Clayton Bass | **Mathematical Data Science Major**
Originally for Prof. Vosoughi's COSC 89.21 (21W) | **Modified for the Culminating Experience**

Predicting StackOverflow R Question Binary Ratings from Question Text

Abstract

Binary classification problems, where our goal is to accurately place an object of interest into one of two class labels, are relatively simple to understand but can become very challenging depending upon the data used and question(s) addressed. In particular, unstructured text data tends to be cumbersome because of the innumerable ways of processing said data into features (and thus this has become a rapidly growing area of interest in the machine learning community). Plenty of thought has gone into formulating efficient and effective ways to featurize such a variable, messy type of data into a format that modern machine learning algorithms can understand. To continue this exploration, this paper aims to determine whether the binary rating (positive or negative) of StackOverflow R questions can be predicted based solely on the text data of each question (i.e., on lexical data alone). Therefore, in this project, my basic hypothesis is that StackOverflow R questions *can* be predicted as having a positive or negative score via a standard binary classification algorithm. The results of the paper support this hypothesis, as our optimal model beats out baseline models (which use random guessing) by about 7.5% in accuracy.

Introduction

Within the broader context of machine learning and classification problems, binary classification is remarkably important and has applications ranging from medicine to sports analytics to the stock market. A binary classifier is a machine learning model that takes in numerical features of some object, and outputs one of two possible classes for that object. For example, if we want to predict whether it will be rainy or sunny on a given day, we may have our machine learning algorithm learn from numerical inputs like temperature, wind speed, season, or month of the year, and then predict the day's weather.

Binary classification problems are relatively straightforward to understand but are not necessarily easy to perform well depending on the specific questions that are being addressed. With certain data formats—such as raw text data, particularly containing code or other unusual objects or data structures—it can be challenging to achieve high accuracy scores due to the very messy raw data format, where there is rarely an obvious way of transforming the data into a numerical format that a standard machine learning algorithm can easily understand. StackOverflow question data in particular has not been appreciably explored in the machine learning or natural language processing literature. Beyond the very specific application of this project, the process of cleaning and featurizing very unclean text data (in this case, data that has HTML tags and code snippets embedded within) and creating a baseline-beating machine learning model is a valuable process for machine learning specialists in all fields. For example, in medicine, patients may write very complicated questions on the patient portal, and it could be of value to predict whether or not the questions are urgent based on the text; in stock market prediction, there could be very unclean Reddit comments or financial reports that indicate some feature of a stock, and it could be worthwhile to predict whether this information could

be signaling a positive or negative impact on the stock's price. Opportunities for further exploring such unstructured text data abound.

In this particular project, I clean the question data with the goal of creating meaningful textual features like term frequency-inverse document frequency (a.k.a. TFIDF). Per a journal paper by Qaiser and Ali from 2018 on the use of TFIDF in text mining, the purpose of the technique is to capture words that are highly specific to a given passage of text (Qaiser and Ali 25-26). To explain further, TFIDF for a given word is the product of term frequency (TF) and inverse document frequency (IDF) for that word; term frequency is just the number of times a term shows up in a given passage (in this case, an R question), while IDF is analogous to the reciprocal of the number of questions that word shows up in. The reason for IDF is that if a word shows up in very many questions, then it is likely not important to any specific question, so TFIDF gets lowered by the IDF score, while if a word shows up in very few questions, then it is likely important for the given question. So, if we use TFIDF scores as features, then we may be able to tease out specific words and phrases that are more common to "good" questions (with positive scores) as opposed to "bad" ones (with negative scores). When referring to CountVectorizer in later steps, or the "bag-of-words" approach, I am simply referring to the TFIDF score without the IDF weighting mechanism (so we are looking only at raw term frequency counts for words); the purpose here is similar to TFIDF, but we do not have the added mechanism of under-weighting words that show up across many documents (S et al. 320-321). (The citation here refers to a paper by Deepu S. et al. relating to the use of the CountVectorizer/bag-of-words approach in text analytics and mining, which builds in some way on the paper by Qaiser and Ali above.)

