higher) accuracy is required”. Just because metrics have the same range across domains, does not mean that they have the same meaning and apply in the same manner. In the same way, while 100% accuracy sounds great for this classification problem, it may only need to be semi-reliably accurate in order to find useful applications in the real-world. As stated previously, the baseline accuracy (coin toss) for this 4 class classification is 0.25 and the classification results prove to be statistically significant with a worst case p value of 0.019 (significance is when p ¡ 0.05). This is further proof that the techniques used in this dissertation have found insight into features of quality and can provide a useful prediction of a question’s or answer’s quality.

6.1 Predicting Quality

The central research question of this dissertation was “Can the quality of a question or answer from Stack Overflow be predicted at the time of creation using just its text?”. The answer to this appears to be yes, with empirical and anecdotal evidence to support it. The classification reports for both question and answer show a significantly more accurate prediction than random guesses would achieve. Additionally, this dissertation aimed to discover what features are indicative of quality and has been successful in doing so. The feature importances histogram graphs show what features were important for correctly predicting the quality of both questions and answers. By leveraging white-box techniques such as tree based learning algorithms, interesting insight can be gained by visualizing and analyzing what was learned and how it performed on real data. The outcome of these results shows that these techniques are successful enough to have practical, real world applications.

An important aspect of evaluating the performance of these classifiers is understanding why the misclassifications may be occurring. An interesting and noticeable difference found is that the performance of classifying the quality of questions is better than answers. One hypothesis for this is that the quality of an answer depends on the question, whereas the quality of a question does not depend on its answers. This adds an extra dimension of variability to the quality of an answer. Additionally, this ties back to an earlier point in that the engineered features in this dissertation do not account account for the content level of a post, especially for answers where their quality is directly correlated to many factors related to content. For an answer to be of a high quality, its content must cover and correctly answer the associated question's content. Diving deeper into how the content of a post will affect the classification accuracy in this dissertation may lead to a greater understanding of quality in general. The surface level features such as readability indexes and the length of the text may be able to separate the quality of the higher quality questions and answers, but once the content is of a low quality it overrides any other factor. These low quality questions and answers may have incorrect information, or may be abusive/ignorant in nature. As this dissertation does not do any content level analysis, this will lead to misclassified questions and answered. An example using the two types of features just mentioned would be where the surface level features detect a post that has ideal readability index and length values, but the information in the post is completely wrong causing the community to downvote it.

Quality is a somewhat subjective attribute that will vary from person to person. This means that the target label to be predicted for each post will vary depending on who is judging its quality. Even between experts, it may be difficult to reach a consensus on the judgement of a posts quality. This means that even the "golden standard" of classification of quality is very unlikely to be 100% accurate. The reference to a "golden standard" refers to the case where a very experienced user of Stack Overflow is the classifier, or whether an incredibly advanced implementation of machine learning is the classifier. When subjectivity is added to the problem of trying to predict something, it is no longer a problem where 100% accuracy could be expected. To gain an idea of the accuracy this system may be expected to achieve, a set of posts from each of the quality bins could be presented to experienced users for classification. The results of this experiment would give a good idea of what the "golden standard" for this classification might be.

Added to this argument of a subjective view of quality, is that the quality labels that are to be learned from and predicted are not even labelled by an expert, or at all for that matter. The labels are inferred from several attributes in the dataset, the main one being the attribute “score”. By inferring quality, this allows us to perform an analysis on what are features of quality and can it be predicted for questions and answers. However, due to this process, it has to be questioned whether the pure quality of a post is being inferred, or the quality mixed with other factors such as popularity. The score that a post obtains is heavily correlated with the number of views it receives, as shown in figure 3.4 in the data section. To some degree, this results in the situation of building on shaky foundations. The accuracy of these labels comes into question, but these are the challenges that must be faced when trying to analyze real data. The process can be messy, and sometimes certain assumptions must be made to get real work done. In this case, this assumption seems to have paid off as the empirical results appear to make sense in many cases. For example, it makes sense that the length of a question or answer might correlate with the quality, due to the fact that the length probably correlates with effort. The results of this dissertation seem to back up this hypothesis. The results that suggest this are the feature importances and feature distribution tables, shown in figures 5.2, 5.4, and tables 5.4, and 5.4.

