StackOverflow Tag Prediction

Aishwarya Singh
aisingh@ucsd.edu
University of California - San Diego
San Diego, California, USA

Ravin Kale
rkale@ucsd.edu
University of California - San Diego
San Diego, California, USA

Sourav Roy
s3roy@ucsd.edu
University of California - San Diego
San Diego, California, USA

Abstract

In this assignment, we want to predict the tags that are assigned to each post on StackOverflow and also investigate what influences a post to have a high score. Our work is inspired by the Kaggle contest "StackOverflow Tag Prediction". The goal of this prediction is to correctly identify tags to every posted question to make life easier for the person posting the question and the ones who are about to answer it.

Introduction

Stack Overflow is a platform where professional and enthusiast programmers can ask and answers questions. They have over 14 million registered users as of March 2021, and they had received over 21 million questions and 31 million responses. Each question is assigned tags, which identify the category to which it belongs. It is in the best interests of both individuals who post questions and those who are interested in the answers if a post is properly tagged. Tags can be assigned manually or by picking from a list of the most frequently used tags.

Modelling tag prediction has various advantages. People could be introduced to tags that are of interest to them and cleanup systems, in which mistags are recognized and repaired, could also be deployed. We have developed a multi-label classification model to predict tags for each question based on it's content.

1. Data

1.1 Identify a dataset

The StackOverflow data sets were obtained from Kaggle [1]. It has three different datasets, namely: Questions, Answers and Tags. Question data set has 1,264,216 questions posted by 645,363 unique users. Answers data set has 2,014,516 answers to 1,102,568 posted questions and tags data set have 37,035 unique tags which are associated with the questions dataset.

1.2 Exploratory Data Analysis

In the data analysis part we first had a gist of three different data sets and cleaned it by removing the HTML tags, Punctuation and stop words doing this we get the cleaned data to run a predictive task or do the further analysis. We then found out the number of tags for each question and further analysed with removing punctuation,without capitalization and with lemmatizer and as well as with removing punctuation,without capitalization and without lemmatizer.

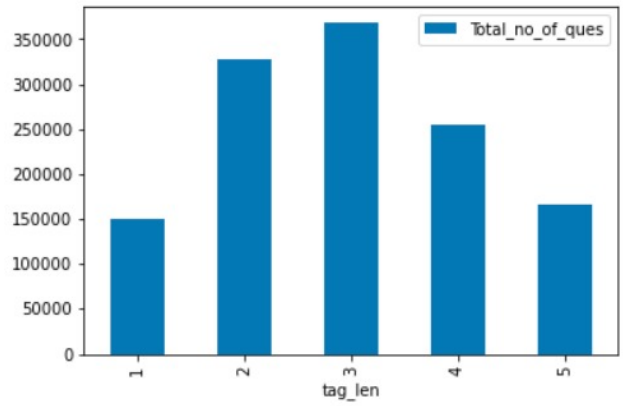

Figure 1. Frequency of number of tags in a question

It was very interesting to find that 650 most popular tags(based on occurance in the dataset covered just about 95% of questions in the dataset. As a matter of fact, 50th most popular tag i.e. Bash, appeared in 7484 questions.

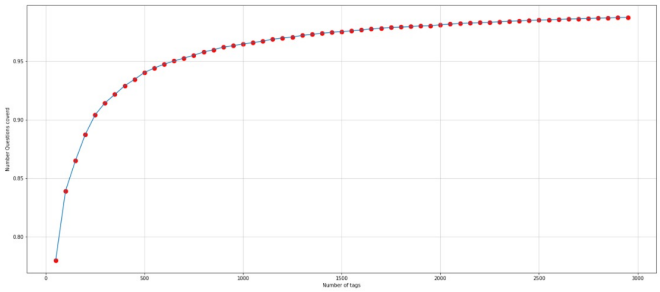

Figure 2. Question coverage base on N-popular tags

Going further we visualised relationship between tags and number of questions they are tagged to. Javascript, Java, C#, php, andorid were 5 most popular tags.

