USING APACHE SPARK'S MlLIB TO PREDICT CLOSED QUESTIONS ON STACK OVERFLOW

by

Preetham Madeti

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Computing and Information Systems

YOUNGSTOWN STATE UNIVERSITY

May, 2016

USING APACHE SPARK'S MlLIB TO PREDICT CLOSED QUESTIONS ON STACK OVERFLOW

Preetham Madeti

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

Preetham Madeti, Student Date

Approvals:

Dr. Alina Lazar, Thesis Advisor Date

Dr. Bonita Sharif, Committee Member Date

Dr. Yong Zhang, Committee Member Date

Dr. Salvatore A. Sanders, Dean of Graduate Studies Date

Dedication

I derive ineffable pleasure in dedicating this thesis to my sweet and loving parents, without whose effort and boundless support none of this would've been possible. I would like to thank my younger brother, Harshith, for standing by me every day and being my greatest admirer. And, I cannot stress how grateful I'm to the almighty for all the blessings.

Acknowledgements

Firstly, I would like to express my deepest gratitude to my advisor Dr. Alina Lazar, for her undivided support and the knowledge she shared with me throughout the process of writing this thesis. I would like to thank her for believing in me giving an opportunity to assist in her research.

Secondly, I would like to thank the committee members Dr. Bonita Sharif and Dr. Yong Zhang for taking time out of their busy schedule and sharing their knowledge and insights.

I would also like to thank all my Professors whose guidance and knowledge has led me into building a good career and values.

Lastly, I would like to thank all my friends and classmates at the University for all the group study sessions and fun we shared.

ABSTRACT

Monitoring posts quality on the Stack Overflow website is of critical importance to make the experience smooth for its users. It strongly disapproves unproductive discussion and un-related questions being posted. Questions can get closed for several reasons ranging from questions that are un-related to programming, to questions that do not lead to a productive answer. Manual moderation of the site's content is a tedious task as approximately seventeen thousand new questions are posted every day. Therefore, leveraging machine learning algorithms to identify the bad questions would be a very smart and time-saving method for the community. The goal of this thesis is to build a machine learning classifier that could predict if a question will be closed or not, given the various textual and post related features. A training model was created using Apache Spark's Machine Learning Libraries. This model could not only predict the closed questions with good accuracy, but computes the result in a very small time-frame.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

Stack Overflow website is an essential and a growing resource among the community of coders all over the world developed and maintained by the Stack Exchange Company. Computer programmers use it to post various questions and answers related to programming. The website, one of the most popular question and answer websites, has slowly evolved into a repository of knowledge. Questions seeking input on some efficient and time-saving methods of coding a particular problem, on getting help on solving various bottlenecks in coding or why code behave in a certain way are commonly seen. To retain its best form, questions posted on the website have to be on-topic, relevant and specific. Before posting any questions, users are required to search the website for similar questions.

Moderation on Stack Overflow is done using badge system. Moderators are elected through popular vote and through regular elections. Users willing to serve as moderator need to earn badges to build their reputation. Considering the influx of approximately 11.80 questions per minute each and every day, moderation has been increasingly becoming tedious. Also, around 800 questions are estimated to be closed on every given week day as of mid-2015 which shows the amount of workload on moderators. As of February 2016, Stack Overflow website has 19 moderators with each one handling around 0.62 questions every minute.

Moderators monitor the website for bad questions. However, bad questions do not get closed immediately, but go through a step by step process. There are several reasons as to why a question gets closed. Moderators close questions for following reasons – off-topic, duplicate, too localized, not a real question and not constructive. The main reason

why questions get closed every day is duplication. As users do not have the patience to search the website thoroughly, exact-duplicates of questions that were previously posted are often submitted. Questions not related to programming are marked ‘off-topic’. Other questions seek for opinion on one’s thought and do not really help in any research effort and therefore are closed as ‘not constructive’. Classifying each submitted question in one of the five categories mentioned above can be done automatically by using text mining and machine learning algorithms. By taking into account all the features responsible for a question being closed, a machine learning classifier model can be created and used for automated detection of the low quality posts.

The goal of this thesis is to extract relevant features from the Stack Exchange data dump and to build a machine learning algorithm capable of predicting which questions are going to be closed. Numerical features of each posted question are created and labels based on its closed or open status are assigned to each question The dataset derived is trained using machine learning algorithms to find the associated pattern between the features, and thereby built a classifier that could predict a closed question [1]. The problem is defined as a binary classification problem and implemented using classification and regression methods.

1.1 Research Motivation and Aim

Previously, researchers used textual features to predict the closed questions using various machine learning algorithms. Data is growing faster and it demands for efficient data analysis tools. With the rise of distributed computing in the big data field, we wanted to leverage its scalability and utilize its machine learning algorithms to give better results.

Currently, Stack Overflow includes 12 million questions, 19 million answers and 47 million comments all available to download as data dump of size 70GB. The data is made publically under the Creative Commons cc-by-sa 3.0 license. That means everyone is free to share this database and adapt it for any purpose, even commercially as long as they properly cite the work. Given the availability and size of the dataset many researchers from fields such as information retrieval, text mining and machine learning have been working with this dataset. Out of the 12 million questions only 503342 questions were ever closed, which means a 4.195%.

Correa et al proposed a method that predicts closed questions by using a machine learning framework [2]. They used features such as question title, tags, body and code snippet and community value and information such as favorite votes, closure time and question status. The classification algorithms they compared were Support Vector Machines, Naive Bayes' and Logistic Regression and Stochastic Gradient Boosted Trees. They found that Stochastic Gradient Boosted Trees gives the best performance out of all classifiers. The algorithm was able to classify 76.5% of closed questions and 69.1% of non-closed questions accurately, which is not very high.