Miller [1956] suggests that there is a limit to the amount of information that we can accurately remember or process. Perhaps there is a similar magic range of distinct information that should be in a question or answer so that it is readable and digestible, whilst comprehensive enough to provide enough information to the community. If Miller [1956] is correct, then the ideal number of chunks of information would be within the range of 5 to 9, 7 plus or minus 2. This could be an interesting hypothesis to explore, although it would be challenging to identify how many ’chunks’ or separate pieces of information there are present. However, counting the number of code and p tags in a Stack Overflow post may be a close enough approximation. If this possibility is explored further, an interesting pattern emerges. If the mean calculation for the number of p and code tags found in a question from table 5.2 are summed together for each of the quality labels, then verygood has just over 7 chunks, good has roughly 6.5 chunks, bad has about 4.5, and verybad has around 3.8 chunks. This means that verygood and good would be in the ideal chunk range, whereas bad and verybad have less than the ideal amount of information chunks. Interestingly, similar behaviour occurs with answers. If the mean calculation for the number of p and code tags found in a answer from table 5.4 are summed together for each of the quality labels, then verygood has around

6 chunks, good has roughly 5 chunks, bad has about 3.3, and verybad has around 3 chunks. Again, this means that verygood and good would be just about in the ideal chunk range, whereas bad and verybad have less than the ideal amount of information chunks.

Ponzanelli et al. [2014a,b] analyze Stack Overflow questions in order to predict quality at the time of creation. They were provided with a list of features that Stack Overflow used at the time to identify poor quality questions. These features were body and title length, emails count, uppercase and lowercase percentage, spaces count, tags count, text speak count, title body similarity, capital title, and Urls count. It is unlikely that this is a comprehensive list, especially as this dissertation's research is set a few years later. However, this list is assumed to be roughly the features that Stack Overflow use. The quality of the content varies from high to low-quality. The job of moderating Stack Exchange Q&A's is left to the community. Stack Overflow performs some basic quality analysis, but this is an area where improvement would have many benefits. Therefore, the results of this dissertation suggest that Stack Overflow should additionally implement checks for features such as the number of paragraphs and code blocks, character n-grams, readability indexes, and text similarity measures.

6.2 Evaluation of Methodology

The methodology choices were a key influence on the success of this dissertation. Data management methods were chosen for their flexibility and efficiency of managing and interacting with the Stack Overflow data. Machine Learning methods were chosen for their effect on the performance of the quality classification, and transparency with regards to gaining insight into what affects this performance. By analyzing what was affecting the performance of the quality classification, the features of quality and how they vary between the quality classes could be discovered.

First of all, Python was the language of choice for this dissertation. Python is a dynamic high-level library that comes with the usual speed versus development ease trade-off that many similar languages share. Everything apart from some shell scripts and the SQL queries was written in Python. This turned out to be a great decision, due to the ease with which Python can be written and maintained, and due to the amazing libraries and community around it. Many 3rd party libraries were utilized in this dissertation, and when issues occurred there was an active community with a mass of experience to help developers solve them. And yes, Stack Overflow posts were indeed used to help solve these problems as they were encountered. The reality of choosing another less popular, more esoteric, technology would mean running into bugs and lack of features that had not necessarily been run into before. This means it would be likely that more development and research time would be spent on fixing issues and implementing things from scratch, even if the functionality that it does provide is far more suited for this project. Julia [Bezanson et al., 2014] is one such language, and "combines expertise from the diverse fields of computer science and computational science to create a new approach to numerical computing". Julia is designed to be written like a high-level language, but have the efficiency of a machine level language such as C. It is beginning to gain a large community, particularly with the Machine Learning community. Julia has many machine learning packages in development, and even has a Scikit-Learn interface package in active development. However, they are all quite young and unstable. Another alternative would have been to use a distributed compute framework such as Hadoop or Spark for this project. However, Python was a great decision as development time is at a premium in a dissertation setting, and the trade-off in performance for less development time is a worthwhile one.

