AI based moderator for online forums

Shubham Bhasin
IEOR Department, IITB
203190002

Utkarsh Konge
IEOR Department, IITB
203190013

Amit Seth
IEOR Department, IITB
203190023

***Abstract*—Quality of questions determine the quality of discussion on an online forum. Due to huge volume of questions, manual moderation of questions may be difficult. In this project, we aim to build machine learning based moderator to classify the questions into three categories. Using principles from Natural Language Processing, two embeddings were compared using their classification performances on various models. It was found that linear models perform typically better than some ensemble models in this task. This approach has its limitations such as this being offline learning, may not be applicable to large scale forums. Further, the performance can be improved using deep neural networks and other embeddings.**

***Index Terms*—Natural Language Processing, multi-class classification,**

I. Introduction

Online forums serves a purpose of bringing a community together for discussion. To maintain the quality of discussion, the content is moderated. The content can be moderated manually before or after posting, if the number of posts being published are less. Moderators of a forum can also do reactive moderation where users can report the post to bring it to the moderators' notice. Such responsibility can also be distributed to the users such that if a post is reported several times, appropriate action will be taken against it. The last type of moderation is the Automated moderation where Machine Learning (ML) and/or Artificial Intelligence (AI) techniques can be used. In this project, we aim to build a quality predictor for a forum like Stack Overflow.

Stack Overflow is one of the most famous (if not the most famous) forum for programmers to ask their questions. Such questions can help in finding/fixing a bug in a software, generate new ideas to improve existing software, improving and understanding of current algorithms or some elegant way to execute some operation. Such outcomes and quality of discussion are a result of well-written questions and moderating the questions avoids those questions which are just spam. Questions whose answer cannot contribute to the community positively, such as open-ended, off-topic, chatty questions, questions without relevant information etc., need to be flagged. With over 8000 new questions are posted on the forum daily, moderating questions for quality may be a difficult task. [1] Thus, an automated way of classifying questions using principles from Natural Language Processing (NLP) can be helpful at least in identifying specific questions so that moderators can take action.

Using machine learning, predicting the quality of questions have been attempted previously. Deep learning based method using linguistic and semantic features obtained an accuracy of about 74% for a binary classification problem. [1] They solved the problem of classifying the question as whether it is to be closed or not. Some more fragmentation can be added to this such as questions with high-quality, questions which are low-quality and need to be edited, and questions which need to be closed.

In this project, three classes of questions as described earlier are considered. Data analysed here consists of numerous questions from Stack Overflow which have been tagged. First step in this case would be to clean the data and perform basic analysis. Next, for training ML models, numerical data is required. There are various ways to convert text into vectors either by considering the semantics or by using the frequency of terms itself. Two of such methods is compared. Multiple models (classifiers) are trained and tested on the data. These models are compared for their performance.

The main challenge in this project was to handle the large computations. Large number of features arising from the embedding was one of the reasons of this. To mitigate this, the experiments were performed on Google Colaboratory. But due to its run-time limit and various other conditions it was difficult to train complicated models for long time.

The report is arranged as follows. In the next section, the approach and methodology practiced in this project is discussed. Results are discussed in the section which follows it. Some future approaches and opportunities are also discussed later. The article is then concluded and statement of contribution is presented as last.

II. Background and prior work

Using machine learning to predict quality of questions have been attempted previously. Deep learning based method using linguistic and semantic features obtained an accuracy of about 74% for a binary classification problem. [1] They solved the problem of classifying the question as whether it is to be closed or not. Some more fragmentation can be added to this such as questions with high-quality, questions which are low-quality and need to be edited, and questions which need to be closed.

Previously, this problem has been approached using Term Frequency Inverse Document Frequency embeddings using only 1-gram terms. Some features which may act as proxy for how the question can is presented such as number of code blocks used, number of lines of code given, were not considered. In this project, we have compared two embeddings based on their performance on various models.

