Stack Overflow: A Deep Dive into Post Quality Analysis

Chris Chankyo Kim ‘23 and Sohit Gatiganti ‘23

Interactive Demo (Google Colaboratory)

Our project was primarily conducted through a Google Colaboratory notebook so that our group could work together in real-time within an IPython Notebook environment alongside a built-in LaTex editor. We highly recommend viewing the Google Colaboratory file associated with this paper, as the paper primarily complements the work done in the notebook. The notebook proceeds with a step-by-step explanation and interactive demonstration of models and tests. Please follow the initial instructions at the top of the file and run each cell in order, to ensure no errors. Our work process and results are publicly available with the following link: https://colab.research.google.com/drive/13V4uGmblY-QyaLlq__dkzSt4UjeLDH3l?usp=sharing

Video Presentation (YouTube)

Our group has also created a video presentation of our project that is publicly available on YouTube with the following link: https://youtu.be/-szBYl9gKxU

Introduction

Motivation

The internet provides opportunities for individuals to rapidly and easily share ideas and information throughout the world. Notable examples include social media platforms and online forums/sites that foster direct communication between users. However, quality of shared content is not guaranteed, as it is common for online posts and/or messages to either lack essential information or contain misinformation. As a consequence, it is generally difficult for users to differentiate between valid and unreliable online content. As students, we recognize the importance of the internet as an informational toolbox for guiding our learning. For this reason, it is beneficial to develop methodologies of filtering the quality of information we see online. In this project, our group seeks to evaluate and develop feasible probabilistic tools to categorize quality of online content, specifically for academic online forums.

Stack Overflow

One of the most popular online forums available is Stack Exchange, a network of multiple question-and-answer (Q&A) sites that span a diverse range of fields. Most notably, Stack Overflow is the principal site for professional programmers and enthusiasts to prompt and discuss questions regarding computer science. A remarkable feature of Stack Overflow, and Stack Exchange affiliated sites in general, is their user reputation award process. This feature acts as the primary governing platform for evaluating quality of questions and answers, allowing the site to be self-moderating by its community. The self-moderation feature inherently provides a rich and extensive database of high-quality and low-quality posts, making it a quite optimal subject to explore in our project.

Our group chose to analyze posts specifically from Stack Overflow in part due to its relevance to CS 109, as it is a primarily computer science driven forum. Further, Stack Overflow is a very popular site for students in general, who often inquire questions or assistance on classwork/projects. For the above reasons, we determined that analyzing Stack Overflow's posts based on quality-level would not only provide great insight on how individuals may discern information online, but also demonstrate local relevance to the course curriculum.

Project Goals

In this project, we seek to primarily accomplish the following goals:

1. Identify significant features that may indicate differences between high and low quality post questions/answers
2. Develop a working probabilistic model(s) that is effective at discerning high and low quality posts, given a body of text

Data

Acquisition

The primary dataset that we based our probabilistic models on is sourced from Kaggle, an online community of data scientists and machine learning practitioners that contains a vast collection of databases on a variety of topics. Our analysis is drawn from the database "60k Stack Overflow Questions with Quality Rating" (Moore). This dataset was favorable because the posts were pre-categorized into distinct quality groups, and there were two sets of training data (45,000 posts) and testing data (15,000 posts) provided. As such, we were able to skip the process of cleaning raw data. In the latter portion of our project, we have also conducted minor analysis with a larger, yet less organized, dataset "StackSample: 10% of Stack Overflow Q&A" by Stack Exchange (Stack Overflow). This dataset represented about 10% of the total number of posts on Stack Overflow at the time of measurement.