Leveraging Postgres and SQL to do the heavy lifting of the data management freed up a lot of time for the core research of this dissertation. Previously, an assortment of Python scripts were utilized to parse and extract the required data. This was time consuming from a management, engineering, and running-time perspective. Databases are optimized for this, and by loading the data into Postgres and creating indexes it became efficient in those three aspects for the necessary analysis, interaction, and extraction of data. In that respect, its likely there would have been less progress in the core research of this dissertation if this move to a database had not been made. This goes to show it is extremely important to define efficient data management processes for data related projects.

Feature engineering was one of the most significant and time-consuming tasks of this dissertation. For example, once the hypothesis for the quality of an answer depends on both the question and answer was established, a feature could be engineered to explore this. Information such as the title and body of each answer's associated question was added to the dataset for classifying the quality of classification. A feature that measured the similarity between the answer's body and its associated question's body was calculated. This feature proved to be one of the most important features for correctly predicting the quality of an answer, as shown in figure 5.4. The computational time of these engineered features was significant, as there were thousands of posts to calculate them for. A parallel map was used to speed up the embarrassingly parallel problem of calculating independent features for each post by utilizing multiple cores. Character N-Grams were found to be both powerful, and quite efficient to generate. The generation of character n-grams is essentially extracting many sub-strings of characters from a piece of text and keeping count of the frequency of each. The generation of the other features involves more complex operations, but this computation is warranted for the gain in accuracy and insight that was achieved. Eventually, a CPU bottleneck will occur and certain techniques will not be tractable as the amount of data to be processed grows. However, CPU is not the only finite resource, as there are significant memory constraints to be circumvented also.

Some of the core machine learning methods used in this dissertation require the whole dataset to be stored in memory at once. This is not a scalable solution, but there are many techniques for avoiding the issue. Some algorithms in Scikit-Learn implement a method called "partial_fit", which implements an online learning version of the algorithm. Online learning refers to the ability of an algorithm to train on the data in multiple segments rather than all at once. This means that only a subset of the entire data needs to be stored in memory at a time. Unfortunately, Random Forests in Scikit-Learn do not implement this function, even though Random Forests learn in batches, and therefore are unable to perform online learning. This presents an issue as the need for learning from more and more data increases. This issue did not particularly affect the outcome of this dissertation, however it is something to consider for future works that aim to iterate on these methods.

The choice of supervised learning algorithm was an important one. Random Forests were chosen because they are efficient, robust, and intuitive, and easy to tune. The tuning parameters include options for setting the maximum depth of the decision trees, the number of decision trees, the minimum number of samples at a split or leaf node depth, or the number of features to use. Most of these parameters have fairly predictable and understandable results, and allow the performance, accuracy, and behaviour to all be adjusted. Random Forests are scale invariant, meaning that they are able to deal with each feature having different scales, something that can massively affect other algorithms accuracy. This removes much of the normalizing process that usually proceeds the learning phase, and therefore removes another moving part that can affect the results positively or negatively. It also means that the features remain in their natural ranges, which is ideal for analyzing and gaining insight into how the features vary for quality. Random Forests are considered white-box models, largely due to the boolean logic inherent in the decision trees. They provide an interesting look at what values of the features are turning points for the various levels of quality, and the relationships between them. As shown in figure 4.11 on page 51, the Visualization of a Decision Tree from the Question Quality Random Forest, the first decision node contains a boolean condition for the feature "num_code_tags". If there are 2 or less code tags found the True branch will be taken and the boolean condition checking the length of the body is calculated next. If there are 3 or more code tags, then the False branch is taken and the feature Gunning Fog Index is checked next. By tracing these decisions to the leaf nodes, insight into the relationships between these features and how they affect quality is gained.

6.3 Alternative Approaches

This section tries to suggest alternative approaches to circumvent some of the limitations within this dissertation. These limitations stem from the nature of the data and the nature of the chosen methodology.

The choice was made to leave edited posts out of the analysis, due to the fact that the quality is therefore changing with each iteration. It would be interesting to include edited questions and answers and observe what affect this has on the results. Additionally, by correlating timestamps from the "PostHistory" and "Votes" table, neither of which are used in this dissertation, it is possible to attribute upvotes and downvotes to each particular version. This also opens up a whole new research question of how edits affect the quality over time. The PostHistory table contains updates to posts, including edits to their body and title. The Votes table contains each vote for each post over time. The score attribute in the Posts table aggregates these votes over time to a single score calculated using the aggregated upvotes minus the aggregated downvotes. This leads to another alternative approach.