2. Related literature

StackOverflow data has been widely used for text analytics works before. There are some researches that are similar to our work.

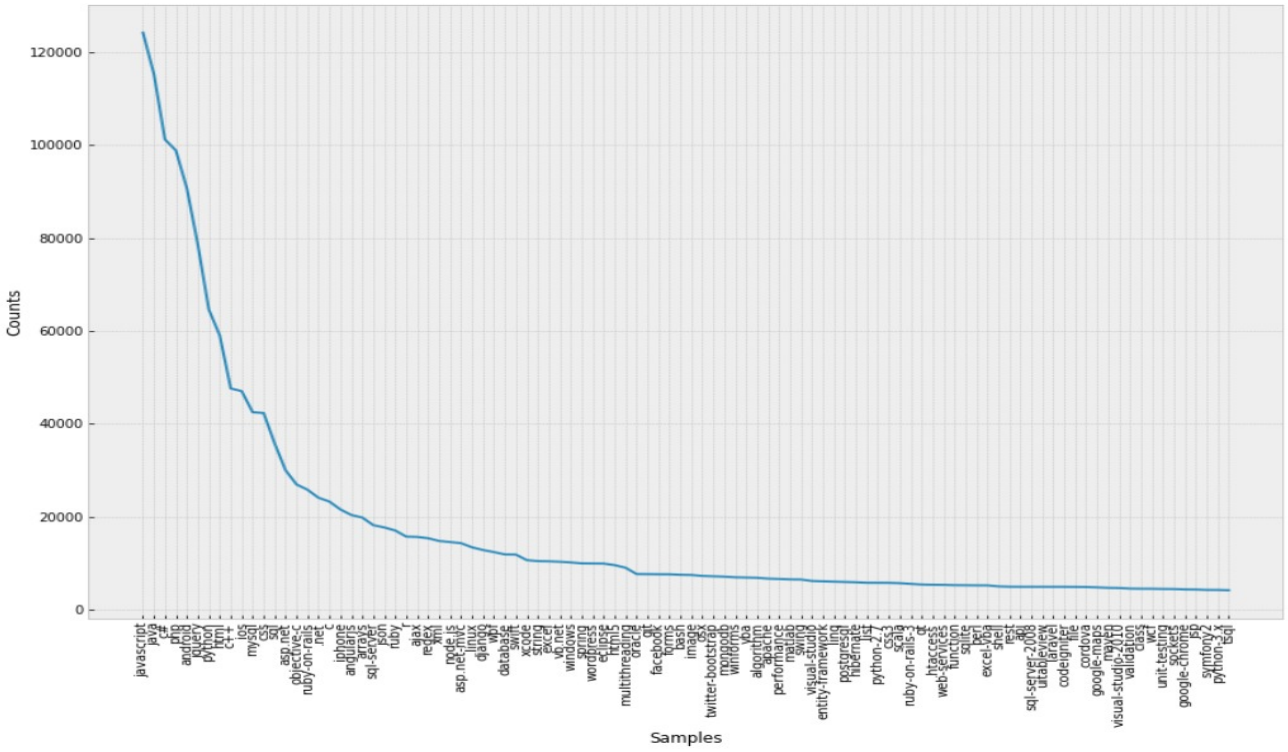

Figure 3. Frequency of each tag in the dataset

In 2011,Kuo [2] had worked to compare word prediction methods where they compared version of co-occurrence, K-Nearest-Neighbor method and Latent semantic indexing against a baseline. Using the same model they predicted tag on StackOverflow dataset by constraining the next word predicted to only tags. Another research in 2013, Predicting Tags for StackOverflow Questions [3] was done by Sebastian Schuster, Wanying Zhu and Yiying Cheng. They used the StackOverflow data set and extracted features such as whether tag is part of the question(title/body) and pointwise mutual information for their content based classifier and code from the body of the question. Finally they combined both to create their hybrid model. State-of-the-art methods applied for such tasks are mostly vectorized word features from the body of the text.