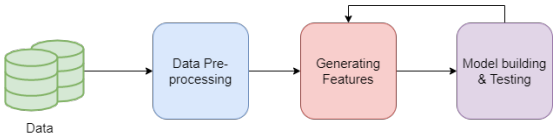

Fig. 1: Framework for predicting quality of a question

III. Methodology

A general approach used in ML project is given in Figure (1). First step is data gathering. The data is then cleaned and processed in the form which can be analysed. Useful insights can be drawn from the data in the next step and features can be extracted. ML models can then be build and features can be adjusted based on the performance. Each step in the framework is explained as follows with context to the project.

A. Data

The data consisted of 60000 labelled questions out of which 45000 were used for training and the rest 15000 were used for testing. The data was obtained from a Kaggle competition. [2] The data contains the title of the question, body of the question, tags, creation data and the the quality being one of "HQ" (high quality), "LQ_EDIT" (low quality - to be edited), "LQ_CLOSE" (low quality - to be closed).

B. Data Pre-processing

The text in the questions was available in HTML format with its tags. To clean it, the tags were removed using `beautifulsoup` package. Exploratory data analysis was also performed in this step.

C. Feature Engineering

The questions are available in text format with HTML tags. To convert the text into numerical data, various embeddings can be used. In this work, we have Term Frequency Inverse Document Frequency (TFIDF) [3] embedding and Sentence Transformers [4]. The former considers the relative importance of the terms based on its frequency and does not consider semantic meaning of the sentence. The latter considers sentence as a whole and embeds it into numerical vector. These individual vectors are stacked together in order of words or sentences. Sometimes, there may be large number of vectors/features which increases the size of the data set. In such case, the number of features can be bounded at some maximum level. Extracting features from the data using these embeddings have been different and is discussed as follows.

1) Sentence embedding: In sentence embeddings, a sentence in a paragraph is converted into a vector of specified dimensions. For this, we have used `sentence_transformer` package [4]. It provides various models for embedding a sentence into a vector. Based on computational time and data size, we have chosen average glove embeddings. It provides 300 dimensional vector for every sentence based on glove embeddings of the words in it. This is done for each sentence in body of the question and the title. As will be seen later, the maximum number of sentences in the question body is 201. Such a huge number of lines is not necessary to describe the whole question. Thus, we have taken 2 cases where we take the first 12 sentences and first 10 sentences into consideration for embedding. Note that, this does not include the lines tagged as code in the body. This is similar to truncating the question body to be maximum 12 or 10 sentences.

2) TFIDF: The term frequencies can be computed for a single word or a group of words. the term "n-gram" refers to clubbing n consecutive terms into a single term. TFIDF can be applied to such clubbed terms. In this project, 1-gram and 1 and 2 gram terms are used and their performance is compared. In the latter case, terms formed with 1 term and 2 terms are considered. to form such terms various short forms such as I've,I'm are expanded. Then the stop-words like is,the,a,an etc which do not convey important information are removed and the text is converted into lower case removing all the punctuation marks. The frequencies of each term are then computed.

As the embeddings capture information in different forms, performances of embeddings are compared. Apart from these, various other features which can act as a proxy for how the question is presented can also be extracted using the HTML tags. Such features include length of the question in sentences, number of tags used, number of lines of code provided etc. These other features and how they are extracted is discussed while presenting the results.

D. Model Building and Testing

The problem here is of classification. Thus, models like Logistic Regression, Linear SVC, Random forest (RF) and Stochastic gradient descent classifier (SGD-C) are considered. Random forest is an ensemble model where multiple predictors (decision trees) are trained and finally, some sort of an aggregate of all the predictions is considered. Others are linear classifiers. We have also tried XGBoost Classifier.The results for each of the model are discussed in the following section.

IV. Results and Discussion

All the programming was done on Google Colaboratory. Results from various stages are discussed in this section.

A. Feature Engineering

After removing the HTML tags in data pre-processing step, the data was analysed. Various features such as the number of sentences in body, codes and the number of tags were created. Figure (2) shows the distribution of the number of tags used to describe the question. Most of the questions were with just 2 tags. To see which tags were used the most, a barplot was created based on the frequencies of each of the tags. This is shown in Figure (3). It can be observed that most used tags are `javascript`, `python`, `java`. Other tags such as `php`, `android`, `c#` etc can also be seen but these have lower frequency than that the former tags.

