Predict Closed Questions on Stack Overflow

A Machine Learning approach for Natural Language Processing

Pranav Pravin Chaudhari
Environmental Science and Engineering Department
Indian Institute of Technology, Bombay
Mumbai, India
20D180023

Saeel Sandeep Nachane
Metallurgy and Material Science Department
India Institute of Technology, Bombay
Mumbai, India
20D180028

***Abstract*—Through this paper, a *learning-from-data* method is explored which is able to predict whether a question on *Stack Overflow* will be closed. Different machine learning approaches for natural language processing are compared in order to predict the status of a question. We have used Gaussian Naive Bayes Classifier, Multinomial Naive Bayes Classifier, Random Forest Classifier and LSTM. We find particularly high accuracy while using LSTM (Long short-term memory) model.**

***Index Terms*—natural language processing, random forest, support vector machine, naive bayes, LSTM**

I. Introduction

Every day, Stack Overflow is used by millions of programmers to find excellent solutions to their programming problems. Developers who encounter programming issues at work ask nearly 8,000 queries on Stack Overflow on an average day. Currently, 6% of all new inquiries are ”closed” after being answered. Questions can be closed as ”off topic”, ”not constructive”, ”not a real question”, ”or too localized”. Each of the reasons is further explained in the Stack Overflow FAQ. Our objective in this paper is to a classifier which has a highest accuracy that can predict, given a question's original submission, whether or not it will be closed. At the time the question is created, we also know information about the user. In this research paper, we have used *Google Colab* as our primary python IDE. We plan to extract all the useful information in one data-frame which will be the metadata which will be numerical, and the second data-frame would consist of the pre-processed text. After this, we have used different models like *Gaussian Naive Bayes*, *Multinomial Naive Bayes*, *Random Forest*, *LSTM* and *XGBoost* and checked their accuracy for the given problem.

II. Dataset

The dataset to this particular problem was found on the kaggle website. The dataset contained many csv files like prior_benchmark.csv, private_leaderboard.csv, public_leaderboard.csv, train.csv, train-sample.csv and many other. The train-sample.csv contains text related to the post and the associated metadata at the time of post creation which will serve as inputs to our machine learning model. The dataset consists of data through July 31st UTC, and the public leaderboard data goes from August 1st UTC to August 14th UTC. Some of input fields in the train-sample.csv are *PostCreationDate*, *OwnerUserId*, *OwnerCreationDate*, *Title*, *Bodymarkdown*, *Tag1*, *Tag2*, *Tag3*, *Tag4*, *Tag5*, *PostId* and *PostClosedDate*. The target variable, or the output variable is the *OpenStatus*. The public_leaderboard data contains all the above field, except for the target field *OpenStatus* and *PostClosedDate*. We have used the train-sample.csv file which we had uploaded on the drive and imported in the Google Colab notebook. We have made different notebooks for different machine learning models and one notebook for the exploratory data analysis and feature engineering. The reason is that the train-sample.csv file is quite heavy on data. Due to this the RAM available on the Google Colab is not supportive. So we have done complete exploratory data analysis and feature engineering in one notebook and downloaded the final csv file which was *EDA.csv*. This file was then imported in all other notebooks for model building, training and predicting.

III. Exploratory Data Analysis

We have apply several data interpretation functions to get different important features from the dataset. The dataset has a total of 15 columns, including the target variable. The total number of entries in the dataset is 140272. Firstly we checked the data types of each of the columns and found out that except *PostId*, *OwnerUserId*, *ReputationAtPostCreation*, and *OwnerUndeletedAnswerAtPostCreation*, all other columns have object data type. The exceptions are integer data type. The dataset contained two *ID* columns namely, *PostId* and *OwnerUserId*. These columns do not have any kind of predictive power. So we safely dropped these two columns. Table. I shows the statistical inferences of the numerical variables left in the dataset, namely, ”ReputationAtPostCreation” and ”OwnerUndeletedAnswerCountAtPostTime”

We could see that 75% of the undeleted answers are below 8. Moreover, there are no empty cells in these two columns as the count matches with the number of entries (rows) in the dataset. Moreover, Fig. 1 gives us an estimate of the null values present in different columns of the dataset. Table. II shows the percentage of null values in each of the columns.