Ponzaneli et al proposed an approach to identify low quality posts on the website [3]. Stack Overflow has a system which could automatically identify low quality posts and put them in a review queue. Their approach relies only on the textual features of the post and results in low precision as it does not consider many other important features of the post and community related aspects of the user. So there is a chance of misclassifying the posts and results in waste of moderator's time. In their research, they used features related to the post as well as community related aspects of the user. These features are

collected from the Stack Overflow metrics and the authors also included some readability metrics and popularity metrics such as community-related aspects. They applied their approach to resolve the Stack Overflow queue by removing the misclassified posts. Their results shown that, they could reduce the size of the review queue and yielded less number of false positives.

Galina et al built a classifier that predicts whether or not a question will be closed given that the question is submitted [4]. Three methods were compared during their research. They are Random Forests, Support Vector Machines and Vowpal Wabbit's online latent Dirichlet allocation algorithm. They used a set of features extracted from the user table such as reputation, the number of up votes it received, the number of down votes it received and number of questions answered by the user are taken into account. Also, they included post related features such as number of blocks of the code, number of links in the body of a post, number of digits, number of sentences, the ratio of upper and lower text characters. Besides, some baseline features such as number of un-deleted questions by the owner at the time of post creation, length of the title, length of the body, number of tags, age of the user and reputation of the user at the time of post creation are used from the baseline model provided by Kaggle. To select the most important features out of these, they implemented a method to estimate the relative importance of features by constructing trees of randomly selected features. The latent Dirichlet allocation algorithm gave them the best result and yielded them position 5 in the leaderboard with 0.31467 points.

Calefato et al researched in a different direction by proposing that emotional style of a post is related to the way it is perceived [5]. The dataset used is from the Stack

Overflow data dump updated on September 2014. The research focused on many factors that the users could implement while reacting to a question. Substantial evidence was established that factors such as information, presentation style and time impact on the success of answers. Their study presents evidence based guidelines which users can follow to improve the chances of getting their answer accepted. This is the first study to show how nuances in language used can affect the success of a post.

In this thesis, we mostly used textual features along with some features related to the popularity of the post such Answer Count, Comment Count, Favorite Count, Closed Date and View Count. This research work is different than others because, we not only focused on deriving features to build a dataframe of labeled point vectors, but also focused on the efficient analysis part of the ever increasing data at hand. Stack Overflow's database has huge amount of data which when completely analyzed, would yield better insights over the data. We developed a training model in Apache Spark using Python and leveraged its pre-built machine learning library to predict the closed questions.

1.2 Organization

Each chapter hereafter, describes the thesis in more detail. Chapter 2 describes the dataset used and its structure, Chapter 3 describes the machine learning algorithms used to obtain the classifier and the method involved. Chapter 4 talks about the extracted features and Chapter 5 show the experiment and results. Finally, Chapter 6 concludes and discusses the scope of future work.

CHAPTER 2. DATASET AND ITS STRUCTURE

The dataset used for this task is obtained from the Stack Exchange Archives. All the questions and answers posted on the Stack Overflow website are stored in a huge database. This database contains information about all the users, posts and related activity on the website. There are a total of 19 tables in the database to store this entire data. The tables are named Posts, Users, Comments, Badges, CloseAsOffTopicResonTypes, PendingFlags, PostFeedBack, PostHistory, PostLinks, PostsWithDeleted, PostTags, ReviewRejectionReasons, ReviewTaskResults, Tags, ReviewTasks, SuggestedEdits, SuggestedEditVotes, TagSynonyms and Votes. To perform the data analysis for this thesis, we used data from the table Posts. This table has 20 fields which gives information about the details and activity related to each post. Table 1 in the next page describes it in more detail.

We downloaded an xml file which is a subset of the entire data from the Posts table. The small file holds 99,997 rows containing information about the details of the posts and has the same meta-structure as the Posts table. A post can either be a question asked or an answer posted to a question. The posts that were marked as closed carries the date it was closed in its respective field. With the help of this information, in order to classify the posts as open and closed we assigned labels for them. The open questions are marked with a label 0 and closed questions are marked with a label 1.

Table 1: Description of the Posts Table

List of Fields	Data Type	Description
Id	int	Tells us the unique Id of a post
Tags	text	Shows various tags associated that describe what a post is about
Title	text	Describes the title of a post
Body	text	Describes the body of a post
AnswerCount	int	Shows the number of answers to a post
CommentCount	int	Shows the number of comments to a post
FavoriteCount	int	Shows the number of people that marked the post as their favorite
Score	int	Shows the score a post received
ViewCount	int	Shows the number of views a post received
AcceptedAnswerId	int	Preset only if PostTypeId is 1. Tells us the Id of a particular answer if the question has an accepted answer
PostTypeId	int	1 if the post is a question 2 if the posts is an answer
ParentID	int	Preset only if PostTypeId is 2. Tells us the Id of a the question to which the answer is related to
CommunityOwnedDate	datetime	Shows the date and time and is present only of the post is community wikied
OwnerUserId	int	Shows the user id of the author of the post
LastEditorUserId	int	Shows the user id of the last person that edited the post
LastEditorDisplayName	text	Shows the user name of the last person that edited the post
LastEditDate	datetime	Shows the date and time of the latest edit on the post
LastActivityDate	datetime	Shows the date and time of the latest activity on a post
CreationDate	datetime	Shows the date and time of the creation of the post
ClosedDate	datetime	Shows the date and time when the post is closed

Out of the 99,997 posts in our file, 2373 posts were identified as closed and the rest are open as shown in the Table 2 below. Eleven out of twenty fields present in the file were used to extract the textual and other related features.

Table 2: Original Dataset

	Total Posts	Open Posts	Closed Posts	Percentage of Closed Questions
Count	99997	97624	2373	2.37

As we started working on the dataset and applied machine learning algorithms upon it we observed some peculiar results. As we delved deeper into the dataset we found that the results are not as expected due to the dataset being highly skewed. As the above table suggests, 97.63% of the posts are open and only 2.37% are closed. This excessive difference in the categories we considered made the dataset highly unbalanced and as a result the machine learning algorithms did not have enough examples from both the categories to train well. Therefore we did some pre-processing wherein we replaced some open posts with the closed ones so that we could get a comparatively balanced dataset. The Table 3 below shows the modified dataset.