When a question is posted on Stack Overflow, it provides the user with various formatting options like making text bold, creating heading etc. The number of such tags used in a question is also analysed. It was seen that close to 40000 out of 45000 questions did not use the extra tags.

Based on the HTML tags the code body and the text body of the question can be separated. The number of lines in the body(`Body_size`) and code blocks can be extracted from this along with the number of code blocks (`Code_blocks`) used in the question. Other features which can be extracted are the number of liens of code (`Code_Lines`) and the tags used for highlighting such (`Extra_tags`). These acts as a proxy for how the question is presented. Thus these features are also extracted from the data. The variation in them is shown in Figure (4) in form of a pairplot. The distribution of these features is highly skewed towards right.

The dependence of these features on the class is quantified through Point Biserial Correlation. [5] The correlation was negligible which states that the presentation of the question alone doesn't matter that much to its classification.

Sentence Embeddings and TFIDF were used to convert text into numerical vectors. First, the sentence embeddings are discussed. It was observed that there are very few questions with high number of sentences. For such cases, instead of formatting some code as code in the question, some users had provided the whole code in text itself. Embedding all the sentences would create a huge number of features. Thus, the number sentences is kept low. Here, two scenarios are considered one where the number of sentences are capped at first 12 ($S1$) and other at first 10 ($S2$). These limits of 10 and 12 were chosen as out of 45000 questions, only 673 had size more than 12 and 1255 had size more than 10. For each sentence, a 300 dimensional vector was obtained using the package `sentence_transformer` and the model `average_word_embeddings_glove.6B.300d`. So in scenario $S1$, this created a vector of 3600 dimensions (12 times 300) for each question and in $S2$ the size was 3000.

For TFIDF embeddings, the whole question including the code blocks were considered as document. In the pre-processing step, various stop-words were removed and text was converted into lower case. One-gram and bi-gram terms are considered for encoding the text. In addition to this, one-gram embedding with maximum features capped as 10000 is also considered.