3. Identify a predictive task

3.1 Feature Selection

Initially, we checked for duplicate entries in the data but there were none, then we cleaned the data set of 1.2M questions by selecting questions that had a score of 1 or more and more than 10 words in the body. After filtering we ended up with around 597K questions and finally we randomly selected 150K questions. We then transformed the title and body of each question to vector of words using Term Frequency-Inverse Data Frequency(tf-idf). We used combination of unigrams and bigrams to build our feature set. The tf technique counts the occurrence of each word in a question and idf technique helps penalize the common words based on their frequency of occurrence throughout the data. The tf-idf formula,

$$w_{ij} = tf_{ij} * log\frac{N}{df_i}$$

where,

N = Total number of documents

$$tf_{ij} = number\ of\ occurrences\ of\ word\ i\ in\ jth\ document$$

$$df_i = number\ of\ documents\ in\ which\ word\ i\ appears$$

Since the data we selected contains code snippets in the body of the question, we decided to extract the codes separately and process it to determine the language in which the code was constructed. Almost all code snippets were traceable except for 7 of them. Each question was then tokenized for the coding language they represented. The tf-idf vector and code language tokens were the final features used to build the model.

3.2 Performance Evaluation

Looking into the task in hand we have to be careful and we want to make predictions with high precision and recall. We have used mirco averaged Precision, Recall and F1 score and also Hamming loss.

$$Micro\text{-}Averaged\ Precision(P_{M_A}) = \frac{\sum_{t_i} TP_s(t_i)}{\sum_{t_i} TP_s(t_i) + \sum_{t_i} FP_s(t_i)}$$

$$Micro\text{-}Averaged\ Recall(R_{M_A}) = \frac{\sum_{t_i} TP_s(t_i)}{\sum_{t_i} TP_s(t_i) + \sum_{t_i} FN_s(t_i)}$$

$$Micro\text{-}Averaged\ F1\ score(F1_{M_A}) = \frac{P_{M_A} * R_{M_A}}{P_{M_A} + R_{M_A}}$$

$$Hamming\ Loss\ = \frac{1}{m} \sum_{i=1}^{m} \frac{|Y_i \cap Z_i|}{|L|}$$

where,
TP = True Positives
FP = False Positives
FN = False Negatives
ti = Tag class
m = No. of Samples
Yi = True label
Zi = Predicted label
L = Number of Labels

Precision is the measure of quality, recall is the measure of quantity, f1 score is the harmonic mean of precision and recall, and Hamming loss tells us the fraction of wrong labels to the total number of labels.

4 Model

Given a question we have to predict the tags that can be assigned to it. Although it looks like a simple classification problem but it isn't one. Binary and multi-class classification has output values as (0,1) and (0,1,2,...,N) respectively. Unlike them, here each question can have set of tags assigned to it, i.e. nth question can have tags k number of tags assigned to it.

$$Q_n = t_1, t_2, ..., t_k$$

To make such prediction we are building a multi-label classification model.

For baseline, we have used Naive Bayes Classifier with One-vs-the-rest(OVR) strategy. OVR uses the binary relevance method to perform multi-label classification, which involves training one binary classifier independently for each label. The baseline model has a f1 score of 0.42, recall of 0.39 and a precision of 0.45.

To better our prediction we have used Linear Classifiers with Stochastic Gradient Descent learning[passing alpha(constant that multiplies regularization), loss(log loss function to get probabilities) and penalty as parameters] and Logistic Regression separately[passing solver(liblinear), C(Inverse of regularization strength) and penalty as parameters] and compared the model performance.