As mentioned in the evaluation of quality prediction, inferring the quality from attributes such as the score has its limitations. However, by using more detailed rules to infer quality this issue could potentially be reduced. For example, it might be valid that a post with 3 upvotes and 0 downvotes is of a higher quality than a post with 5 upvotes and 2 downvotes. These both result in posts with a score of 3, but the makeup of that score is different. That the ratio of upvotes to downvotes might matter with respect to quality is essentially the idea here. Testing which leads to a better classification score would be interesting, but does not necessarily make it a more accurate inference. More inference rules could also be used, such as author reputation or the count of answers (for questions) and comments. The rules of inference are not grounded in science, so it would be worth looking into these further, and how varying them affects the results.

Focusing more on the score attribute, an interesting alternative approach would be to use treat the quality prediction problem as a regression problem rather than a classification problem. Instead of using heuristics to quantize the posts into quality levels, train a regression model on score and predict the actual score of a new post. If quality levels are desired, the output score could then be quantized.

With regards to the scalability of the technology in this dissertation, using a distributed computing framework such as Spark [Zaharia et al., 2010] on a cluster of nodes would provide a scalable architecture with which to develop on. MLlib [Meng et al., 2015] is a powerful machine learning library that supports many distributed learning algorithms. This would allow a massive amount of information to be efficiently processed compared to a local Python solution. Spark even has a library called Sparkit-Learn which copies the Scikit-Learn interface so that programs written for the latter can run on the former with minor modification.

In order to try increase the classification accuracy, an alternative algorithm could be selected. Ponzanelli et al. [2014b] switches from tree methods to genetic algorithms. Although many other methods will not give such an interesting insight into how the quality of questions and answers are predicted, they may provide a more accurate classification. Neural Networks (or deep learning methods) are considered black-box methods, in that they can be validated as working but it is not possible to know why or how. What they lose in intuitiveness, they often gain in performance. Neural Networks are often lauded for their accuracy, and it would be interesting to see if they improve significantly on the current implementation using Random Forests.

6.4 Further Work

This dissertation has answered the main questions it set out to, however as a result there are new and interesting questions to answer. This section discusses the most promising research avenues to pursue in the area of this dissertation. These include areas that are severely lacking, important, or nice-to-haves.

An argument could be made that the content of a question or answer is the most important influence on the perceived quality of a question or answer by a person. Therefore it may seem odd that this dissertation does no such analysis. That is, not surprisingly, because analyzing whether or not information is correct is difficult, or more specifically, not easily implemented with the current open tools. This kind of functionality is akin to what IBM's Watson computer is trying to achieve. [Ferrucci, 2011] Watson can be asked questions and it tries to answer or solve them. It does this by learning from huge amounts of unstructured natural language data on a subject from materials such as Word documents, PDFs and web pages. When a question is asked, it will evaluate what is being asked. Then it will look at its knowledge base and answer using a ranked selection of possible answers and their supporting evidence. To do all this, it performs extremely cutting edge analysis on the grammar and context of the text to extract the meaning. With the current pace of research, this functionality may be available to a masters student in the near future, however it is unfortunately not currently available. Regardless of this, the key to accurate quality analysis is to be able to validate whether the content is correct and other such information.

Related to the above, is the idea that a question and answer should share a similar topic distribution if the answer is properly answering the question. Topic modelling is a technique that can be used to extract clusters of topics from text data. Once topic modelling is run across the questions and answers, their distributions can be compared. If they are closely matched, then it might be likely that the answer is stating relevant information. This could be used as an additional engineered feature for predicting the quality of posts, and would make a useful addition to this system.

Sentiment analysis is rather lacking in effectiveness for technical data at this time. Most such tools are developed and trained to be effective for data from social media sites such as Facebook and Twitter. The sites often have short text with simple language, expressing strong emotions and opinions. Technical language will often differ on both accounts, and things such as tech jargon can cause the overall sentiment to be incorrect. More research should be performed into how to correctly identify the emotions in a technical post, or how to judge the subjectivity of the text more accurately. This analysis should also include such phenomena as hedging and politeness.