Parameters and Features

From both datasets, we primarily sought to evaluate the relationship between quality of posts and discrete parameters -- such as body text, title, and related tags -- and concrete parameters -- such as creation time. Our primary dataset distinguished post quality into three categories -- High Quality (HQ), Low Quality Edited (LQ_EDIT), and Low Quality Closed (LQ_CLOSE) -- and contained information on a post's title, body text, creation time, and related tags. For each of these parameters we followed a three step procedure for initial analysis:

1. Determine the mean, median, and standard deviation of the particular subset of information in our sample. These subsets were distinguished by post quality for comparison.
2. Determine the p-values for the difference in means and medians between High and Low quality posts for each subset information.
3. Perform a rudimentary prediction model based on the absolute difference of a data point from the mean (i.e. the difference from the mean for our target post quality is less than that of the mean of our non-target post quality, then the prediction is accurate).

For each feature, we also evaluated its Pearson Correlation Coefficient with every other feature to determine which are independent. Upon inspection, every feature had a fairly low coefficient (< 0.128) with other features. With this result, we proceeded to develop our models based on the assumption that these features are independent from each other. The purpose of this analysis was to extract a subset of independent features that we could hypothesize to be sufficiently independent parameters in training our logistic regression and Naive Bayes model. Moreover, it helped us understand our dataset and many of the differences between high and low quality posts on a basic level. Please refer to the Google Colaboratory for detailed analysis.

Probabilistic Modeling

Logistic Regression

Theory

After we normalized the data, we attempted to build a Logistic Regression model using observed features that seemed significant and mutually independent. Our model included two versions, one from first principles and the other from the sklearn machine learning library to compare effectiveness. While our model is based on basic principles of Logistic Regression, we decided to use an unorthodox optimiser. Normally, the optimiser used to complete the logistic regression model is gradient ascent. However, we decided to challenge ourselves to use the Newton-Raphson Method, a common calculus method that uses second partial derivatives to determine the point of convergence.

In essence, Newton's Method is a root-finding algorithm that produces successively more accurate approximations of the roots/zeros of a real function. In its basic form, when given a function f, parameter x, its derivative f', and an initial guess x_0 for a root of f such that the initial guess is close to the real root, we can determine a successively better approximation in the following form.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Newton's method can be extended further to solve minimization and maximization problems by applying the same concept to a function's first and second derivatives. Because the derivative is zero at a minimum or maximum, local minima and maxima can be determined by applying Newton's method to the derivative function.

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

This, in essence, will be the foundational backbone of our logarithmic regression model.

Mathematical Derivation

In deriving a mathematical model for the linear regression, we begin by first expressing the likelihood function as a maximum-likelihood estimator of our binary classifier according to the following

$$\Pi_{i=1}^{m} \sigma(\theta^T x^{(i)})^{y_i} * (1 - \sigma(\theta^T x^{(i)}))^{1-y}$$

We can then take the logarithm of this expression to simplify it.

$$\Sigma_{i=1}^{m} y_i * log(\sigma(\theta^T x^{(i)})) + 1 - y * log(1 - \sigma(\theta^T x^{(i)}))$$

From this point, our course introduced the classic gradient ascent optimiser as the method in determining the parameter to maximize this expression. For this project, we attempt to use a multi variable Newton's Method instead. Newton's Method is a viable optimizer, as the method essentially finds the root of the first derivative of our maximum log likelihood estimator. We recognize that the maxima must occur when the first derivative is 0, so we are just finding the parameter values where this occurs.

According to our mathematical sources below, the second derivative of the our log likelihood is determined by the following expression

$$-\Sigma_{i=1}^{m} x_i x_i^T y_i * \sigma(\theta^T x^{(i)}) * (1 - \sigma(\theta^T x^{(i)}))$$

Since we can now calculate second derivatives across our parameters, we then calculate our Hessian Matrix, which in our case is a 4 by 4 matrix (for the 4 features we are using). We will now refer to this Matrix as H.

