Final Project

COMP 562-001

Nov. 21, 2020

Classification of Stack Overflow Questions

Denny Cmiel, Jason Moxley, Randy Sievers, & Caitlin Vickery

Project Overview

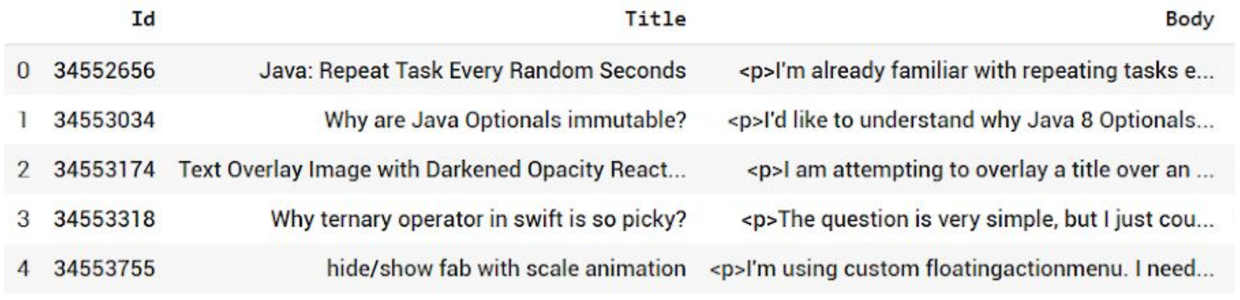
Stack Overflow is an extremely popular go-to resource for both professional and beginner programmers. Since its creation in 2008, this online community has over 20 million questions, and this number grows daily, making Stack Overflow the largest question-and-answer site for the wide range of topics that make up computer science.

Stack Overflow posts can have two states: open or closed. Closed posts are questions that have been determined to be low quality which can no longer be answered by users. Open posts are questions that have been determined to be high quality and which remain open to user interaction. Stack Overflow gets an incredible amount of new questions each day and predicting the quality of new posts and recommending high quality posts is critical to maintaining a high quality of experience for Stack Overflow users. One way users can interact with posts is to either upvote or downvote each post, which we will call the post's score. Thus, posts with a positive score tend to be high quality posts while those with a negative score tend to be of low quality.

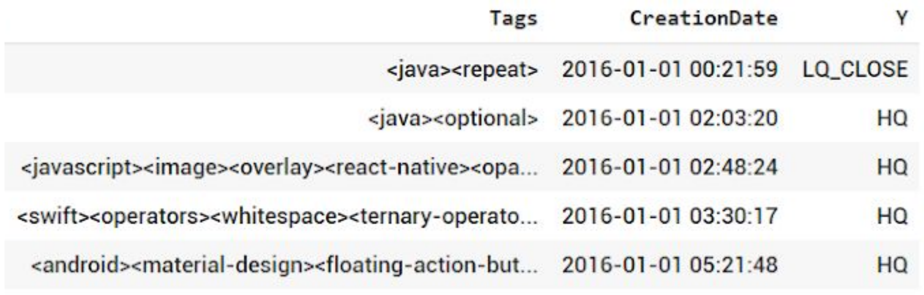
The aim of this project was to utilize machine learning models to implement a prediction system that predicts the overall quality of Stack Overflow posts based on a dataset of 60,000 Stack Overflow posts that have been labeled as high quality or low quality.

Dataset & Features

Our dataset comes from Kaggle and is titled, "60k Stack Overflow Questions with Quality Rating." [1] We used pandas to import the csv files to pandas data frames. [8] The dataset was split into 45k posts for training and 15k posts for validation. The dataset is organized such that each question has a unique identification number, the question title, the question body, tags, creation date, and a label ("Y" in the sample table below). Each Label has three possibilities: HQ, LQ_EDIT, LQ_CLOSE. HQ is for high quality posts with more than 30 upvotes and no edits. LQ_EDIT is for low quality posts with a negative score that remain open after several edits. LQ_CLOSE is for low quality posts that have been closed. [1] The figure below shows the first 5 rows of the training data with these attributes.



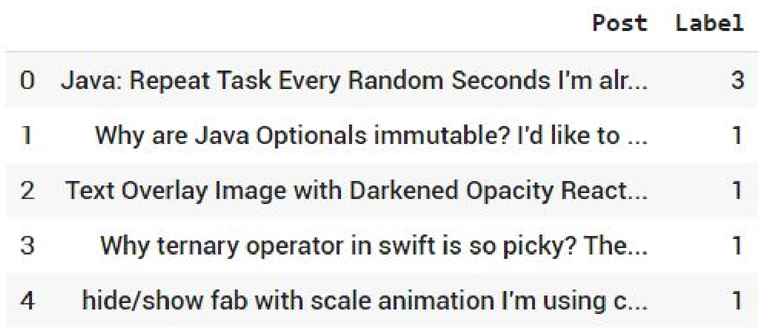
	Id	Title	Body
0	34552656	Java: Repeat Task Every Random Seconds	<p>I'm already familiar with repeating tasks e...
1	34553034	Why are Java Optionals immutable?	<p>I'd like to understand why Java 8 Optionals...
2	34553174	Text Overlay Image with Darkened Opacity React...	<p>I am attempting to overlay a title over an ...
3	34553318	Why ternary operator in swift is so picky?	<p>The question is very simple, but I just cou...
4	34553755	hide/show fab with scale animation	<p>I'm using custom floatingactionmenu. I need...