Table 3: Modified Dataset

	Total Posts	Open Posts	Closed Posts	Percentage of Closed Questions
Count	99997	84468	15529	15.52

Table 4 below gives a label wise description of some statistics about various features of the dataset. One observation worth taking note is, the average values of 8 out of 10 fields in the following table are greater for the label 1 than that of the label 0. It tells us that irrespective of how many views, answers and comments a post receives, if it is not in accordance with the framed guidelines, it will eventually be closed.

Table 4: Average values of some features in the Dataset

Fields	**Label '1'**	**Label '0'**
Views	18601	2551
Number of words in the text	86.37	76.81
Number of sentences in the code	0.36	0.91
Favorite count	29.95	1.73
Comment count	1.36	0.78
Answer count	9.93	0.94
Tags	2.62	0.64
Score	32.70	10.81
Number of occurences of '?'	0.01	0.04
Number of occurences of 'I'	2.2	1.26

CHAPTER 3. FEATURE EXTRACTION

Feature extraction is the process of building derived values from a dataset which are informative and non-redundant and those that could help in gaining insights into the dataset. This process is used widely in the natural text processing and textual analysis methods.

3.1 Parsing the Data

The text and code components of the body of a post play a vital role in defining the question. We see a lot of instances such as users posting all of their code and seeking help as to where they went wrong. Such questions are not encouraged by the website since its purpose is not to debug but rather, to gain insights on different methods to code a program. So there can be a partial relationship between the length of the code and the status of the question, provided other factors as well. Therefore, in order to calculate the effect of the aforementioned features on a question being closed, we had to parse the body of the post into text and code components [6]. The parsed body of a post could be seen in the Figure 1 in the next page. Features such as the lengths of the text and code are analyzed afterwards.

< text >	Well, here's what I have tried:

< code >	var ts = new TimeSpan(DateTime.UtcNow.Ticks - dt.Ticks)
	double delta = Math.Abs(ts.TotalSeconds)
	if (delta < 60)
	  return ts.Seconds == 1
	elseif (delta < 120)
	  return “a minute ago”

< text >	But there are many more methods of doing this such as the ones using DateTime and TimeStamp. I have one in my mind:
< code >	Int months = Convert.ToInt32 (Math.Floor((double)ts.Days / 365)) int years = Convert.ToInt32 (Math.Floor((double)ts.Days / 365))
< text >	Suggestions? Comments? Ways to improve this algorithm?

Figure 1: Parsed Text and Code from a Post

3.2 Extracted Features

The original data file we used for this purpose has a set of 20 fields as discussed in the Chapter 2. For building a machine learning model for this task, we decided to use textual features and post statistics such as the number of views, answers, comments and favorite count it received. We were able to derive 12 features which were used in previous researches conducted by *Correa et al* [2] and *Lezina et al* [4]. They are all shown in the Table 5 below. The table describes the features and explains what each of these features is.

Table 5: Features Calculated from the Dataset

Features	**Explanation**
Number of Tags	Every question has to be assigned some Tags to let others know which language or which technology it is associated with. The maximum count is 5
Length of the Code Block	Number of sentences in the code block
Length of the Text Block	Number if sentences in the text block
Number of Words	Number of words in the Text Block
Number of occurrences of 'I'	We calculated how many times the user used the word 'I' in the body or in the Title
Number of occurrences of '?'	The number of times the user used the word '?' in the body or in the Title is calculated
Post Score	The score the posts received so far
View Count	Number of views the post received
Answer Count	Number of answers on each post
Comment Count	Number of comments on each post
Favorite count	Number of people that marked the question as favorite
ClosedCreationDateDifference	The number of days elapsed from the day the question was posted till the day it was closed.

CHAPTER 4. MACHINE LEARNING ALGORTHMS AND METHOD USED

In this thesis, we trained the following three machine learning algorithms on the dataframe built from the extracted features:

4.1 Support Vector Machines with Stochastic Gradient Descent

Support Vector Machines are supervised learning models available in machine learning that could be used for binary classification. An SVM model classifies given set of training examples into set of mapped points and divides them into different categories by forming a hyper-plane in between the categories [7] [8]. Intuitively, a hyper plane that is more distant from the nearest training-data point of any class gives a good separation basis.

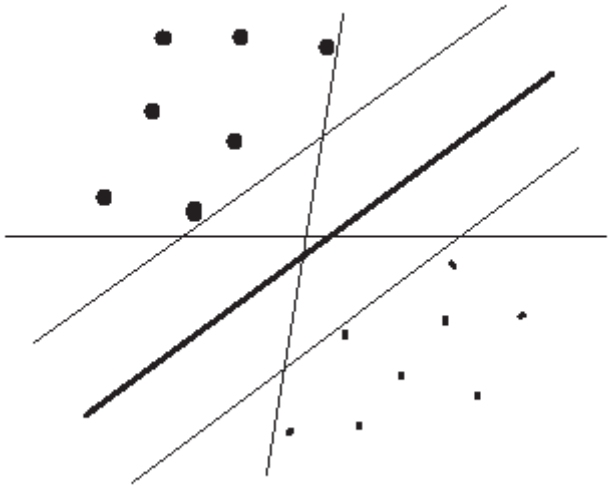

Figure 2: Typical Representation of an SVM model

From the Figure 2 in the previous page, the assumption of SVM is that there can be several hyper-planes dividing the represented by the thin lines, but intuitively the plane with the largest margin (thick line) has the best generalization

SVMs derive their name from the concept of support vectors, which are training points that are not classified with acute probability or are correctly classified but fall inside the margin region. 'Margin' is a measure of how distantly a hyper plane separates the data points. So, it is basically the distance between the hyper plane and the closest point in the dataset. Therefore, once the margin is defined, an SVM model's goal would be to find a hyper plane that separates by the maximum margin from a training set $\{(x_i, y_i)\}$ n i=1, where x_i are the observed data and y_i the labels [9]. The support vector machines algorithm with stochastic gradient descent is different from the usual gradient descents present. It is another way around by which we find the gradient with respect to a single randomly chosen dataset.