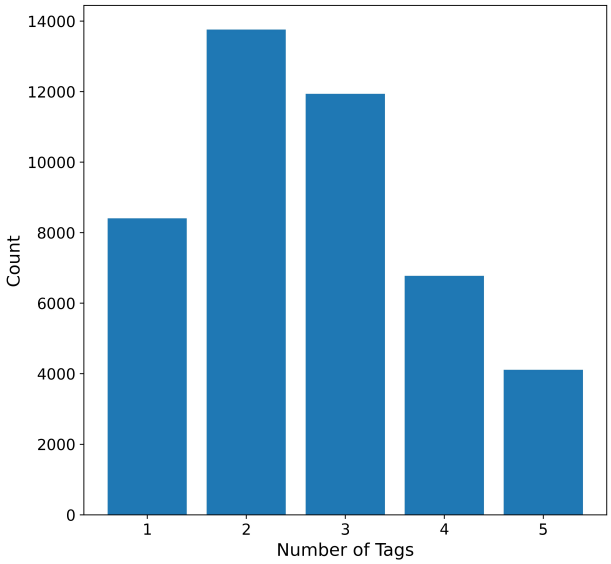

Fig. 2: Distribution of number of Tags used for question

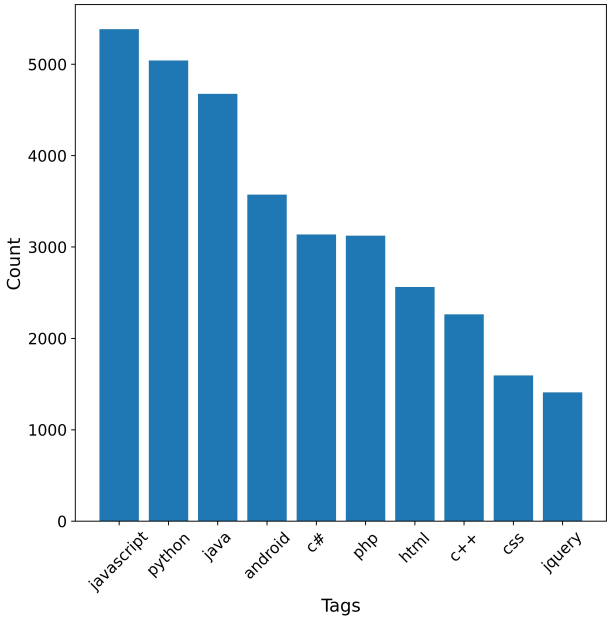

Fig. 3: Bar plot of top 10 tags

B. Model Building and Testing

It was observed that the classes are perfectly balanced with 15000 instances of each label in training set and 5000 instances of each label in testing set. Thus, accuracy is used as a metric for the models. All of the cases discussed above are subjected to hyper-parameter tuning with 5-fold cross validation. Various models are tested and their performance is discussed in this section. The results are segregated embedding wise.

1) Sentence embeddings: First the results of sentence embeddings with 12 sentences ($S1$) is discussed. Due to large number of data, the processing of data was slow and the colab session used to crash multiple times.

To reduce dimensionality, principal components of the data were calculated and ML models were trained on it. First the performance of the models is checked with 800 principal components which explained about 80% of the variance. The model performances is shown in Table (1). It can be observed that RF model over-fits while the decision tree and Neural network under-fits. XGBoost classifier also shows some over-fitting.

For RF model, maximum depth was 20 and number of estimators was 200. In the Neural Network Model,Linear stack