Therefore, we can now update our thetas using the standard form on Newton's equation.

$$\theta^{new} = \theta^{old} - H^{-1} * \frac{\partial L(\theta)}{\partial \theta}$$

Naive Bayes Classification

Theory

A major probabilistic model that we developed from our dataset analysis is a modified Naive Bayes Classifier based on document term frequencies. For this project, we decided to develop our own model of Naive Bayes from scratch without the use of any existing machine learning libraries. The foundations of our model is based on Bayes' Theorem of calculating the posterior probability of a given parameter, and inferring the value of the parameter that maximizes the posterior.

$$posteriorprobability = \frac{conditionalprobability * priorprobability}{evidence}$$

Given a feature vector sample X containing n features, and class scalar Y with m distinct classes, we can determine the general formula for the posterior probability to be

$$P(Y_j|\vec{X}) = \frac{P(\vec{X}|Y_j)P(Y_j)}{P(\vec{X})}$$

We want to find the scalar value for Y such that the posterior probability is maximized

$$arg_{Y_j}max(P(Y_j|\vec{X})) = arg_{Y_j}max(\frac{P(\vec{X}|Y_j)P(Y_j)}{P(\vec{X})})$$

Mathematical Derivation

In expanding the above argmax notation, we first factor out the denominator value because it acts as a constant value and will not be significant for our purpose of maximizing the posterior.

$$arg_{Y_j}max(\frac{P(\vec{X}|Y_j)P(Y_j)}{P(\vec{X})}) = arg_{Y_j}max[P(\vec{X}|Y_j)P(Y_j)]$$

This means we will only be concerned with finding the prior probability (probability of specific classes of Y) and the conditional probability.

Prior Probability

Finding the probabilities of each class of Y is a simple computation that divides the number of a specific class (HQ, LQ_EDIT, LQ_CLOSE) of a body text by the total number of body texts.

$$P(Y_j) = \frac{\Sigma d \in Y_j}{\Sigma d}$$

Where we denote d as the number of body texts in the training data.

Conditional Probability

When finding the conditional probability portion, we utilize the property of Naive Bayes by considering the joint conditional probability as a product summation of individual feature conditional probabilities. As such, we treat each individual feature as conditionally independent from one another.

$$P(\vec{X}|Y_j) = P(X_1, X_2, \ldots, X_n|Y_j) = \Pi_{i=1}^{n} P(X_i|Y_j)$$

From here, the individual conditional probabilities for each feature can be determined via maximum-likelihood estimate, which is a frequency in the case of categorical data

$$\hat{P}(X_i|Y_j) = \frac{N_{X_i,Y_j}}{N_{Y_j}}$$

where the numerator is the number of times the feature appears in samples from the class Y_i.

For text classification, an alternative approach to the maximum-likelihood estimate is given by term frequency (Raschka). Term frequency is defined as the number of times a given term appears in a document d, also defined as raw frequency. The sum of raw frequencies can then be used to compute maximum-likelihood estimates from training data.

$$\hat{P}(X_i|Y_j) = \frac{\Sigma tf(X_i|d \in Y_j) + \alpha}{\Sigma N_{d \in Y_i} + \alpha * V}$$

where the numerator is the sum of raw frequencies of a term X_i from a document in class Y_i plus alpha, a smoothing constant.

The denominator is the sum of the total number of documents in class Y_i plus the product of alpha and the size of dataset vocabulary (length of all terms in the dataset). We also note that in this case we implemented Laplace smoothing, causing our alpha to be 1.

$$\alpha = 1$$

When tested using a separate sample of 100 body texts provided by the dataset, the model was able to score 71% of the valid trials accurately, a promising result.

Text Generation with Markov Chains

As an extension of our work, we created a rudimentary text generator using Markov Chains. These structures are basic in that for each set of unique N-words (or N-grams), we store all possible next words – and to make our predictions more realistic, these next words have probabilities associated with them that correspond to the probabilities found in the real text; we just end up with an array of probabilities where each word has a probability of number of times it follows each N-gram/total number of words that follow this N-gram. Because these texts are memoryless and are based purely on probabilities, we mostly expected the application to return some ridiculous paragraphs.