Tags	CreationDate	Y
<java><repeat>	2016-01-01 00:21:59	LQ_CLOSE
<java><optional>	2016-01-01 02:03:20	HQ
<javascript><image><overlay><react-native><opa...	2016-01-01 02:48:24	HQ
<swift><operators><whitespace><ternary-operato...	2016-01-01 03:30:17	HQ
<android><material-design><floating-action-but...	2016-01-01 05:21:48	HQ

For preprocessing, we first decided that the Id and CreationDate of each post provided no useful information in determining the quality of the post. Thus, we dropped those columns from our dataset. The question title is in plain text; however, the body of the question is in HTML format, which required a fair amount of cleaning up to get it into a useful format. Additionally, some body posts also contained a special <code></code> section which contained any code. We decided to convert each code section into the word "code." Next, we added spacing before and after each html tag to prevent words from not being space separated. The BeautifulSoup library was then used to convert the HTML into text, thus removing the tags. [7] To finish up formatting, excess spacing needed to be removed. In addition, the tags needed to be transformed into text so that they would just be separated by spaces.

After the body and tag attributes were cleaned up, the title, body, and tags columns were combined into one column called Post so that our prediction implementation would apply to the text inside all of those fields. The column Y was then changed to Label, and the text labels were converted into numerical categories ("HQ": 1, "LQ_EDIT": 2, "LQ_CLOSE": 3). We applied these processing methods to both the training and validation datasets. The figure below shows these changes to the same 5 rows of the training data.



	Post	Label
0	Java: Repeat Task Every Random Seconds I'm alr...	3
1	Why are Java Optionals immutable? I'd like to ...	1
2	Text Overlay Image with Darkened Opacity React...	1
3	Why ternary operator in swift is so picky? The...	1
4	hide/show fab with scale animation I'm using c...	1

Out of curiosity, we then stored all words from each post in a dictionary and printed the most common words in each of the posts separated by label. For HQ posts, the three most common words are "\n", "the", and "to". For open LQ posts, "the", "to", and "=" were the most common words. Similarly, for closed LQ posts, the most common words included "the", "\n", and "to". As one might imagine, there was an incredible amount of overlap of the most occurring words for each label. This demonstrates the need and importance of a tf-idf vectorizer discussed below. [2]

To extract features from the post, we used scikit-learn's feature extraction vectorizers. The count vectorizer implements both tokenizing, which converts each word into an integer id, and occurrence counting. [2] The tf-idf vectorizer does the same as the count vectorizer but then normalizes each word. Term frequency–inverse document frequency (tf-idf) is a numerical statistic that reflects how important a word is in a collection or corpus. [9] With the tf-idf vectorizer, the value increases proportionally to count, but it is inversely proportional to frequency of the word in the corpus. [2] Thus, in simple terms the tf-idf vectorizer places more weight on less common words and less weight on more common words. To compare the vectorizers, the training and the validation datasets were vectorized using both a count vectorizer and a tf-idf vectorizer.

Methods

The first statistical model tested in this project was multiclass logistic regression. The logistic regression model we implemented was from scikit-learn [3] We left most parameters as default but we had to increase the max iteration values to 4000 and 200 on the count and tf-idf vector data, respectively, to get the models to converge. [3] The tf-idf vector data resulted in a higher classification accuracy of 79.24%, while the count vector data was slightly lower at 77.0% classification accuracy.

Naive Bayes was the second statistical model analyzed in this report. The Naive Bayes algorithm for multinomially-distributed data from scikit-learn was implemented using default parameters also using both the count and tf-idf vectorizer. [4] The difference between utilizing count versus tf-idf vectorization was less drastic than in the previous model, with count vectors resulting in 73.7% accuracy and tf-idf vectors 73.3%.

The last statistical model investigated was a neural network, specifically, scikit-learn's multilayer perceptron classifier again using default parameters. [5] Unfortunately, training with this model took several hours, and this investigation could not be completed fully due to the compressed schedule for this report. However, using tf-idf vectors, the neural network, multi-layer perceptron classifier had a 73.03% classification accuracy.

Conclusion and Future Work

For this project our goal was to classify Stack Overflow Questions based on quality. We tried multiple classification methods and got overall successful results. The best model we tested was multiclass logistic regression trained with tf-idf vectors with an accuracy of 79.24%. If we were to keep working on this project, there are a variety of things that could improve the classification accuracy. With regards to preprocessing, we think that finding a way to process any code sections instead of replacing them would greatly improve accuracy. Additionally, we would have preferred to have had more time to experiment with data preprocessing ideas, such as just removing extremely common words. Regrettably, with regards to models, we did not have enough time to further experiment with neural networks and try to improve its performance. Either way, we believe this project was successful in its mission to classify StackOverflow Posts.

References

[1] 160k Stack Overflow Questions with Quality Rating. https://www.kaggle.com/imoore/60k-stack-overflow-questions-with-quality-rate

[2] Feature Extraction. Scikit Learn. https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[3] LogisticRegression. Scikit Learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[4] Naive Bayes. Scikit Learn. https://scikit-learn.org/stable/modules/naive_bayes.html.

[5] MLPClassifier. Scikit Learn. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[6] Murphy, Kevin P. (2021) *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[7] BeautifulSoup. https://pypi.org/project/beautifulsoup4/

[8] Pandas. https://pandas.pydata.org/

[9] tf-idf. https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Github

https://github.com/jasonmoxley/StackOverflowPredictor