4.2 Naive Bayes

The Bayes' theorem is the basis for Naive Bayes. It's a classification technique wherein the set of predictors of a particular item are assumed to be independent of each other. A Naive Bayes classifier assumes that the presence of a particular feature in a class does not depend on any other feature [10]. For example, a living thing may be considered to be a human if it is has two hands and two legs, with a round head on the top. Even if these features are inter-dependent or related to other features in some manner, all of these properties independently contribute to the probability that this living thing is a human and that is why it is prefixed with 'Naive' and was termed Naive Bayes' algorithm.

Bayes theorem provides a way of calculating posterior probability P(c|x). The equation below explains the formula:

$$P(c|x) = \frac{P\ (x|c)P(c)}{P(x)}$$

Where,

P(c|x) is the posterior probability

P(x) is the likelihood

P(c) is the class prior probability

P(x|c) is the predictor prior probability

The model is easy to build and works perfectly for very large data sets. This algorithm is most widely and successfully used among all the other classifiers and machine learning algorithms out there. Along with simplicity, Naive Bayes is known to outdo some deep machine learning algorithms and classification methods as well.

4.3 Logistic Regression

Logistic regression is a classification method in machine learning and is a form of Regression Analysis. It makes use of one or more predictor variables that may be either continuous or categorical. It measures the relationship between the categorical dependent variable and one or more predictors/independent variables by calculating/predicting the probabilities using a cumulative logistic distribution - a type of logistic function [11]. Logistic regression can be divided into multiple types, namely binomial, multinomial or ordinal. Binary logistic regression or Binomial is applied in those problems wherein the observed outcome for a dependent variable can have only two possible outcomes namely,

yes or no, good or bad, win or lose. Multinomial logistic regression comes into play where the outcome can have more than or three possible types (e.g., vitamin A vs. vitamin B vs. vitamin C) that are not in a specific order. The third one, Ordinal logistic regression deals with dependent variables that are in an ordered fashion.

In binary logistic regression, the outcome is usually assigned as 0 or 1, where a 1 refers to a positive outcome and a 0 refers to a negative outcome. For example, in this case study that we made, closed questions were assigned a 1 even though they don't mean to be positive since they are the ones to be filtered out. Contrarily, the open questions were assigned a value 0 as this leads to a more straightforward interpretation. Similarly, if an observed outcome for the dependent variable is apparently the possible outcome (referred to as a success) it is usually assigned a value 1 and the contrary outcome (referred to as a failure) as 0. Logistic regression predicts the odds of being a positive or negative based on the values of predictors. The odds are defined as the probability that a particular outcome is a yes divided by the probability that it is a no.

Logistic Regression is a method of learning functions of the form f: $X \rightarrow Y$, or $P(Y|X)$ in the case where Y is discrete-valued, and $X = (X1 \ldots\ldots Xn)$ is any vector containing continuous or discrete variables [12]. The Figure 3 in the next page demonstrates the form Logistic Regression takes. It is a Sigmoid curve defined by $Y = 1/(1+ \text{Exp}(-X))$, where

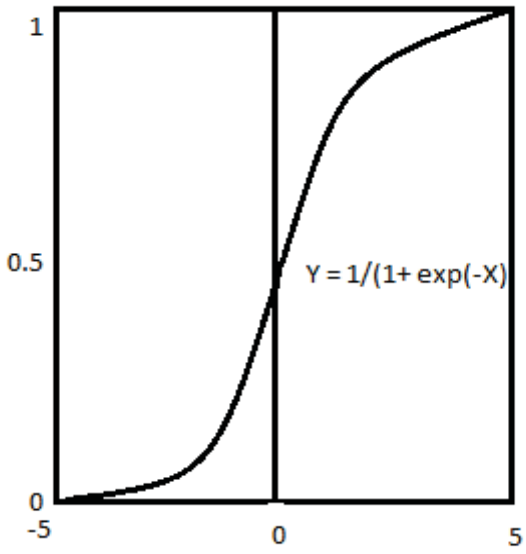

Figure 3: Form assumed by Logistic Regression

The parametric model assumed by Logistic Regression can be explained by the following equations:

$$P(Y = 1|X) = \frac{1}{1+\exp(Wo+\sum_{i=1}^{n} Wi\ Xi)} \qquad (1)$$

$$P(Y = 0|X) = \frac{\exp(Wo+\sum_{i=1}^{n} Wi\ Xi)}{1+\exp(Wo+\sum_{i=1}^{n} Wi\ Xi)} \qquad (2)$$

In the above equations 1 and 2, Y is discrete-valued and a Boolean variable, X $=(X_{1.....}X_{n})$ is any vector containing discrete or continuous variables and where $(w_0.....w_i)$ are the vectors of parameters to be estimated. Logistic Regression assumes a parametric form for the distribution P(Y|X), then directly estimates its parameters from the training data.

We used Apache Spark's machine learning library to train the above discussed machine learning algorithms on the dataset. Apache Spark is an open source cluster computing framework which provides faster data analytics and proven performance on large scale datasets. It is proved to out-perform the previously popular big data platform Hadoop by a scale of 10 to 100 times [13]. The bigger the amount of data, the better the algorithm performed. Apache Spark's prime feature is the Resilient Distributed Dataset and the abstractions it provides. It is a logical collection of data partitioned across various machines. The task is split across various nodes and the assignment is carried out. Similar to Hadoop, there is less chance of node failure in Apache Spark as well since it uses Hadoop Distributed File System to store the data [14]. With a combination of in-memory data analysis and lazy evaluation Apache Spark quickly rose into usage. Lazy evaluation is a method of analyzing the data only when the need arises. The functions in Spark are termed as Actions and Transformations. Whenever a transformation is called upon, the data is read and stored. No further processing is done until an Action is called upon the data. It is at this stage the interpreter performs the data analysis. It is deemed to be a very good alternative to Hadoop's two-stage MapReduce programming style which is comparatively slower in computation as a result of disk storage [16].