Each StackOverflow question about the R programming language has an integer score. When a question is posted, it starts at a score of 0. Any user on the platform can upvote or downvote the question by a score of ± 1. For example, if 10 users upvote the question and 4 users downvote the question, then the question has a score of $10 - 4 = 6 > 0$, which is in our "positive" class (if the overall score were a negative integer, the question would be in our "negative" class). My hypothesis is that the direction of the overall rating (positive or negative) of StackOverflow R questions can, in fact, be predicted solely from the text content of the question. Performance is judged strictly based upon the weighted average f1-score, which itself is essentially an average of two quantities: precision and recall, which is explained much further in a paper by Lipton (Lipton et al.). We define precision as the proportion of questions we place into a given class that are actually in that class; we define recall as the proportion of questions truly in a given class that we have accurately predicted; and f1-score combines these two metrics into a unified view of performance that better accounts for class imbalance than accuracy (which can get dominated by the majority class if major skew exists in the data) (Lipton et al.). The best model was able to consistently earn about a 7.5% improvement in f1-score over baseline random-guessing models, which is appreciable and indicates that our models are teasing out significant patterns in the data.

Methods

In order to procure the R question data, I downloaded the Kaggle data file 'Questions.csv' from the following link: `https://www.kaggle.com/stackoverflow/rquestions?select=Questions` ("R Questions"). The raw data file is easily read as a table in Excel, contain-

ing the following columns: (1) unique question Id as an integer 'Id', (2) date and time of the question being posted as 'CreationDate' in YYYY-MM-DD format, (3) question score as an integer 'Score', (4) question title as a string 'Title', and (5) question body as a string 'Body'. There are about 189000 unique records in the original dataset.

First I shall discuss initial data preprocessing and cleaning. I converted the 'CreationDate' column into a pandas datetime object, converting the index of the dataframe into the dates so that years can be extracted easily. Since I am most familiar with the more recent versions of R, I extracted only those questions from the dataframe from the years 2016 or 2017, and then removed all questions with a score of zero (since it is unclear whether they are regarded as good or bad questions by the StackOverflow user audience). 40736 samples remained.

I then removed all of the HTML and LaTeX tags present in the data (which weren't readily visible just by looking at the question text on the website; one would have to inspect the website's code in order to actually see the tags). I removed all code tags in the following list: <code>[...]</code>, <pre><code>[...]</code></pre>, <blockquote></blockquote>, <href=[...]>[...], [...], [...]. I also removed all code within the tags. Some code tags remained for some reason, so then I removed these tags: , , </pre>, <pre>. I replaced all of the aforementioned tags with a blank space. I removed code itself since the feature generation methods I will discuss later in this section (TFIDF and CountVectorizer) work well only with English words, not with R function/command names. I then applied this removal mechanism function to every entry in the 'Body' column.

I then converted the 'Score' column into 1 if strictly positive, and 0 if strictly negative, in a new column called 'score_binary'. The data appeared imbalanced, with 35561 records in class 1, and 5175 records in class 0, so I decided to downsample class 1 to make the classes more balanced, since imbalanced classes can cause the binary classifier to become biased toward the majority class since it has many more majority examples to learn from (which may make our classifier less accurate). Using the .sample() method in pandas, I sampled 0.5 of the positive class in the dataframe, and then rejoined the negative and remaining positive data in one dataframe. I then split into train and test sets, using the default ratio of 75% training and 25% testing.

I then pursued two different variations of data cleaning: a complete data cleaning, and a partial data cleaning (to see which one yields better performance).

For the full cleaning: I clean the 'Title' column by lowercasing it and removing all words from each title that are in nltk's set of English stopwords (via nltk's guide on their stopwords, found at `https://www.nltk.org/book/ch02.html`) ("Accessing Text"). I then remove all punctuation from every title. I finally apply nltk's WordNetLemmatizer to every word in the tokenized title (i.e., where each word in the title is separated into a list) in order to turn all nouns into singular form and to turn all verbs into the same tense (present). Finally, I return a string that is the cleaned title. Now, I clean the 'Body' column. For each question's body, I first tokenize the body, remove all stopwords in the same set as above, lemmatize the tokenized words, and return a string with the entire body text. I then concatenate the 'Title' and 'Body' columns into a new column called 'text_cleaned'.

For the partial cleaning: Now, we clean the data a little less thoroughly. I clean the titles only partially by lowercasing, removing stopwords, and removing punctuation from the set above. I clean the question bodies only by removing stopwords and removing

the punctuation from the set above. I then keep a separate dataframe that houses the concatenated cleaned title and body text for each question.

Now, I shall discuss my approaches to feature selection and modeling. We will be comparing CountVectorizer and TFIDF features, as discussed previously, and performing 5-fold cross-validation in order to best assess the predictive power and performance of our models. 5-fold cross-validation means that we will split our training data into five equally-sized pieces – each one of those pieces will serve as the test set in one iteration of training (while the other four pieces serve as training data), and the average performance over all five splits will be calculated; a 1995 conference paper by Kohavi gives much deeper explanation of the power of this methodology (Kohavi 1-2). The parameters we test for CountVectorizer are n-gram range of (1,2) and (1,3) (where an n-gram is a collection of n adjacent words to be considered as a feature – sometimes collections of words are more meaningful than just individual ones) and minimum document frequencies of 5 and 10 for each term (since we want to ignore some words that are extremely rare in our database of questions, as they may not mean anything significant or simply may be misspelled). The parameters we test for TFIDF are analogous to those used for CountVectorizer.