Moving on to the tags columns. There are 5 columns, namely, *Tag1*, *Tag2*, *Tag3*, *Tag4* and *Tag5*. These columns contain keywords (or catchwords if you may). These columns

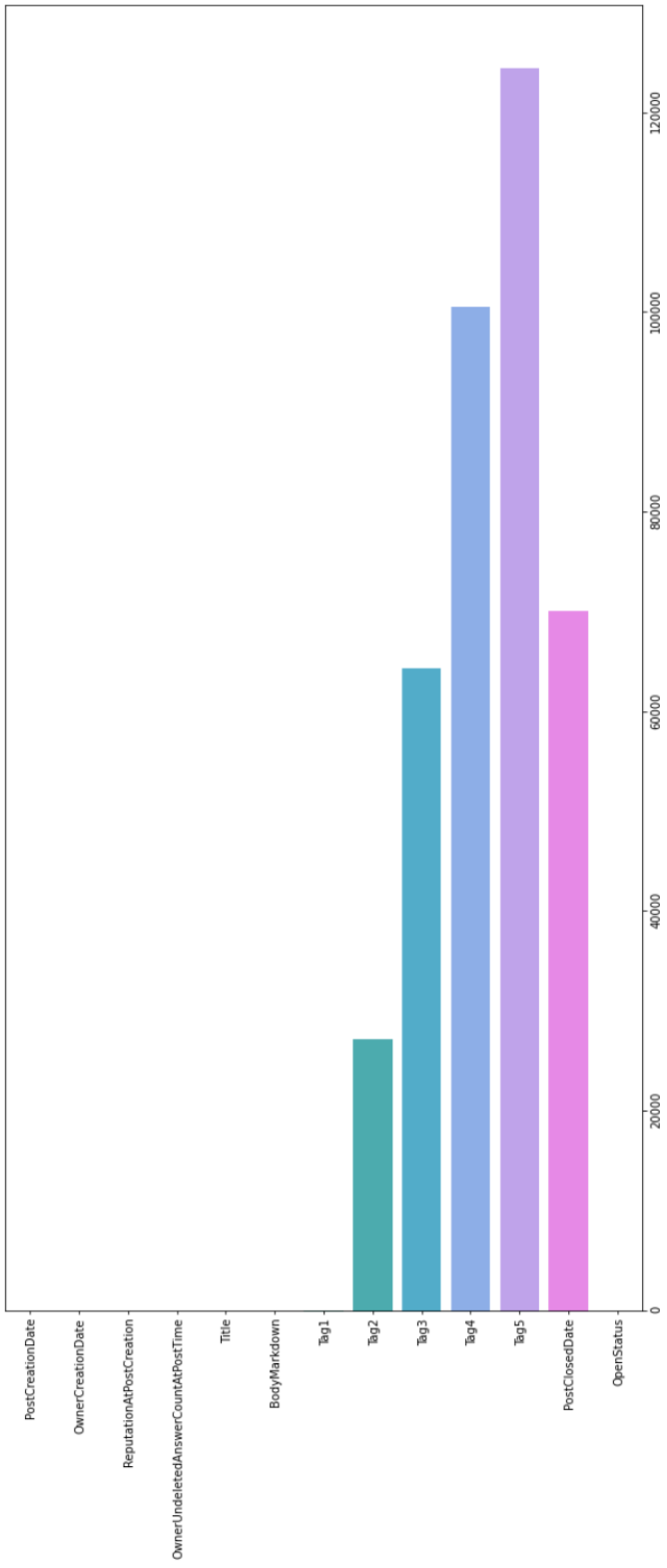

Fig. 1. Null Values

TABLE I
STATISTICAL ANALYSIS OF NUMERICAL DATA

	Statistical inferences	
	ReputationAtPostCreation	Undeleted Answers [a]
Count	140272	140272
Mean	506.63	19.47
STD	2495.90	90.12
Min	-34	0
25%	1	0
50%	35	1
75%	267	8
Max	209631	5772

[a]Undeleted Answers is column *OwnerUndeletedAnswerAtPostCreation*.

TABLE II
PERCENTAGE OF NULL VALUES

Column Name	**Percentage**
PostCreationDate	0
OwnerCreationDate	0
ReputationAtPostCreation	0
OwnerUndeletedAnswerCountAtPostCreation	0
Title	0
BodyMarkdown	0
Tag1	0.0071
Tag2	19.4272
Tag3	45.88
Tag4	71.7335
Tag5	88.7974
PostClosedDate	50
OpenStatus	0

are a valuable data which we will feed in our model. The *Tag1* column is almost complete (around 1000 entries are null out of nearly 1.5 lakh entries). The small number of null values present in *Tag1* were interpolated by using the *mode* of the column. Rest all have quite a number of null values. So we concatenated all the tags columns and stored the result in a new column named *Tags*. We dropped all the tags column. Once we got the *Tags* column, we binary encoded it so that it would be easier for the model to understand.