Spark version 1.6.1 can be used in Python, Scala, and Java or R environments. We implemented the Python API to calculate the features and implement the algorithms. Besides the speed and ease of computation, it provides us with higher levels of libraries for Machine Learning, SQL, Dataframes, GraphX - for plotting graphs, Streaming and many more tools to accomplish wide range of tasks. It can be integrated with HDFS making it easy for users to migrate from Hadoop.

We have implemented some of the Python libraries to derive features out of the large chunk of text from the data file [17]. The dataset available on the Stack Overflow's website is an xml file which led us to use Python's lxml for the task. It is a natural text processing library that can be used to sort out code and text [18]. We coded accordingly so that the body of the post is separated in to blocks of code and text. Textual Analysis is done with the help of both Python libraries and Spark's Core. Apache Spark offers rich library of machine learning algorithms and utilities, including classification, collaborative filtering, clustering, regression and dimensionality reduction.

CHAPTER 5. EXPERIMENT AND RESULTS

Using the derived features and a label assigned to each post based on the question's status, a Spark dataframe of labeled point vectors is created. It basically represents the attributes derived out of every post in the data file. We applied the algorithms provided by the Spark's Machine Learning Library on this dataframe. The task is performed using the supervised learning algorithms choosing a binary classification method. We tested the data on NaiveBayes, Support Vector Machine with Stochastic Gradient and Logistic Regression all of which fall under classification and regression methods.

We found that there is something else to be taken care of before applying machine learning algorithms on this dataset. The obtained dataframe has various numerical features of varying ranges. So in order to balance the effect of the features uniformly before applying the machine learning model on them, we need to scale all of those features in between 0 and 1. Doing so, there wouldn't be a scope for some features being over-weighed while calculating the accuracy in the prediction step. Both the original dataframe and the scaled dataframe are listed out in the Appendix section. The first and last few rows are listed out to get a better understanding at the resultant dataframe. The code written to filter the data, extract the features and train the machine learning algorithms on the dataframe derived out of the features can be found as well in the Appendix section of this document.

A supervised learning algorithm is one in which the training label has a known class. The model is trained in such a way that it needs to make predictions and will be corrected if the resultant predictions are not as expected. The training process carries on until the desired accuracy is obtained. This trained model is then applied in the test dataset. We split the data into just two sets – a training set and test set in the ratio of 70 and 30. The accuracy obtained from each of the algorithms is different, Naive Bayes outperforming others by a small margin. Table 6 below describes various statistics related to the confusion matrix calculated out of the predicted dataset.

Table 6: Classification Statistics

Statistics	SVM	Logistic Regression	Naive Bayes
Area under ROC	0. 9249487	0. 9483218	0.9574136
Accuracy	0.9021556	0.9249406	0.9345574
Confusion Matrix	Dense Matrix (2, 2, [24064.0, 0.0, 4250.0, 210.0], 0)	Dense Matrix (2, 2, [24064.0, 0.0, 2791.0, 1669.0], 0)	Dense Matrix (2, 2, [24064.0, 0.0, 2108.0, 2352.0], 0)
Precision	1	1	1
Recall	0.849	0.896	0.919
F-measure	0.918	0.945	0.958

The Figure 5 in the next page shows the various accuracies obtained from the all three machine learning algorithm tested against varying percentage of training dataset. The x-axis bears different percentages of the training dataset which starts from a split of 5:95 and continues up to a corresponding value of 80:20 and the y-axis bears the corresponding accuracy values.

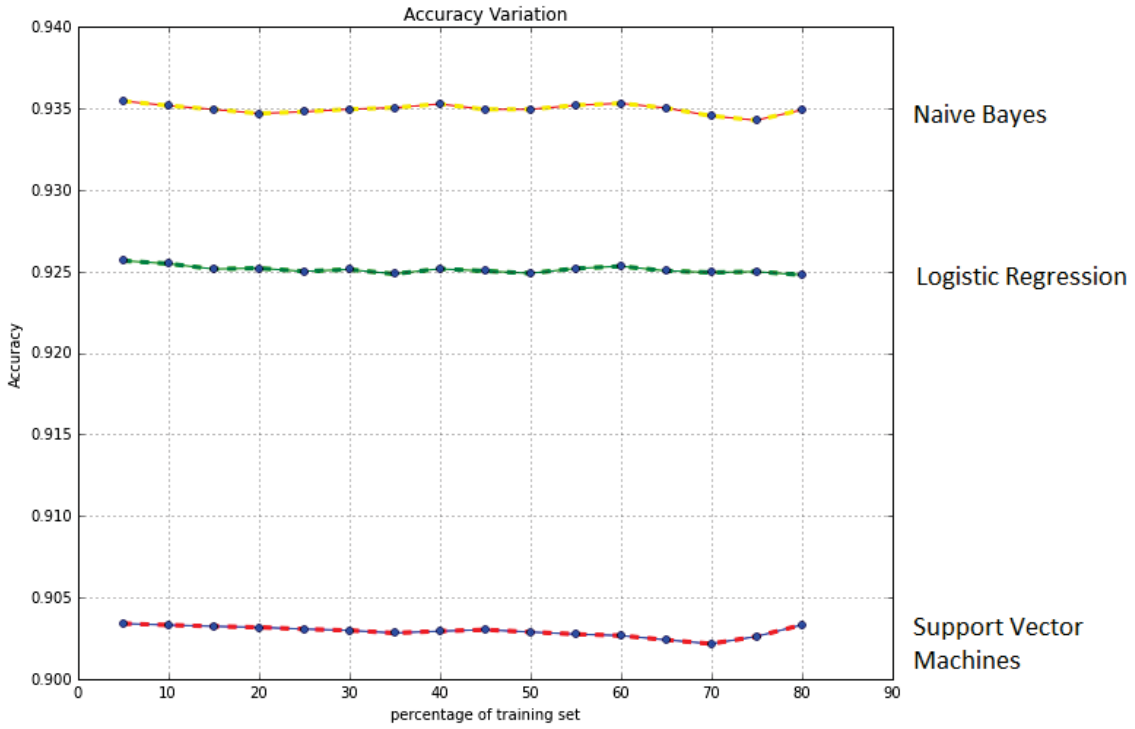

Figure 4: Percentage Training Data vs Accuracy

It can be observed from the above graph that Naive Bayes performed best out of the three algorithms followed by Logistic Regression and Support Vector Machines. Python's matplotlib is used to plot the data.