We will then be applying a random forest classifier and logistic regression classifier, and searching for optimal parameters for each of these models. A random forest classifier is a collection of decision trees (each of which learns how to split the data along certain feature values); this model is particularly effective since it doesn't make crucial distributional assumptions about our data and does not require us to scale any of our features – a 2001 paper by Leo Breiman gives much deeper information on the theory of the classifier (Breiman). It also runs quite quickly. A logistic regression model is a binary classifier that learns optimal parameters β to model the sigmoid equation $y = \frac{1}{1+e^{-\vec{\beta}\cdot\vec{x}}}$ where $\vec{x}$ is our vector of independent variables (our TFIDF or CountVectorizer features). In a similar manner to the random forest classifier, while logistic regression is a linear model (and so does not deal with non-linear data in a manner nearly as effective as a random forest classifier), it has very few distributional assumptions and is computationally efficient, making it a robust, apt choice for our large dataset.

The parameters we test for our random forest classifier include a range of estimators from 50 to 100 to 200 (which is the number of decision trees to consider in the forest–if we use more decision trees, we may get a more generalizable result), and a maximum individual decision tree depth of either 5, 10, or None (sometimes cutting off trees at smaller depths can prevent overfitting to the training data). The parameters we test for our logistic regression classifier include a regularization parameter ('C') of 0.1, 1, or 10 (a higher penalty permits overfitting to the training data and allows for a model with higher variance), and penalty options of l1 (lasso regression, which sometimes reduces model coefficients to zero and prevents overfitting) and l2 (ridge regression, which shrinks coefficients and prevents overfitting).

These models will be evaluated against baseline dummy models. The reason for this is that if our models are doing worse than dummy models that use naïve methods like random guessing (or do only as well as the dummy models), then we will know that our models are not learning any significant patterns from the data. The optimal baseline dummy model is essentially a weighted random guesser: it guesses the class prediction randomly according to the distribution of the classes. For example, if our classes were 75% positive and 25% negative, the baseline dummy classifier would randomly assign positive class to a question with 75% probability, and negative class with 25% probability. The

other kind of dummy model used is simply a completely random guesser (assuming a 50-50 guess), but this model consistently performs worse than the weighted random guesser in our situation.

For all of the aforementioned models, the f1-score is calculated as the performance metric of the model. As mentioned before, f1-score is essentially a mean of precision and recall; the f1-score is calculated on each class (negative and positive), and the average is taken to summarize that model's performance. We test every possible combination of features, level of cleaning, and classifier choice (including every possible combination of classifier parameters), and we see how much better the f1-score from this optimal model is on the test set compared with the f1-score from the optimal dummy model predicting on the test set.

Results

The optimal model as determined through 5-fold cross-validation used the following combination of preprocessing and modeling steps: (1) partially cleaned data; (2) TFIDF features, with a minimum document frequency of 10 and an ngram range of (1,2); and (3) a logistic regression model with ridge penalty and a regularization strength C of 10. This model achieved an average f1-score of 0.72 in training across all five folds of the 5-fold cross-validation.

The optimal dummy model, given by the weighted random guessing model described previously, achieved a maximum f1-score score of 0.67. So, in training, our optimal model is about 7.5% better than the best dummy model.

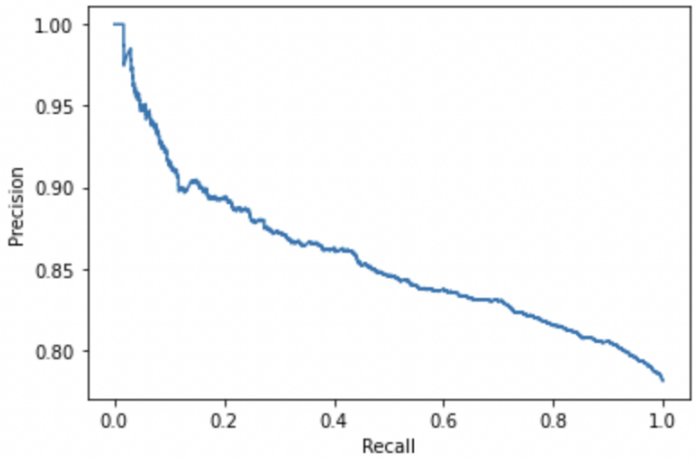
In order to truly validate our results, we predicted on a hold-out test set (which was entirely unseen during training) using both our optimal model and the best dummy classifier. Our optimal model achieved an f1-score of 0.72 on the test set, and the best dummy model achieved an f1-score of 0.67 on the test set, representing yet again a 7.5% improvement of our own model over the dummy model.

