INDEX

Abstract

Analysis of stack overflow data is used a proxy for analysis of a corporate intranet discussion forum to make recommendations about what type of neural network achieves the best accuracy when assigning tags to a corpus of text. Testing many configurations of the Fully Connected Feedforward Network yielded the highest accuracy (approximately 85%) when it contained a post title and post body. A confusion matrix was generated which can be interpreted to show that our tagging system was valid and does not incorrectly tag categories at an unacceptable rate.

Introduction

Corporate intranet discussion forums are a valuable tool for software development companies. The discussion forums enable employees to facilitate content creation, customer support, community-based support, project management, project collaboration, and knowledge base creation. We used Stack Overflow as a proxy for a corporate intranet forum because we could not gain access to a private company's proprietary intranet forum. Stack Overflow data can be scraped from the internet, has a question and answer section, and tags for questions categorization. Private intranet forums would be structurally different with far fewer users, but Stack Overflow is a suitable proxy to compare a question content and tagging system. Stack Overflow's robust tagging system predicts tags based on inputs and user history. For the purposes of our analysis we identify a use case where it is necessary for the tagging system to predict tags based on the question content. Sorting through the categories of these questions both internally and externally is time consuming and AI categorization based on tags would be a time saver for askers and answers. In this analysis we will test Fully Connected Feed Forward (FCFN) networks and identify the network with the highest accuracy.

Related Work

There are many examples of using Fully Connected Feedforward networks to tag a corpus of text. After completing our analysis, we identified another developer who categorized Stack Overflow, from the Data Driven Investor. The analysis used a multi-tag classification and logistic regression. They achieved an accuracy of 23

percent. For our analysis we chose to target a higher accuracy, so we reduced our analysis to 20 tags and used a and single classification.

https://medium.com/datadriveninvestor/predicting-tags-for-the-questions-in-stack-overflow-29438367261e

Another instance of where this analysis was conducted can be found in the link below:

https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568

Here, the developer has tried a similar approach by restricting the number of tags. They have also omitted the 'title' column, choosing instead to base their analysis solely on the question the user asks. Their logistic regression model trained by the doc2vec features achieved their highest accuracy of 80.45%. Our goal was to determine if including the title in the analysis improved the classification accuracy.

Models and Methodology

The data for this project was acquired from Kaggle's public big query library. The data itself if from Stack Overflow, a site where over five million users post questions about various computer science and machine learning problems to have the answered. The big query file contains sixteen different datasets such as 'posts_answers', posts_links, post_history etc. Stack Overflow already employs a method based on a users previous answers to categorize them as experts in certain programming languages. Since our goal is to classify those questions to make it easily accessible, we will be using the posts_questions dataset. The dataset has nineteen different columns, of which only four were significant in solving the problem at hand. These four are the user identifier (id), the title provided by the user (title), the question (body), and the labels (tags) under which the questions would be classified. Since this is a big query file, the data could not be downloaded like other files on Kaggle. So, we used Kaggle's Bigquery integration to query and scrape the data. Initially, we thought of using two million rows but the added accuracy lift we might have achieved did not justify the computational time and load on our laptops and we decided to use the top two hundred thousand rows. We then proceeded to query the

dataset by selecting the top twenty most frequently occurring tags and set the limit to two hundred thousand. Now that we had our dataset, we loaded it into a Jupyter notebook. Upon inspecting the dataset using pandas, and found a lot of issues with both the Title and Body (renamed as posts while querying). The first step to a successful NLP project is to convert all words and sentences to lowercase so the same words with different cases do not get treated as different words. This was done using an anonymous function.
We found that there were over 45 million words in the post column and approximately 1.8 million words in the title column. This indicated the presence of many stop words, so the next step was to remove words like 'how', 'the', 'can' etc. since these words can throw off our analysis due to their prevalence in most posts. This was achieved using the stopwords module from the NLTK package. We then observed the presence of URLs in our dataset and decide that they should be removed. We used a regular expression to identify these values and executed a simple replace function. After this, we previewed our dataset and came across some values in our posts column like '</p>' and '$' which could potentially throw off our analysis. We proceeded to use a simple replace function to deal with these. Additionally, we replaced all characters that were not white spaces or characters that we needed. Since people often make typographical errors, to further improve our analysis, we lemmatized the words. This was done using an anonymous function to apply the lemmatize function that can be found in the word module of the textblob package.
We then split the dataset into eighty percent train and twenty percent test.

Since this is an NLP problem, we figured that neural networks would probably work best. But we needed a baseline to measure our model's effectiveness. Since this is a multi class classification with more than twenty labels, using the most common class as a baseline didn't make logical sense since it would be too low. In total we ran three different models, they were:

1. Naive Bayes Classifier- This was done just to get a baseline accuracy that

we would try to beat with our more complex models.

2. Logistic regression- While it is not the most complicated algorithm out there, it has proven to be useful for classification problems such as this time, and time again.
3. Fully-connected Feed Forward Neural Networks- Initially, we discussed using a recurrent neural network with an LSTM layer but since almost all questions on stack overflow contain the name of the programming language in them, we felt it was unnecessary. We tried three different approaches while building our model:

    I. Feed-Forward Network with only the posts column.
    II. Feed-Forward Network with only the title column.
    III. A concatenated model with both columns

We experimented with different number of layers and neurons but in the end, the change in accuracy was so marginal that we decided to use a fairly simple neural network:

I. Fully-Connected with posts and title column separately: The first dense layer contained 500 neurons and the activation function used was rectified linear units. We also employed a dropout of 0.5. The input shape was our vocabulary size (max_words).
The dense output layer used had number of neurons equal to number of labels (20). This layer had a 'Softmax' activation function. The loss function employed was categorical crossentropy since we are trying to predict multiple classes along with an adam optimizer.