An interactive demo, along with more detailed examples of bodies and titles we generated can be found on the Colaboratory notebook. In general, we found that larger values of N produced slightly more coherent output. We have pasted some output below:

> *Our Favorite:* "Sorry for the noob question.I am trying to validate access tokens against at_hash. Token header is like thisI have a repo with various components and I have 2 issues:Currently we try to use pm.sendRequest. For example :I have a pre-existing dynamo db table to which i want to have all lint warnings"
>
> *Example 2:* "I have a monaco code editor embedded in my app.Although inline css is not recommended but I just want to be able to create a library which makes declaring tasks with similar settings less verbose, for instance.I am trying to use the @blueprintjs/core library in my project. However, when I compile my code,"

We recommend playing around with the Markov Text Generator application available in the Collaboratory notebook. It is quite amusing to say the least.

Conclusions

Discussion and Insights

We preface this section that our results are largely available at the end of each section in the Colaboratory notebook. Instead of discussing each finding, we believe it is easier for readers to proceed with the demo in the Colab notebook.

In general, we found that our p-values for the difference in sample means for our features were significant, and each feature was independent. This allowed us to confidently make a fairly accurate Logistic Regression model (~58%). Our Naive Bayes model was arguably the more challenging model due to the mass computation power required for text classification and the data integration, leading to outsource of computation on local machines. However, we were able to attain a decently accurate model (~71%). As for the Markov Chain text generator, we were able to draw surprisingly humorous and somewhat coherent simulated posts.

Limitations

Upon reflection, we realize there exist many limitations of our project. Although these don't warrant significant concern, we believe that their solutions will lead to more substantial and accurate results. Firstly, a significant limitation of our Naive Bayes model is that it occasionally suffers from computational underflow. Due to the nature of the mathematical processes behind Naive Bayes, the algorithm may work with a number that is smaller than 10 to the power of -320, the lower limit of which our program can describe a number to a degree of accuracy. If a number is beyond this threshold, the program will simply evaluate it to 0, resulting in loss of significant information. Often, this computational underflow will result in an indeterminate parameter and the trial will have to be invalidated. We attempted to remedy this limitation by applying a logarithmic scale. However, the model was significantly inaccurate without the scaling. Unfortunately, we did not have enough time to investigate the issue, and we leave solutions for this problem for the future. We speculate that if a solution to underflow was implemented, there potentially would be an increase in the model's accuracy rating.

For the logistic regression model, we realised that our predicted accuracies are relatively low (~55-65%). Although not ideal, we do not believe that the issue lies with our regression algorithm; rather, it appears to be an issue with our naive approach of predicting a complex result, that may be influenced by a multitude of unknown nuanced factors, with a few basic quantitative parameters. We speculate that if we could extract more information pertaining to each post, we can understand the greater relationship between features that impact the final result and accuracy.

Another notable limitation of our project is that a major portion of our work has been done using the smaller dataset containing 60,000 posts. As of writing, Stack Overflow has 20,416,967 questions posted (https://stackoverflow.com/questions), meaning we have only evaluated roughly 0.3% of the total population of posts. This limitation implies that we would have a better estimate of

the overall population of Stack Overflow questions if we took a larger sample size. Lastly, we want to address the limitation of the amount of time our group had to work on the project. We recognize that the project duration crossed with major events and holidays --such as the 2020 election and Diwali-- that may have reduced the amount of time we could dedicate to the project. If given a larger time window, our group could have spent more time refining and exploring our results.

References

Moore. (2020, October). 60k Stack Overflow Questions with Quality Rating. Retrieved November 15, 2020, from https://www.kaggle.com/imoore/60k-stack-overflow-questions-with-quality-rate.

Raschka, S. (2014, October 04). Naive Bayes and Text Classification. Retrieved November 15, 2020, from https://sebastianraschka.com/Articles/2014_naive_bayes_1.html

Stack Overflow. (2020, October). StackSample: 10% of Stack Overflow Q&A. Retrieved November 15, 2020, from https://www.kaggle.com/stackoverflow/stacksample

Further references available in the Colaboratory file.