Discussion

Given the improvements in our optimal model mentioned above, we significantly beat the baseline models. Therefore, this addresses our hypothesis that one can predict the binary rating (positive or negative) of StackOverflow R questions strictly from the question text data. Although the model may have a higher ceiling with alternative hyperparameter tuning or feature extraction methods to better account for variations in the way that the code is presented in each R question, the fact that we are beating baseline dummy models consistently (and the best baseline model possible, at that) illustrates that our model is learning some significant patterns from the data that can separate positive questions from negative ones; in other words, it is quite clear that we are not modeling random noise.

Furthermore, here is the precision-recall curve generated on the test set predictions:

This was done using the precision_recall_curve function in sklearn, using the test data labels and the test data predictions. Notice that the precision still remains high (above 0.7 uniformly, and mostly above 0.8) on the y-axis even as the recall shifts from 0 to 1. Therefore, out of all of the questions classified as positive questions, even as we increase the recall (which is the proportion of truly positive questions that are classified as positive) to 1, the proportion of the positively-classified questions that are truly positive remains high.



Conclusion

There appears to be much potential to feed unstructured, messy, relatively emotionally neutral text data—even data that contains code snippets and rather annoying tags—into machine learning models with proper feature engineering. Although the ceiling for this particular problem has likely not yet been reached, it is promising to see that with some amount of feature engineering and model tuning, there is still consistently an appreciable gain over baseline models when trying to make predictions of question rating (as positive or negative) strictly from the text data. An interesting further direction to take this project with more time would be to examine how the actual code snippets themselves within the questions could be of any predictive value. In addition, exploring other kinds of models would possibly elevate the performance here. However, the hypothesis has been successfully addressed. Clearly, the findings are that there *is* a way to predict question rating positivity or negativity strictly from the question text, using lexical features (and in particular TFIDF and CountVectorizer features, and relatively simple models like the linear model LogisticRegression).

With regard to future research in this area, it is also useful to consider that rather emotionally neutral text data (as in the case of a StackOverflow question about R programming) is very different from a news article or a product review, where perhaps it is easier to extract sentiment from the text and to generate positive and negative classes more easily. Text classification problems that do not focus cleanly on sentiment or category prediction (such as predicting the content category of a news article) seem to be more intractable in this regard, at least from an intuitive standpoint, and so the problem of predicting the quality of some kind of object (whether it be an R question, a product itself, or some other object of interest being described) with limited sentiment and content data is a crucial field of further exploration given the difficulty of this problem.

Works Cited

"Accessing Text Corpora and Lexical Resources." Natural Language Toolkit, `www.nltk.org/book/ch02.html`. Accessed 12 Oct. 2021.

Breiman, Leo. "Random Forests." Statistics at UC Berkeley, UC Berkeley, Jan. 2001, `www.stat.berkeley.edu/~breiman/randomforest2001.pdf`. Accessed 12 Oct. 2021.

Kohavi, Ron. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation

and Model Selection." International Joint Conference on Artificial Intelligence, 1995, pp. 1-7, ai.stanford.edu/~ronnyk/accEst.pdf. Accessed 12 Oct. 2021.

Lipton, Zachary C., et al. "Thresholding Classifiers to Maximize F1 Score." arXiv.org, arxiv.org/pdf/1402.1892.pdf. Accessed 12 Oct. 2021.

Qaiser, Shahzad, and Ramsha Ali. "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents." International Journal of Computer Applications, vol. 181, no. 1, July 2018, pp. 25-29. IJCA, www.ijcaonline.org/archives/volume181/number1/29681-2018917395. Accessed 12 Oct. 2021.

"R Questions from Stack Overflow." Kaggle, www.kaggle.com/stackoverflow/rquestions?select=Questions. Accessed 12 Oct. 2021.

S, Deepu, et al. "A Framework for Text Analytics Using the Bag of Words (BoW) Model for Prediction." International Journal of Advanced Networking Applications, pp. 320-23. IJANA, www.ijana.in/Special%20Issue/S71.pdf. Accessed 12 Oct. 2021.