Next we saw that the *PostCreationDate* and *OwnerCreationDate* could give us the age of the account as they were in *DateTime* format. We made a new column named *AccAge* by simply finding the difference between the timestamps of the two mentioned columns. After this, we drop the *PostCreationDate* and *OwnerCreationDate*.

Next, we deal with the *Title* and *BodyMarkdown* columns. These two columns contain the title and the body of the question. So we have to input these as string itself. But before doing that, we have to clean these two columns. Firstly, we changed them to lower case. Moreover, we created a new column named *Description* which was the concatenation of the two mentioned columns, and then safely drop the previous two columns.

The *PostClosedDate* colum would also not be used in the model. Se we have dropped this column.

Lastly, we dealt with the target variable *OpenStatus*. We know that this variable is a categorical variable with 5 distinct

values. So we mapped the five values to numbers from 0 to 4 as shown in Table. III.

TABLE III
MAPPING OF *OpenStatus*

Value	Key
not a real question	0
not constructive	1
off topic	2
open	3
too localized	4

This ends our exploratory data analysis and feature engineering. Next we start training different models. We will go through *Gaussian Naive Bayes* , *Multinomial Naive Bayes*, *Random Forest*, *LSTM* and *XGBoost*.

IV. NAIVE BAYES

The first model we tried was the Naive Bayes. We used two Naive Bayes models namely Gaussian Naive Bayes and Multinomial Naive Bayes. We would be analyzing all the models using the *Confusion Matrix* and *accuracy score*.

A. Splitting the Dataset

The two Naive Bayes models consist of the same inputs. While doing feature engineering on the dataset, we created the *Description* column which is the result of concatenation of the *Title* and *BodyMarkdown* columns. This column is the most important one in the dataset. So for these two models, our input feature would be the *Description* column. This column contains string. So to use this as the input, we performed vectorization of column. After this, the column would be ready to use in the training. The output variable was the *OpenStatus* column. Using these as our feature and variable, we train and test the two models. 30% of the dataset was used for testing.

B. Accuracy and Confusion matrix

We would be analyzing the *confusion matrix* and the *accuracy score* of the two models. So we initially imported the models from *scikit-learn's naive bayes* library. We trained the models using the *Description* and *OpenStatus* as the inputs and target variable and calculated the accuracy and confusion matrix of the two. The accuracy of the two models are shown in Table. V.

TABLE IV
ACCURACY SCORE

Model	Accuracy
Gaussian Naive Bayes	28.26%
Multinomial Naive Bayes	59%

We would notice that the accuracy of *Multinomial Naive Bayes* was more that twice the accuracy of the *Gaussian Naive Bayes*. This shows that the *Multinomial Naive Bayes* is a better fit for our data than the *Gaussian Naive Bayes*.

V. ENSEMBLE MODELS

After checking the accuracy of the Naive Bayes models, we moved forward to the *Ensemble Models*. We have used two models namely, *Random Forest* and *XGBoost* (or *Extreme Gradient Boosting* models. These are tree models. Here aswell, we would be analyzing the accuracy and the confusion matrix of the two.

A. Splitting the Dataset

The two Ensemble Models have the same type of inputs as that of the Naive Bayes. So the way we splitted the dataset into training and testing, the same files would be used to train these two models.

B. Accuracy and Confusion Matrix

The model is in the *scikit-learn's ensemble* library. After inporting, we trained the model using the dataset and calculated the accuracy. The accuracy of the two models are given in the Table. VI.

TABLE V
ACCURACY SCORE

Model	Accuracy
Random Forest	57.56%
XGBoost	59.11%

We could notice that both the models have nearly equal accuracy, but the XGBoost model is higher in accuracy. In fact this model has the highest accuracy among all the models we have selected (as would be discussed in the *Results* section). This shows that *XGBoost* model is best fit for our data.

VI. LSTM

After using the Naive Bayes and the Ensemble Models, we moved forward to the *Neural Network*. Starting from the end of our exploratory data analysis and feature engineering, we used the processed dataset to train and validate our neural network (unlike using only *Description* and *OpenStatus* columns for training and testing). Before training, we broke the dataset into two parts: first one containing all the text of the dataset (basically *Description* column), and the second one containing the numerical inputs or the meta data (all the columns except *Description* and *OpenStatus*). As we have two inputs, we were in need of a multi-input neural network. This is were LSTM comes in play.