This dissertation analyzed a filtered set of posts from the years 2014 and 2015. With more computational resources and by using more scalable technology, it would be extremely interesting to analyze the entire dataset. This number of posts would completely overwhelm many laptops, desktops, and servers. However, 40GB of posts would be light work for a well provisioned Spark cluster. A common piece of wisdom in machine learning is that more data beats a more complex algorithm. The results from this process could be far more comprehensive and interesting, than using the smallish sample size used in this dissertation for practical reasons.

This dissertation chose the technical domain of Stack Overflow for the quality analysis. Transferring these techniques to another domain or community such as Facebook and Twitter would be very interesting. In this case, it may be more applicable to try and predict the popularity of the post, through the metrics of likes, comments, re-tweets, replies, etc. It would be interesting to see not only if they work by re-training on the new domain and testing the performance of the classification, but to train on a different set of data or a combination of the sites data, and then test on a different site or all of the sites.

Chapter 7

Conclusion

This dissertation aimed to discover indicators of quality, and to use this knowledge to correctly classify the quality of questions and answers from Stack Overflow. Both of these goals were achieved quite successfully, with empirical and anecdotal evidence to back them up. The results provide a measure of how well each feature indicates quality, and several accuracy measures for the classification process. The F_1 scores for question and answer quality prediction were 0.45 and 0.42 respectively, which is a significant improvement over a baseline prediction rate of 0.25 for a 4 label classification. However, the performance for the two good quality labels is far better than the performance for the two bad quality labels. The reason for this was evaluated, and the hypothesis of These results proved to be significant, meaning that these techniques have found a real pattern in the data. Therefore, the problem of predicting the quality of a piece of text is solvable.

This dissertation will hopefully be utilized as a foundation for further work in this area, from which others can learn from and build upon to further the classification accuracy and gain more insight about quality. The outcome of this dissertation shows that these techniques are successful enough to be used in some real world applications, and that there is a huge amount of scope for this area to be improved. Moderating Stack Overflow is a time-consuming task essentially performed by volunteers form the community. Improvements in analyzing the quality of Stack Overflow posts to the point where it could be automated would have many benefits. For example, the moderators jobs would become far easier as some of the more obvious low quality posts would be closed or deleted, and high quality posts would be noticed more. Therefore, the quality of the content would be massively improved, benefiting the millions of users of Stack Overflow. Many of the same techniques can be applied to other domains such as the expert website "Quora", or the customer support industry.

Improving to a high-level of classification performance will likely require new and cutting edge methods to be developed and released, especially for the task of content analysis and validation. Not only have the techniques used been described and evaluated, but a set of alternative approaches and further work has been listed in order to circumvent some of their limitations. These include both minor modifications to the methodology, and significant changes in the approach and architecture of the project. However, regardless of how advanced these techniques become, quality will always remain a subjective notion that varies from person to person. This means that achieving perfect performance is not plausible or probable. In an ideal system, the accuracy of the quality classifications will be comparable to that of a subject matter expert. The work performed in this dissertation is a step along the way to this becoming a reality.

Bibliography

Asaduzzaman, M., Mashiyat, A. S., Roy, C. K., and Schneider, K. A. (2013). Answering questions about unanswered questions of stack overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 97–100, Piscataway, NJ, USA. IEEE Press.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2014). Julia: A fresh approach to numerical computing.

Bondy, B. (2015). Stackoverflow amongst nofollow web abuse sites. "https://brianbondy.com/blog/104/stackoverflow-amongst-nofollow-web-abuse-sites".

Bradley, A. P. (1997). The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Calefato, F., Lanubile, F., Marasciulo, M. C., and Novielli, N. (2015). Mining successful answers in stack overflow. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 430–433, Piscataway, NJ, USA. IEEE Press.

Cavnar, W. B., Trenkle, J. M., et al. (1994). N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175.

Coleman, M. and Liau, T. L. (1975). A computer readability formula designed for machine scoring. *Journal of Applied Psychology*, 60(2):283.