CHAPTER 6. CONCLUSION AND FUTURE WORK

Stack Overflow is a popular Question and Answer website for programmers around the world and evolved as a dispensable knowledge repository. The website framed some rules and guidelines to be followed while using it. However, it is often violated and affects the quality of the website. Therefore, questions deemed to be unfit on the site are marked as closed by moderators and experienced eligible users. A study was conducted to build a machine learning classifier that could predict the closed questions on Stack Overflow. Naive Bayes performed the best of out of all the three models tested and predicted with an accuracy of 0.9345.

It was found that some questions were classified as open even though they were actually closed. The post and text definitely play an important role in getting a question closed but the user comments should not be overlooked. After careful observation it was found that many questions were closed due to the communication in between the users. Therefore, in later stages, we would like to include more number of features to get more precise analysis of the data at hand. Community related features and user communication analysis can be added in the further research. Features related to the User such as the number of up-votes the user received, users' closed question history and features related to user's account can be added in the further research. Another important feature is the code content. The code content is as important as the length of the code in the post. Gaining more insights in to the user characteristics would definitely help in efficiently analyzing low quality posts.

REFERENCES

[1] B. Li, T. Jin, M. R. Lyu, I. King, and B. Mak, "Analyzing and Predicting Question Quality in Community Question Answering Services," in *Proceedings of the 21st International Conference on World Wide Web*, New York, NY, USA, 2012, pp. 775–782.

[2] D. Correa and A. Sureka, "Fit or Unfit: Analysis and Prediction of 'Closed Questions' on Stack Overflow," in *Proceedings of the First ACM Conference on Online Social Networks*, New York, NY, USA, 2013, pp. 201–212.

[3] "Ponzanelli, Luca, et al. 'Improving low quality stack overflow post detection.' 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2014."

[4] "G. Lezina, A. Kuznetsov, P. Braslavski, 'Learning to predict closed questions on Stack Overflow', Physics and mathematics, Uchenye Zapiski Kazanskogo Universiteta. Seriya Fiziko-Matematicheskie Nauki, 155, no. 4, Kazan University, Kazan, 2013, 118–133."

[5] "Calefato, Fabio, et al. 'Mining successful answers in Stack Overflow.' Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, 2015."

[6] M. Duijn, A. Kučera, and A. Bacchelli, "Quality Questions Need Quality Code: Classifying Code Fragments on Stack Overflow," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, Piscataway, NJ, USA, 2015, pp. 410–413.

[7] "Fan, Rong-En, Pai-Hsuen Chen, and Chih-Jen Lin. 'Working set selection using second order information for training support vector machines.' The Journal of Machine Learning Research 6 (2005): 1889-1918."

[8] *Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297. .*

[9] *Platt, John C. "12 fast training of support vector machines using sequential minimal optimization." Advances in kernel methods (1999): 185-208. .*

[10] *Hall, Mark, et al. "The WEKA data mining software: an update." ACM SIGKDD explorations newsletter 11.1 (2009): 10-18. .*

[11] "Lin, Chieh-Yen, et al. 'Large-scale logistic regression and linear support vector machines using Spark.' Big Data (Big Data), 2014 IEEE International Conference on. IEEE, 2014."

[12] *Hosmer Jr, David W., and Stanley Lemeshow. Applied logistic regression. John Wiley & Sons, 2004. .*

[13] "Zaharia, Matei, et al. 'Fast and interactive analytics over Hadoop data with Spark.' USENIX; login 37.4 (2012): 45-51."

[14] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "MLlib: Machine Learning in Apache Spark," *ArXiv150506807 Cs Stat*, May 2015.

[16] "Bolón-Canedo, V., N. Sánchez-Maroño, and A. Alonso-Betanzos. 'Recent advances and emerging challenges of feature selection in the context of big data.' Knowledge-Based Systems 86 (2015): 33-45."

[17] D. Mertz, *Text Processing with Python*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[18] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2012.

APPENDIX

DATAFRAME

Id	tags	sents @code	sents @text	words @text	num Of I	num Of Q	Score	View Count	Ans Count	Comment Count	Favorite Count	Closed Creation DateDiff	label
4	4	3	15	65	5	1	322	21888	13	1	27	0	0
6	4	5	17	96	4	4	140	10912	5	0	7	0	0
9	3	1	3	21	2	2	862	223405	48	5	256	0	0
11	2	1	11	24	2	1	817	89203	31	11	453	0	0
13	4	1	5	34	0	3	319	89637	23	5	102	0	0
14	1	2	4	15	0	1	208	59368	8	1	33	0	0
16	4	7	29	97	4	5	42	56591	5	0	8	0	0
17	4	0	4	12	1	1	68	25923	10	0	8	0	0
19	5	13	25	179	7	1	164	25934	23	16	58	0	0
24	3	0	4	31	2	1	55	24855	4	0	13	0	0
25	5	7	14	133	10	2	55	5397	10	1	2	2895	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
28922397	4	1	8	89	4	2	2	16	1	6	1	0	0
28922446	4	0	2	71	4	0	0	22	1	2	1	0	0
28922472	5	2	39	95	3	0	0	7	0	4	1	0	0
28922481	4	4	17	85	7	0	1	15	1	1	1	0	0
28922693	5	2	8	38	1	0	0	8	1	7	2	0	0
28922720	2	5	19	181	6	2	0	5	0	0	1	53	1
28922771	1	7	22	236	5	1	-2	17	3	2	1	0	0
28922820	4	1	3	53	2	1	0	7	1	3	1	0	0