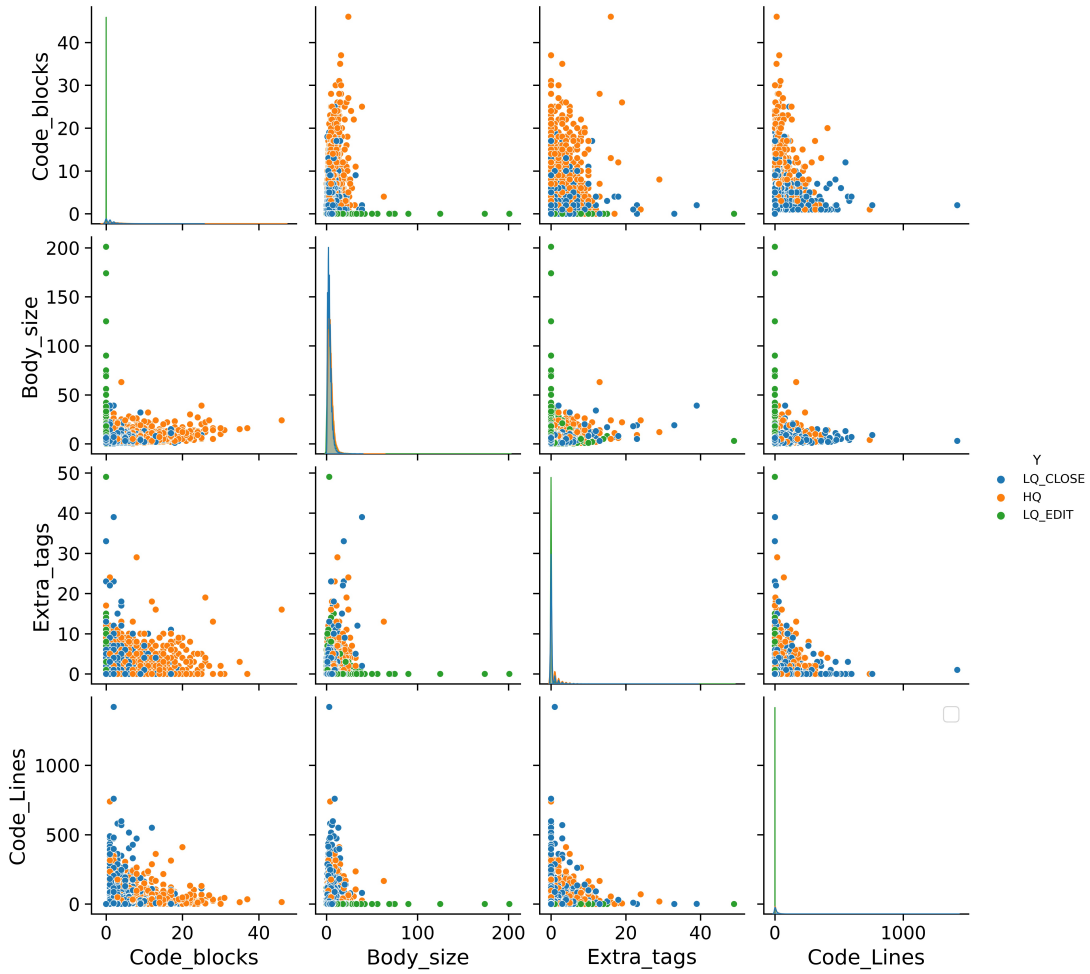

Fig. 4: Variation in extra features

Model	Training accuracy	Testing accuracy
RF	100%	62%
Decision tree	50%	40%
Neural network	54%	49%
XGBoost	83%	68%

Tab. 1. Results for scenario $S1$ (800 components)

Model	Training accuracy	Testing accuracy
RF	62%	59%
Decision tree	88%	69%
Neural network	70%	65%
XGBoost	85%	71%

Tab. 2. Results for scenario $S1$ (2000 components)

of layer is used. Two hidden layers are used with activation functions as ReLU. First hidden layer has 12 neurons and second hidden layer has 8 neurons. In compilation of the model, adam optimizer is used which combines the best properties AdaGrad and RMSProp. The batch size is kept fixed. For XGBoost, number of estimators was 700, learning rate 0.05 and maximum depth was 4. For decision tree classifier, the maximum depth was 5 and split criterion was Gini index. These hyper-parameters were used for model with 12 sentences.

Table (2) shows model metrics when 2000 principal components were used as features which explain about 99% of the total variance. The performance of the models which were under-fitting has improved due to increase in number of features, but the testing accuracy is still low. XGBoost however gives better results but at cost of increased training time.

Next, results for scenario $S2$ (considering only first 10 sentences) are discussed. For this case, only linear models were tested without any dimensionality reduction to find a balance between computational complexity and model complexity. Table (3) shows the results of models without any extra features which were described earlier. For this data-set, through hyper-parameter tuning using grid search and cross validation was performed. Random forest took too long to converge for this data-set. The linear models performed fairly same on training and testing data.

For Logistic Regression the hyper-parameters were `C = 0.01, max_iter=150`. The parameter

Model	Training accuracy	Testing accuracy
Logistic Regression	67%	65%
Linear SVC	69%	65%
SGD-C	68%	65%

Tab. 3. Results for $S2$ (10 sentences) w/o extra features

Model	Training accuracy	Testing accuracy
Logistic Regression	80%	73%
Linear SVC	79%	73%
SGD-C	76%	73%

Tab. 4. Results for $S2$ (10 sentences) with extra features

`C` was same for Linear SVC and for SGD-C, `alpha=0.001,l1ratio=0,loss=hinge`

It was discussed earlier that extra tags like the number of sentences in the body, number of code blocks etc. were did not contribute to prediction of the classes. But after including them in features, a small improvement in the prediction accuracy was observed as reported in Table (4). The training and testing accuracy both has increased. This is a typical case where some features alone may not be able to directly predict the class but can show synergistic effect with other features.

For this scenario, the hyper-parameters for Logistic Regression was `C=10.0,max_iter=600`. For Linear SVC, `C=1.0` and for SGD-C `alpha= 0.0001, l1_ratio= 1.0, loss = hinge`.

2) TFIDF Embeddings: Now, the results from TFIDF embeddings are studied. After pre-processing the data appropriately as discussed earlier, 1-gram embeddings were generated and various models were trained on it. The results obtained after through hyper-parameter tuning is given in Table (5). The accuracy of linear models has increased to more than 80% but the models are over-fitting maybe except the case for SGD-C. RF model does not perform as good as the linear models. Gradient boosting was tried on this data but it took too long to converge for the grid search. Hence its results are not included, but are available in the source code submitted along with this.