LSTM (*Long Short-Term Memory*) is an artificial neural network (ANN) used in the field of artificial intelligence and deep learning. Unlike standard feed-forward neural network, LSTM is a feedback neural network. We made an eight layer neural network. Our neural network is shown in Fig. 2. We used *adam* as our optimizer of the model, *sparse_categorical_crossentropy* as our loss function of the model and *accuracy* as the metric of the model. We have used two callbacks, *ReduceLROnPlateau* (having patience as 7) which reduces the learning rate when the metric has stopped improving, and *EarlyStopping* (with a patience of 35) which

is a regularization technique for deep learning neural networks that stops training when parameter updates no longer begin to yield improves on a validation set.

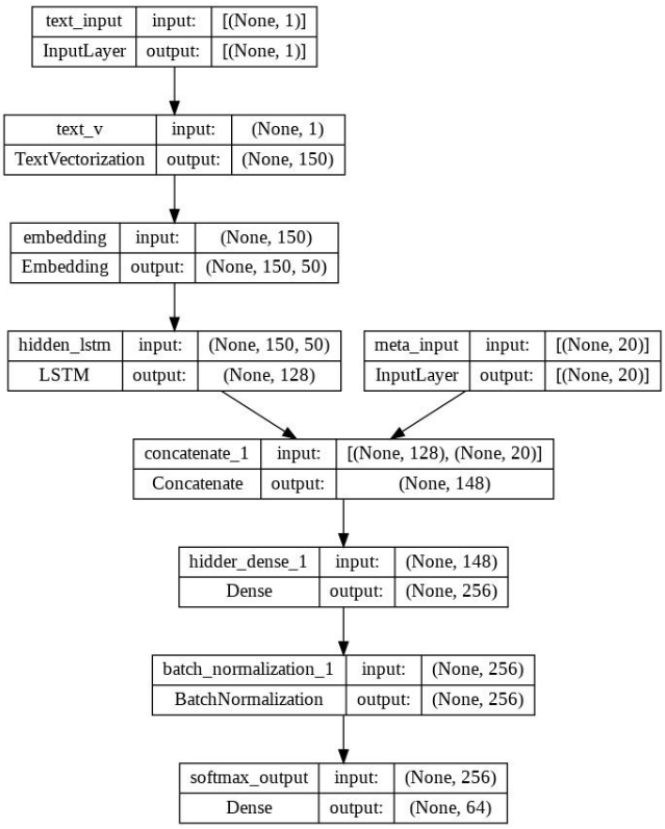

Fig. 2. LSTM Neural Network

As said before, we have two inputs. We have split the total entries into two parts training set and validation set. The validation set is 40%. After this, we ran the neural network with a *batch size* of 256 and 100 *epochs*.

Finally using this as our neural network, we found out that it had an accuracy of 59.10%.

CONCLUSION

We have tried to use five different models, namely, *Gaussian Naive Bayes, Multinomial Naive Bayes, Random Forest, XGBoost* and *LSTM neural network*. The accuracies of all the models are shown in the Table. VI.

TABLE VI
MODEL ACCURACY

Models	Accuracy
Gaussian Naive Bayes	28.26%
Multinomial Naive Bayes	59%
Random Forest	57.56%
XGBoost	59.11%
LSTM Neural Network	59.10%

Overall, we can see that basic models like Gaussian Naive Bayes don't give good performance. But its other version, Multinomial Naive Bayes, is designed explicitly for NLP. Its performance was comparable with complex models like LSTM. This was mainly because of the model's simplicity, preventing the major overfitting problem. Complex models gave high accuracy on the training dataset but comparatively lower accuracy on the test data, highlighting the problem of overfitting. To overcome that, we used models designed to prevent it, like the XGBoost Classifier. Hyperparameter tuning can further increase the accuracy of XGBoost as we would have the ideal combination of the parameters for our problem. So, with acceptable accuracies, we can predict which new questions will be closed on Stack Overflow and their reasons for the same.

FUTURE SCOPE

We tested numerous machine learning (ML) models, ranging from the straightforward Naive Bayes to intricate ensemble models like XGBoost. We also experimented with the recurrent neural network-based LSTM model. We can delve into deep learning for further research on the same subject. We can employ deep learning models such as many-to-one RNN with robust CNN architectures such as Inception or DenseNet spanning the middle layers. The number of parameters will be significantly reduced, and meaningful information will be extracted from the text input with the aid of CNN models. They'll be much more effective for processing data quickly and performing at higher levels. For improved outcomes, research in this area should move in that direction.

ACKNOWLEDGMENT

We would like to show our gratitude to the teaching team of the course *DS203: Programming for Data Science* which was taught by *Prof.* Amit Sethi, *Prof.* Manjesh K Hanawal and *Prof.* Sunita Sarawagi. Moreover, a special thanks and gratitude to all the *teaching assistants* (TA) of this course who have been very helpful towards the students.