SCALED DATAFRAME

Id	tags	sents@ code	sents@ text	words @text	num Of I	num Of Q	Score	View Count	Ans Count	Comme ntCoun t	Favorite Count	Closed Creation DateDiff	label
4	0.750	0.063	0.048	0.038	0.125	0.009	0.095	0.017	0.041	0.029	0.005	0.511	0
6	0.750	0.104	0.055	0.058	0.100	0.036	0.043	0.009	0.016	0.000	0.001	0.511	0
9	0.500	0.021	0.007	0.009	0.050	0.018	0.249	0.175	0.152	0.143	0.045	0.511	0
11	0.250	0.021	0.034	0.011	0.050	0.009	0.236	0.070	0.098	0.314	0.080	0.511	0
13	0.750	0.021	0.014	0.017	0.000	0.027	0.094	0.070	0.073	0.143	0.018	0.511	0
14	0.000	0.042	0.010	0.005	0.000	0.009	0.062	0.046	0.025	0.029	0.006	0.511	0
16	0.750	0.146	0.097	0.058	0.100	0.045	0.015	0.044	0.016	0.000	0.001	0.511	0
17	0.750	0.000	0.010	0.003	0.025	0.009	0.022	0.020	0.032	0.000	0.001	0.511	0
19	1.000	0.271	0.083	0.111	0.175	0.009	0.050	0.020	0.073	0.457	0.010	0.511	0
24	0.500	0.000	0.010	0.016	0.050	0.009	0.019	0.019	0.013	0.000	0.002	0.511	0
25	1.000	0.146	0.045	0.082	0.250	0.018	0.019	0.004	0.032	0.029	0.000	0.790	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
28922386	0.750	0.021	0.014	0.049	0.125	0.018	0.003	0.000	0.000	0.086	0.000	0.511	0
28922397	0.750	0.021	0.024	0.053	0.100	0.018	0.003	0.000	0.003	0.171	0.000	0.511	0
28922446	0.750	0.000	0.003	0.041	0.100	0.000	0.003	0.000	0.003	0.057	0.000	0.511	0
28922472	1.000	0.042	0.131	0.057	0.075	0.000	0.003	0.000	0.000	0.114	0.000	0.511	0
28922481	0.750	0.083	0.055	0.051	0.175	0.000	0.003	0.000	0.003	0.029	0.000	0.511	0
28922693	1.000	0.042	0.024	0.020	0.025	0.000	0.003	0.000	0.003	0.200	0.000	0.690	1
28922720	0.250	0.104	0.062	0.113	0.150	0.018	0.003	0.000	0.000	0.000	0.000	0.511	0
28922771	0.000	0.146	0.072	0.148	0.125	0.009	0.002	0.000	0.009	0.057	0.000	0.511	0