Duijn, M., Kučera, A., and Bacchelli, A. (2015). Quality questions need quality code: Classifying code fragments on stack overflow. In *Proceedings of the 12th Working*

Conference on Mining Software Repositories, MSR '15, pages 410–413, Piscataway, NJ, USA. IEEE Press.

Ferrucci, D. A. (2011). Ibm's watson/deepqa. *SIGARCH Comput. Archit. News*, 39(3):–.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W. H. Freeman and Co.

Goodspeed, E. (2015). A diagram showing a perceptron updating its linear boundary as more training examples are added. "https://en.wikipedia.org/wiki/Perceptron/".

Gunning, R. (1952). *The Technique of Clear Writing.* McGraw-Hill, New York.

Ho, T. K. (1995). Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282 vol.1.

Hodgins, G. (2015a). https://data.stackexchange.com/stackoverflow/revision/413629/528335/count-of-new-questions-and-answers-since-a-specified-year.

Hodgins, G. (2015b). Github repository for stackoverflow-quality-analysis. https://github.com/ghodgins/stackoverflow-quality-analysis.

Kincaid, J. P., Fishburne Jr, R. P., Rogers, R. L., and Chissom, B. S. (1975). Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Technical report, DTIC Document.

Klare, G. R. (2000). The measurement of readability: Useful information for communicators. *ACM J. Comput. Doc.*, 24(3):107–121.

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317.

Mc Laughlin, G. H. (1969). Smog grading-a new readability formula. *Journal of reading*, 12(8):639–646.

McGinnis, W. (2015). Beyond one-hot: an exploration of categorical variables. "http://www.willmcginnis.com/2015/11/29/beyond-one-hot-an-exploration-of-categorical-variables/".

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. (2015). Mllib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*.

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81.

NLTK (2015). Learning to classify text. http://www.nltk.org/book/ch06.html.

Ojala, M. and Garriga, G. C. (2010). Permutation tests for studying classifier performance. *J. Mach. Learn. Res.*, 11:1833–1863.

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

Ponzanelli, L., Mocci, A., Bacchelli, A., and Lanza, M. (2014a). Understanding and classifying the quality of technical forum questions. In *Proceedings of the 2014 14th International Conference on Quality Software*, QSIC '14, pages 343–352, Washington, DC, USA. IEEE Computer Society.

Ponzanelli, L., Mocci, A., Bacchelli, A., Lanza, M., and Fullerton, D. (2014b). Improving low quality stack overflow post detection. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ICSME '14, pages 541–544, Washington, DC, USA. IEEE Computer Society.

PostgreSQL (2016). Postgresql. http://www.postgresql.org/.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Senter, R. and Smith, E. (1967). Automated readability index. Technical report, DTIC Document.

Sparck Jones, K. (1988). Document retrieval systems. chapter A Statistical Interpretation of Term Specificity and Its Application in Retrieval, pages 132–142. Taylor Graham Publishing, London, UK, UK.

StackExchange (2015a). http://stackexchange.com/sites.

StackExchange (2015b). https://data.stackexchange.com/.

StackExchange (2015c). https://archive.org/details/stackexchange.

StackExchange (2015d). http://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede.

Vogel, C. and Sanchez, L. M. (2012). Epistemic signals and emoticons affect kudos. *3rd IEEE Conference on Cognitive Infocommunications*, pages 517–522.

Vogel, C. and Sanchez, L. M. (2013). Imho: An exploratory study of hedging in web forums. *Proceedings of the SIGDIAL 2013 Conference*, pages 309–313.

Vogel, C. and Sanchez, L. M. (2015). A hedging annotation scheme focused on epistemic phrases for informal language. *Proceedings of the IWCS Workshop on Models for Modality Annotation*, pages 9–18.

Wagstaff, K. L. (2012). Machine learning that matters. In *In Procs of ICML*.

WSO2 (2016). Model evaluation measures. https://docs.wso2.com/display/ML100/Model+Evaluation+

Yeh, A. (2000). More accurate tests for the statistical significance of result differences. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 2*, COLING '00, pages 947–953, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA. USENIX Association.