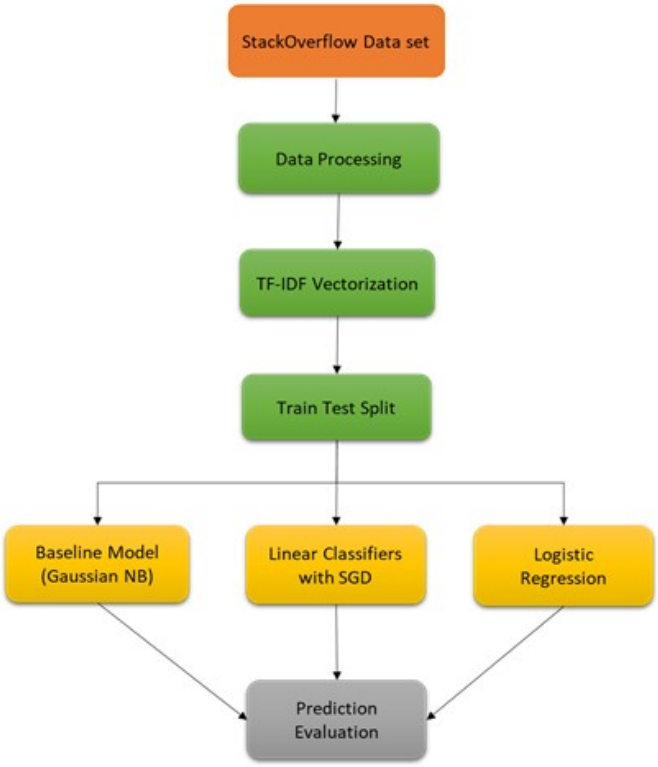

Figure 4. Process Flow Diagram

Stochastic Gradient Descent is computationally fast, it allows minibatch learning and for large datasets it can converge faster as it updates its parameters frequently. Since the updates are frequent, they are computationally expensive. Logistic Regression is easy to implement, can be easily extended to multiple label classification(that's what we need), training a logistic regression on such a high dimensional data is cheap and it is less inclined to over-fitting. But a big setback of logistic regression is that it assumes linear relationship between independent and dependent variable, also it requires independent variables to have average or no multicollinearity. Due to complexity of modelling with OVR, using Random Forest and Support Vector Machine was not a feasible option. We have used L1 penalty(as the data is huge and sparse) for all models and validation performance on test to prevent over-fitting.

5 Model Performance

5.1 Train Test Split

We randomly divided the 150K cleaned questions into train and test sets at a 75:25 ratio. For tf-idf model, the dependent variable consisted of vector representation of whether the

tags of the question are part of top 650 tags from the training set.

5.2 Model Prediction

For a given question, the model return a set of 650 binary values, which indicates the tag it can be assigned to. If no tags are returned we have assigned the top 2 tags based on their probabilities. While Linear Classifiers with Stochastic Gradient Descent learning predicted maximum of 7 tags, the Logistic Regression returned maximum of 8 tags, same as our baseline model. The performance of all three models based on the number of tags returned clearly shows that Linear Classifier with SGD performs better. Although we have a total recall for Gaussian Naive Bayes at 7 and 8 tags but precision has dropped by quite a lot.