II. Fully-Connected Concatenated:
First we created two neural networks for the posts and dense columns, each of these has two dense layers with 100 neurons and ReLu as the activation function. The input shape is the number of rows in the training set for each of these features. The model is then combined using the concatenate function from the merge module of the Keras package. Another dense layer with 100 neurons was added to this combined model along with an output layer with number of neurons equal to the number of labels and a softmax activation function. The loss

function used was categorical crossentropy and adam was the optimizer.

In both cases, our metric was accuracy.
A validation split of 0.1 was used while training the models.

Model Results

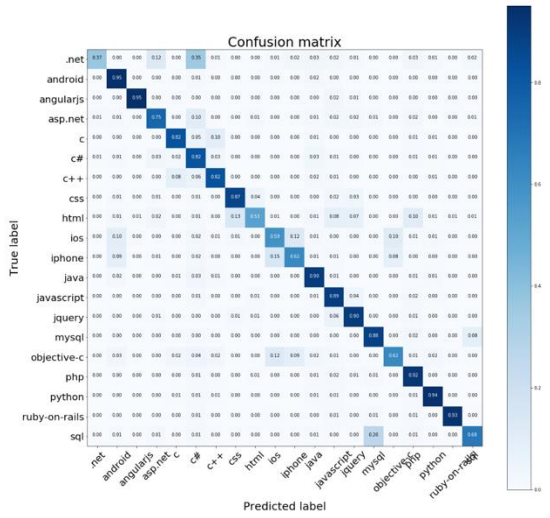
Since this is a multi_class classification problem with 20 classes, using the common class as a baseline (~12%) didn't seem to be a valid approach. Using a random guess probability(5%) was also out of the question so we concluded that the best way to asses our models would be by comparing them to each other. To establish a logical baseline, a Naïve Bayes classifier was used.

Table 1. Results of the models

Model	**Test Accuracy**
Naive- bayes	61.15%
Logistic Regression	80.55%
FCFW (title only)	68.94%
FCFW (post only)	83.295%
FCFW (both title and post)	87.75%

As depicted in Table 1, the Bayes Classifier achieved an accuracy of 61.15% The logistic regression model was tested next and there was a significant improvement. Our classification accuracy improved to 80.55%. Finally, we tested the neural networks on both columns separately and then together. The result shows that the FCFW model which used the concatenated post and title as input has the highest test accuracy, reaching 87.75%. Additionally, we can see that the NN with the post column only achieved a higher accuracy than one with just the title column. From this, we can infer that 'title' is not as good a predictor of class as the question itself. The FCFW(concatenated) model has the best performance, with a test accuracy ~7% higher than that of logistic regression and ~27% higher than that of naive-bayes. If we were to rank the accuracies of the 3 FCFW models, the post contains more information than the title, this also makes logical sense. Adding the title column as an additional input for the post will also increase the accuracy. So next we use our best model for prediction, and generate the confusion matrix.

Figure 1. Confusion Matrix Plot



From the confusion matrix it is evident that, of all the observations that were misclassified under the label 'html', 15% are in 'css', and 11% are in 'javascript', that probably because the syntax of these 3 languages have a lot of similarities. We also observe a 40% misclassification rate between 'C#' and '.net'. This is due to the fact that knowledge of c# implies some knowledge of .net. Finally, the labels 'iphone' and 'ios' are classes that easily get confused with each other, combining these two columns would likely result in a lower misclassification rate and a higher accuracy.

Limitations

We discussed attempting multi tag classifications and more complex models, but encountered some issues during the procedure. This was mainly due to our lack of computational power and programming experience.

Conclusion

Of the models that we used, the Fully connected feedforward network with both the title and post column had the highest accuracy ~85.33 percent. This is much higher than a random set with 20 classes.

Our analysis of the stack overflow data would reinforced what many have observed. Categorizing a large number of features decreases the accuracy of the network. In the case of our 20 features, the feed forward network with both the title and the post columns performed the best.

For a corporation to use a neural network to tag their corporate internet forums they would benefit greatly from having a post and body column similar to the ones that we identified. A great amount of time can be saved. To improve our analysis we would have greatly benefited from internal access to a corporate intranet forum. Companies can benefit from creating requirements to post to their intranet forum which contain a title and body of text. These should have input requirements and not accept characters which cannot be cleared from the data. Analysis of our

methods could benefit companies both looking to implement internal corporate intranet forums and to read their existing forums.

References

Some codes and information were adapted from:

1. https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568

   Multi-Class Text Classification Model Comparison and Selection Natural Language Processing, word2vec, Support Vector Machine, bag-of-words, deep learning -Susan Li

2. https://medium.com/datadriveninvestor/predicting-tags-for-the-questions-in-stack-overflow-29438367261e

   Predicting Tags for the Questions in Stack Overflow-Megha Shyam

3. https://stackoverflow.com/questions/43196636/how-to-concatenate-two-layers-in-keras/46442161#46442161