CS 498 Midterm Report (ADV17)

By: Jackson DeDobbelaere, Devashri Nagarkar, Hanna Parker

Introduction

The goal of our project is to find the reputations of users on stack exchange, based on the content they're creating, to ultimately decide who is a trustworthy source. Stack exchange has a lot of users posting tons of content, so it is important know their reputation to figure out if someone's answers to questions are credible. This is a very challenging problem because we cannot go hand-check the validity of every answer on stack exchange so we will need to figure out a user's reputation programmatically. Since we can't programmatically figure out if a post is correct or not, we have to find another way to measure reputation. After extensive research, we have decided to use sentiment analysis, specifically using the aspects "polarity" and "subjectivity", as a measure for reputation. To accomplish this goal, we have determined the sentiment of piece of content, created feature vectors of this sentiment data, created a K-Nearest Neighbors object to classify the data into one of three reputations, and did kfold splits to determine the accuracy of our classifier.

Related Work Survey

The first paper that we chose to review was *Predicting the Quality of Questions on Stackoverflow* by Antoaneta Baltadzhieva and Grzeorz Chrupala of Tilburg University. They noted the importance of attempting to determine the quality of a user, but decided to analyze the quality of questions in this paper. For this reason, they use the user's reputation as a variable in

determining the quality of the question. In addition, they took into account question tags, length, code snippets, and lexical terms. As a note, Baltadzhieva and Chrupala did not use machine learning to gather the user's reputation; but, instead used Stackoverflow's user reputations as they are determined by the upvotes a user gets on their questions.

The main study was focused on how questions features can be used to predict the number of answers as well as the question quality score. For each of the features mentioned in the previous paragraph, independent features were tested individually on a set of 1.7 million questions on Stackoverflow. They took the terms with the highest and lowest coefficients and ran sentiment analysis on them to determine the terms that would derive quality of a question as well. They categorized good terms as ones that have strong positive or negative connotations and bad terms as ones with off topic terms, spelling errors, or grammatical errors, such as interjections within a term.

The major strengths of this paper is that the authors decided to recognize real world obstacles that could obscure the results. For this, they mentioned that the two characteristics that they were attempting to measure (number of answers and question quality) may not be correlated. This may be due to various factors such as no one actually being able to answer the question without the required skillset of knowledge; however, it could get a great question score because the question sparks curiosity throughout the Stackoverflow community.

The reputation of the users answering the question is also taken into account as it is almost seen as an endorsement if a user with a high reputation chooses to answer a specific question, much like how the page rank algorithm works. Tags are analyzed because they found that a large number of unanswered questions are all related to one another with the same tag(s)

because there are simply not enough experts on those topics to answer them. In addition, they concluded that length of question is inconsistent with their study because they found that short questions are either answered quickly or too short to be meaningful. On the contrary, a long question could lead to an expert answer or a question that is too time consuming to full understand and craft a response for.

The user reputation of answers was found to be one of the most dominant terms in determining question quality. This page rank type algorithm is what Google was founded on and it is no surprise that this would be the most distinguishing feature when determining the quality of a question on Stackoverflow.

This study chose not to analyze the content of code snippets because that is a whole other can of worms and hard to infer if the code snippet is bad code or good code. Therefore, the presence of a code snippet is represented as a boolean value. Stop words are also filtered out of the questions bodies before Natural Language Processing is ran on them.

To calculate the correspondence between question score with all the question features and the correspondence between number of answers with all the question features, linear regression was used with each of the question features as independent variables in the linear function. We also liked how these authors noted that linear regression was a good choice because they can account for highly variable estimates within the data. They decided to use Ridge Regression to avoid overfitting as the data set they were analyzing was extremely large.

One weakness they had was splitting the training set at first 60%, validation set as next 20%, and test set as last 20%. We think that k-fold validation would have been a lot better and would even more ensure that their model is not being overfit.

As a result, they found that combining all the independent variables was a better predictor than any subset combination of them. For question quality linear regression, they obtained a mean squared error of 4.622 and a mean absolute error of 1.286. For the number of answers linear regression, they obtained a mean squared error of 2.514 and a mean absolute error of 1.163.

Baseline Implementation

To help classify the users into the three categories of reputation based on their posts, we used K-Nearest Neighbors. This algorithm maps each data point by each parameter and then finds the K nearest points to a test data point and then takes the label of the most popular one. Before we could find the distance between all the test points to all our train points, we had to pre-process the data. We decided to condense the text description parameter into a numeric data point by getting its sentiment analysis. We chose this because sentiment analysis is often used to predict reputation for users or posts. We also considered word count and forming a tf-idf vector as well. We choose sentiment analysis mostly because we were curious about how well it would act as a parameter but we plan on experimenting with the other two or some combination of the three for our final implementation. Sentiment analysis is an extremely useful tool for measuring the public's opinion for a specific social media post, which would help us derive ratings from our model. Sentiment analysis works by using either using a General Inquirer Dictionary that correlates emotional connotation scores with words and phrases or with underlying Machine Learning to train algorithms to predict correct emotional connotation scores with the other posts that have emotional connotation scores and that are most similar to them.

Our sentiment analysis computes the polarity and the subjectivity of the text description. We also changed the content type column to a numeric 1-5 to keep everything consistent for our distance calculation. In the end we have a 2D matrix with the columns as [polarity, subjectivity, contentTypes, userToReps] with userToReps as the reputation label. Since the data file was so large, we saved it as a CSV so that instead of recomputing the data each time we ran it, it will simply reload from the CSV file. In order to get the userToReps we had to create a dictionary of the user of each post to their reputation by looking it up in the dictionary that we created for the labels_train file.

Our actual algorithm for K-NN was written from scratch using numpy and scipy. The train function just takes in the X_train and y labels and stores them. The predict function calls a distance function to get the distance from every X_train point to the X_test point. If X_train had dimensions N by D where N is the number of data sets and D is the number of parameters, and X_test had dimensions M by D where number of data sets and D is the number of parameters, then the distance function returns an N by M matrix of the distance between every X_train and X_test point. Then our predict function gets the K closest X_train points for every X_test data point and gets the most popular y label for it. The code has been optimized by completely vectorizing it by having no for loops. All of the arrays are numpy arrays to speed up the computing process.

Interpret the Results

Once we had the matrix with all of the data, we had to separate it into test and train sets to perform the accuracy test of our algorithm. Originally we just did a 80%-20% split for our

train and test but then we decided to go with a K-fold validation. This was done by simply importing the K-Fold library from sklearn. We then took the average of the accuracies for every iteration of the k folds as the final average accuracy. For the baseline, our K number of splits in K fold validation will be 5 splits. We varied the K in the nearest neighbors, treating it as a hyper-parameter to find the K that gives the highest accuracy. The following table represents the average accuracy for its corresponding K.

K	Average Accuracy given 5 splits
1	77.3744
3	85.3271
5	86.8709
10	87.6643
50	87.8354

As K increased, the accuracy increased as well. This makes sense because it considers a larger number of possible points and uses their labels as votes. The fact that the accuracy is increasing means that so far the parameters with certain labels are closely clustered together. It is important not to let K get too big, otherwise the wrong labels (ones much further away) will get more and more votes to swing the prediction. For the final report we will check against a larger range of K.