CODE

The following lines of code display step by step procedure of the feature extraction from the downloaded file and training a machine learning model that builds the classifier. As mentioned previously, Spark's Python application programming interface, MlLib and Python libraries are used in the process.

```
sqlContext = SQLContext(sc)
from pyspark.sql.types import *
from pyspark.sql.functions import *
import re
textFile = sc.textFile("/home/datascience/Downloads/Posts.small.xml")

#

postsXml = textFile.map( lambda line: line.strip() ).
filter( lambda line: line != "<posts>" and line != "</posts>").
filter( lambda line: not line.startswith("<?xml version="))

#
from datetime import datetime
def days(d1, d2):
d1 = datetime.datetime.strptime(d1, "%Y-%m-%d")
d2 = datetime.datetime.strptime(d2, "%Y-%m-%d")
return ((d1 - d2).days)

import datetime, dateutil.parser
def parsedate(x):
d = dateutil.parser.parse(x)
return d.strftime('%Y-%m-%d')

#
postsRDD = postsXml.map( lambda s: pyspark.sql.Row(
Id = re.search('Id=".+?"', s).group(0)[4:-1],
Label = 1.0 if re.search('ClosedDate=".+?"', s) != None else 0.0,
Score = re.search('Score=".+?"', s).group(0)[7:-1],
Text = ((re.search('Body=".+?"', s).group(0)[6:-1] if
re.search('Body=".+?"', s) != None else "")+ " " +
(re.search('Title=".+?"', s).group(0)[7:-1] if re.search('Title=".+?"',
s) != None else "")),
Tags = re.search('Tags=".+?"', s).group(0)[6:-1] if
re.search(Tags=".+?"', s) != None else 0 ,
ViewCount = re.search('ViewCount=".+?"', s).group(0)[11:-1] if
re.search(ViewCount=".+?"', s) != None else 0,
AnswerCount = re.search('AnswerCount=".+?"', s).group(0)[13:-1] if
re.search(AnswerCount=".+?"', s) != None else 0,
CommentCount = re.search('CommentCount=".+?"', s).group(0)[14:-1] if
re.search(CommentCount=".+?"', s) != None else 0,
FavoriteCount = re.search('FavoriteCount=".+?"', s).group(0)[15:-1] if
re.search(FavoriteCount=".+?"', s) != None else 0,
ClosedCreationDateDiff = (days(parsedate(re.search('ClosedDate=".+?"',
```

```
s).group(0)[13:-1]),
parsedate(re.search('CreationDate=".+?"', s).group(0)[16:-1])) if
re.search('ClosedDate=".+?"', s) != None else 0)))

#
import lxml.etree
a1 = postsRDD.map(lambda (a,j,b,c,d,e,f,g,h,i):
(d,u''.join(h).encode('utf-8').decode('utf-8'),g,f,i,a,b,c,e,j))
a2 = a1.map(lambda (a,b,c,d,e,f,g,h,i,j): (a, b.replace("&lt;",
"<"),c.replace("&lt;", "<"),d,e,f,g,h,i,j))
a3 = a2.map(lambda (a,b,c,d,e,f,g,h,i,j): (a, b.replace("&gt;",
">"),c.replace("&gt;", ">"),d,e,f,g,h,i,j))

def parsefunc (x):
html = lxml.etree.HTML(x)
code_block = html.xpath('//code/text()')#prints the code
#text_block = html.xpath('// /text()')
text_block = html.xpath('//*[not(self::code)]/text()')#prints the rest
a4 = u''.join(x for x in code_block).encode('utf-8').decode('utf-8')
#converted lxml.eTree_elements into list of strings to obtain a
DataFrame
a5 =  len(code_block)
a6 = u''.join(x for x in text_block).encode('utf-8').decode('utf-8')
a7 =  len(text_block)
a8 = u''.join(text_block).encode('utf-8').decode('utf-8').split(' ')
a9 =  len(a8)
a10 = u''.join(text_block).split()

numOfI = 0
numOfQue = 0
numOfExclam = 0

for x in a10:
if x == 'I':
numOfI +=1
elif x == '?':
numOfQue +=1

return (a4,a5,a6,a7,a9, numOfI,numOfQue)
def tags(x):
tags = ''.join(map(str, x))
return tags.count('<')

a11 = a3.collect()

a12 = map(lambda (a,b,c,d,e,f,g,h,i,j): (a, tags(c), parsefunc(b),
d,e,f,g,h,i,j), a11)

#
import pandas as pd
columns = ['tags', 'sents@code', 'sents@text', 'words@text', 'numOfI',
'numOfQ','Score', 'ViewCount', 'AnsCount', 'CommentCount',
'FavoriteCount','ClosedCreationDateDiff', 'label']

index = map(lambda x: x[0], a12)
data = map(lambda x: (x[1], x[2][1], x[2][3], x[2][4], x[2][5],
x[2][6], x[3], x[4], x[5], x[6], x[7], x[9], x[8]), a12)
```

```
df = pd.DataFrame(data = data, columns = columns, index = index)
df.index.name = 'Id'
df

#
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns,
index = df.index)
df_scaled

#
featuresDF = sqlContext.createDataFrame(df_scaled)
featuresDF.registerTempTable("featuresTable")
featuresRDD = featuresDF.rdd


#
from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint
def parsePoint(line):
values = [float(x) for x in line]
return LabeledPoint(values[-1], values[0:-1])
data = featuresRDD.map(parsePoint)
# Split data aproximately into training (70%) and test (30%)
training, test = data.randomSplit([0.7, 0.3], seed=0)
# Train a naive Bayes model.
model = SVMWithSGD.train(training, 1.0)
labelsAndPreds = test.map(lambda p: (p.label,
model.predict(p.features)))
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)
accuracy =1.0 * labelsAndPreds.filter(lambda (v, p): v == p).count() /
test.count()
accuracy

#

from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(predictionAndLabels)
confusionMatrix = metrics.confusionMatrix
confusionMatrix()


#
from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint
def parseLine(line):
values = [float(x) for x in line]
label = values[-1]
features = values[0:-1]
return LabeledPoint(label, features)

data = featuresRDD1.map(parseLine)
# Split data aproximately into training (70%) and test (30%)
training, test = data.randomSplit([0.7, 0.3], seed=0)
```

```
# Train a naive Bayes model.
model = NaiveBayes.train(training, 1.0)
# Make prediction and test accuracy.
predictionAndLabel = test.map(lambda p: (model.predict(p.features),
p.label))
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)
accuracy = 1.0 * predictionAndLabel.filter(lambda (x, v): x ==
v).count() / test.count()
accuracy

#

from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(predictionAndLabels)
confusionMatrix = metrics.confusionMatrix
confusionMatrix()

#

from pyspark.mllib.classification import LogisticRegressionWithLBFGS,
LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint
def parsePoint(line):
values = [float(x) for x in line]
return LabeledPoint(values[-1], values[0:-1])

data = featuresRDD.map(parsePoint)
# Split data aproximately into training (70%) and test (30%)
training, test = data.randomSplit([0.7, 0.3], seed=0)
# Train a naive Bayes model.
model = LogisticRegressionWithLBFGS.train(training, 1.0)
labelsAndPreds = test.map(lambda p: (p.label,
model.predict(p.features)))
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)
accuracy =1.0 * labelsAndPreds.filter(lambda (v, p): v == p).count() /
test.count()
accuracy

#

from pyspark.mllib.evaluation import MulticlassMetrics
metrics = MulticlassMetrics(predictionAndLabels)
confusionMatrix = metrics.confusionMatrix
confusionMatrix()
```

```
#

%matplotlib inline
import matplotlib
import numpy as np
import matplotlib.pyplot as plt

x= [5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80]
y= [0.90339, 0.90330, 0.90322, 0.90316, 0.90305,0.90297, 0.90281,
0.90292, 0.90301, 0.90287, 0.90274, 0.90265, 0.90239, 0.90215, 0.90259,
0.90330]

x1= [5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80]
y1= [0.92566, 0.92548, 0.92515, 0.92520, 0.92501,0.92512, 0.92485,
0.92516, 0.92503, 0.92489, 0.92518, 0.92532, 0.92504, 0.92494, 0.92499,
0.92479]

x2= [5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80]
y2= [0.93546, 0.93518, 0.93495, 0.93470, 0.93480,0.93496, 0.93505,
0.93528, 0.93495, 0.93495, 0.93520, 0.93532, 0.93504, 0.93455, 0.93428,
0.93491]

fig = plt.figure(figsize=(10,8))
plt.axis([0, 90, 0.90000, 0.94000])
axes = fig.add_subplot(111)
axes.plot(x, y)
axes.plot(x1, y1)
axes.plot(x2, y2)
axes.plot(x,y,color="red", linestyle='dashed',  linewidth=3,
marker='o',markerfacecolor='blue', markersize=5)
axes.plot(x1,y1,color="green", linestyle='dashed',  linewidth=3,
marker='o',markerfacecolor='blue', markersize=5)
axes.plot(x2,y2,color="yellow", linestyle='dashed',  linewidth=3,
marker='o',markerfacecolor='blue', markersize=5)
axes.set_title('Accuracy Variation')
axes.grid()
axes.set_xlabel('percentage of training set')
axes.set_ylabel('Accuracy')
plt.show()
```