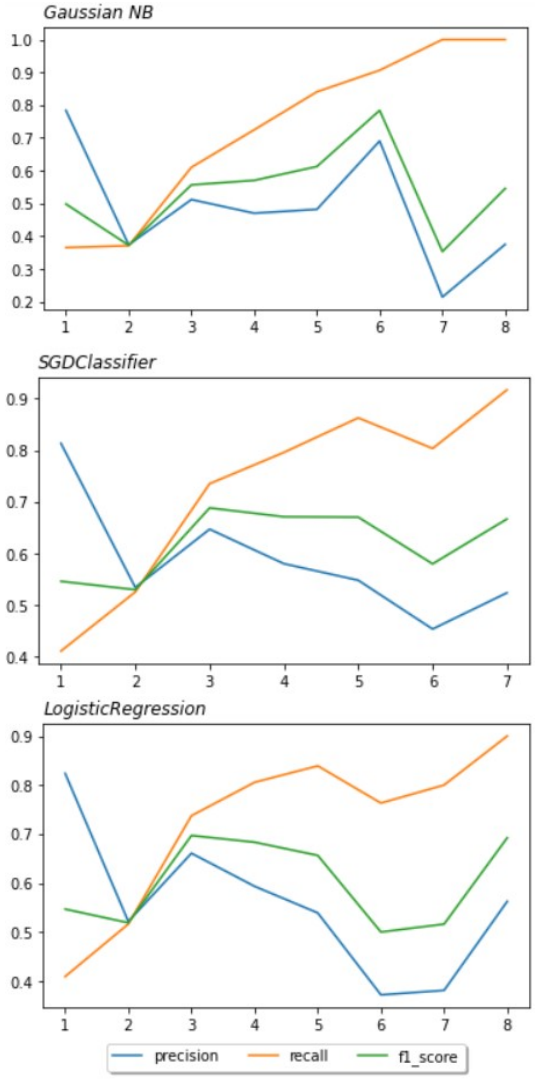

Figure 5. Number of Predicted tags v/s Performance

5.3 Hyper Parameter Tuning

After drawing down on models to use, we used hyper parameter tuning to find the best fit model for the data we have processed. Initially, we tweaked with the feature preparation and prepared combination of uni-gram, bi-gram and tri-gram tf-idf vectorized features, fit them into our model and checked the performance on test set. Similarly, we removed the tri-gram feature from our model and checked the performance and so on.

The combination of all three has 80,387 features, followed by 73,854 features for uni-gram and bi-gram, and just 'x' features for the model with only uni-grams. Then I tuned Linear Classifiers with Stochastic Gradient Descent learning for all three combinations by regulating it's alpha value and check it's performance on both train and test set . We repeated similar process for Logistic Regression, where we iterated over inverse of regularization strength, and Gaussian Naive Bayes Model, where we iterated over the alpha value of multinomial naive bayes.

5.4 Final Model

The best model was the Linear Classifiers with Stochastic Gradient Descent learning with uni-gram and bi-gram features performed the best(Precision:0.6164, Recall:0.5154, F1 Score:0.5614). Model with tri-gram features(Precision:0.6033, Recall:0.5021, F1 Score:0.5529) was close to the one without, and also logistic regression performed(Precision:0.6056, Recall:0.5014, F1 Score:0.5544) close to SGD but our final model won by a slight margin.

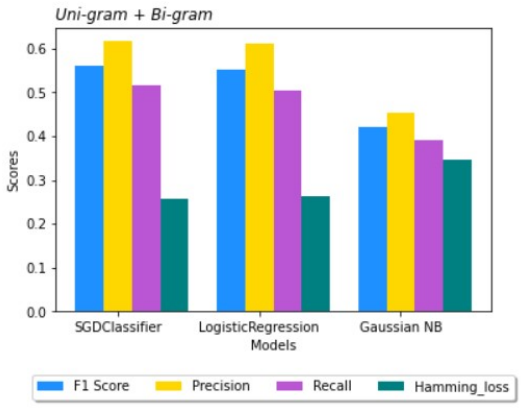

Figure 6. Performance of different models on Test Data for uni-gram and bi-gram features[Note:Hamming loss was multiplied with 100]

Conclusion

In this paper, we described about how we solved a multi-label classification task to predicts tags to StackOverflow questions. We made use of Linear Classifiers with Stochastic

Gradient Descent learning, Logistic Regression and Gaussian Naive Bayes to solve the problem in hand.

We used mainly references from [3] to compare our solutions. Their hybrid model achieved a better recall by a slight margin but in return they had to compromise with the precision. Our models are better balanced and have mostly outperformed their models. The input variable to the model made a big difference. While they searched for presence of tags tags in body and title of questions, we took into consideration all words in the question weighted on how less frequently they occur in the training data set. Our models predicted close to the actual number of tags when compared to predictions.

Our current model can be improved by digging deeper into the feature vectorization. Since the data is huge it takes some time to process the data. The runtime performance can also be improved by optimizing features to be used. Also we could make use of word2vec feature and look at the co-occurance of tags with respect to tf-idf feature built from the questions.

References

[1] StackOverFlow Dataset. [n. d.]. Master's thesis. https://www.kaggle.com/stackoverflow/stacksample

[2] Darren Kuo. 2011. *On Word Prediction Methods*. Master's thesis. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-147.html

[3] Yiying Cheng Sebastian Schuster, Wanying Zhu. 2013. *Predicting Tags for StackOverflow Questions*. Master's thesis. Stanford University, California, USA.