For this case, the best hyper-paramters for Logistics Regression were `C = 1.0, max_iter = 150`, for Linear SVC it was `C =1.0`, for RF `max_depth = 20, n_estimators = 600` and for SGD-C `alpha = 0.0001, l1_ratio = 1.0`

In another scenario, the number of features of TFIDF were capped at 10000 and its results are presented in Table (6). It shows slight improvement in testing accuracy while checking the over-fitting. Here too, the performance of all the linear classifiers was similar.

Model	Training accuracy	Testing accuracy
Logistic Regression	90%	83%
Linear SVC	96%	82%
RF	86%	75%
SGD-C	82%	81%

Tab. 5. Results for TFIDF embeddings (1 gram)

Model	Training accuracy	Testing accuracy
Logistic Regression	87%	83%
Linear SVC	86%	82%
RF	86%	77%
SGD-C	82%	81%

Tab. 6. Results for TFIDF embeddings (10000 features)

Model	Training accuracy	Testing accuracy
Logistic Regression	100%	84%
Linear SVC	100%	83%
RF	81%	75%
SGD-C	88%	81%

Tab. 7. Results for TFIDF embeddings (1 and 2 gram)

For this case, the best hyper-paramters for Logistics Regression were `C = 1.0, max_iter = 50`, for Linear SVC it was `C =0.1`, for RF `max_depth = 20, n_estimators = 600` and for SGD-C `alpha = 0.0001, l1_ratio = 1.0`

Next, 1-gram and 2-gram terms are also checked for accuracy. These embeddings were generated by a similar process as 1-gram TFIDF but by changing a parameter in its function. The same models as before were trained and ehtir performance is tabulated in Table (7). The linear models except SGD-C show some degree of over-fitting by having 100% training accuracy. RF model also shows over-fitting but with worse performance.

For this case, the best hyper-paramters for Logistics Regression were `C = 100.0, max_iter = 150`, for Linear SVC it was `C =1.0`, for RF `max_depth = 20, n_estimators = 600` and for SGD-C `alpha = 0.0001, l1_ratio = 0.0`

V. Future Opportunities and Limitations

In this work 2 embeddings were compared with majorly simple models. Although, linear models performed well their accuracy can still be increased by using complex models like deep neural networks. Various other embeddings other than Glove can be used. Apart from this, it must be noted that this exercise is offline learning. To apply it to the scale of Stack Overflow may be difficult due to huge volume and variety of questions. However, such approach can be applied to small scale educational forums like Spoken Tutorial (https://spoken-tutorial.org/).

VI. Conclusion

In this project, a multi-class classification problem was solved to classify the questions on Stack Overflow forum into three categories. Two types of embeddings were compared with respect to the performances on various models. TFIDF embeddings seem to give better results with linear models. This also reduces the training time for the computation. Future work and limitations of the approach are also discussed.

VII. Statement of Contributions

Utkarsh Konge contributed to gathering, pre-processing data and generating features. Shubham Bhasin contribured to pre-processing data and testing models. Amit Seth contributed to literature survey and report writing. Shubham and Amit contributed to video making. Utkarsh Konge controbuted to arranging all the codes, report and data.

References

[1] L. Tóth, B. Nagy, D. Janthó, L. Vidács, and T. Gyimóthy, "Towards an accurate prediction of the question quality on stack overflow using a deep-learning-based nlp approach." in *ICSOFT*, 2019, pp. 631–639.

[2] "60k stack overflow questions with quality rating," Oct 2020. [Online]. Available: https://www.kaggle.com/imoore/60k-stack-overflow-questions-with-quality-rate

[3] C. Sammut and G. I. Webb, Eds., *TF–IDF*. Boston, MA: Springer US, 2010, pp. 986–987. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_832

[4] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[5] D. Kornbrot, *Point Biserial Correlation*. American Cancer Society, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat06227