StackOverflow Tag Predictor - Machine Learning Project Progress Report

Navya Aggarwal
2018349

Nitin Gupta
2018251

Sandeep Kumar Singh
2018363

I. Abstract

As the times moves, the amount of data present on the internet is increasing rapidly, and so is the data stored by the online education forums. Some of the common education forums are GeeksforGeeks, Stackoverflow, Chegg. Large-scale datasets for these websites are available online. We are trying to develop a system for Stack Overflow in which we are processing a publicly available dataset, to predict the correct tag by merely looking at the provided title and the body of the question. For this, we are using 6000 most frequently used tags in the available dataset.

Keywords - Stack Overflow, tags, prediction, SVM, Naive Bayes

II. Introduction

Stack Overflow is the largest, most trusted online community for developers, students, and other educationists throughout the world. It is an educational resource available to everybody to clarify doubts, ask questions, answer questions, vote questions, and answers up or down and similarly, post questions and answers like Wiki or Quora.

.

Stack overflow's popularity is due to its high response time from fellow users; this happens because when a question is posted then for every tag other users who have shown interest in those topics get notified. Also, users earn points which further help them gain some benefits on the website like the ability to comment or upvote or answer other questions. All of these play a crucial role in creating such a responsive and informative website.

For each question on StackOverflow, tags are used to relate different questions to categories. Tags are assigned to questions by users only. Further, users can subscribe to tags on a particular topic to receive digests of new questions for which they might be curious to learn and share knowledge. StackOverflow allows users to assign between one to five tags to a posting manually.

Even when an old or experienced user posts a question on Stack Overflow, he faces difficulty in tag identification which describes the question correctly. Most users use the terms present in the questions as the tags, but in most of the cases, it does not describe the question adequately. Due to this, some questions are given incorrect tags which do not represent their question.

The tag predictor can be used for the following scenarios :

1) All the questions belonging from the same topics can be grouped for the user to navigate easily.
2) We can show the post to the user related to the questions he or she is asking on the Stack Overflow

.

III. Literature Review

Many articles and similar projects are available on the internet, which gives us an insight into the tag assignment to a question or an answer. Students, educators, and computer science enthusiasts from all across the globe have contributed.

A paper titled 'Predicting Tags for Stack Overflow Questions Using Different Classifiers' was presented by Taniya Saini and Sachin Tripathi of ISM Dhanbad. According to the paper, relevant data extraction from the data was a crucial task, and text classification is a multi-class and multi-label classification problem.

The approach that was followed in this paper was to use a one-vs-rest classification method. In the one-vs-Rest strategy, we fit one classifier per class. Each class is fitted against the rest of the classes. The different classifiers used in this paper were Logistic Regression, Naive Bayes, SVM.

Another paper titled 'Predicting Tags for StackOverflow Questions' by PhD students at Stanford demonstrates an approach which helps in building a predictor. According to the paper, tags for a question can be divided into two parts.

- Firstly, all questions can be broadly classified using a tag indicating the technology or the programming language they are related. This type of methodology to classify files according to language is also employed by GitHub.
- Lastly, the questions can be related to a subtopic or paradigm, and a separate classifier can be used to predict them.

This dividing approach can be used to predict tags more concisely. Thus, building classifiers for different types of classification can help predict labels with greater accuracy.

IV. Dataset

The dataset used is available at Kaggle as part of a past competition. The dataset has 60,34,195 data points, which are Stack Overflow questions with their associate tags. For the project, we will be restricting ourselves to a subset of it. Each data point in the dataset consists of 3 columns specifying question, title, body and the related tags for the question. The body contains the extracted HTML text from the web site.

Dataset Description	
ID	Integer
Title	String
Body	String
Tags	Space Separated List

A. Data Preprocessing

- **Remove HTML tags:** Since the data was directly scraped from the website, it contains HTML tags which are present in every data point and create noise and hence, needs to be removed.
- **Remove extra whitespaces and other special characters:** There were some user generated extra spaces and special characters which were contributing to the noise, and were removed.
- **Lowercase all texts:** To standardise the dataset, we converted all the text to Lowercase.
- **Convert numeric form to number words:** Since some questions contained numeric data which can't be processed in text processing. Hence, it was converted to number words.
- **Remove stopwords:** For text preprocessing words such as "as", "is", "a", "the" etc. creates noise and hence, were removed.
- **Remove duplicate entries:** There were duplicate data poits in the dataset, which would have affected our prediction later, and hence, were deleted.
- **Vectorization:** convert a list of tags into a set of columns denoting the presence of the tags.
- **Lemmatization:** Removing the inflectional endings and stored the base or dictionary form of a word.
- **Tokenization:** Breaks the raw text into words, sentences called tokens. These tokens help in understanding the context and developing the model.
- **Class normalization:** Converting all those words that resemble the same technology, e.g. converting "ReactJS", "React-JS", "React-Js", "react-JS" etc. to "reactjs".
- **Convert accented characters to ASCII characters:** Convert accented characters to its equivalent. For example, covert ė, ę to its ASCII equivalent "e".
- **Expand contractions:**Expand words like "don't", "can't" to "do not" and "can not" respectively.

B. Data Visualization

Initially, data size that we used for sampling was 2 million. Total of 232713 duplicate entiries was found. After removing duplicate entries we had 1767287 entries. We then plotted graph for frequency vs number of tags and frequency of tag of top 30 in the dataset.

1) Maximum Number of tags in any question - 5
2) Minimum Number of tags in any question - 1
3) Total Number of unique tags - 38427
4) Most Occuring tag - C#

We then analysed how many number of questions were covered with respect to the number of tags.

After analysing the graph, we decided to use top 6000 tags as it was covering 99.04 per cent of questions.

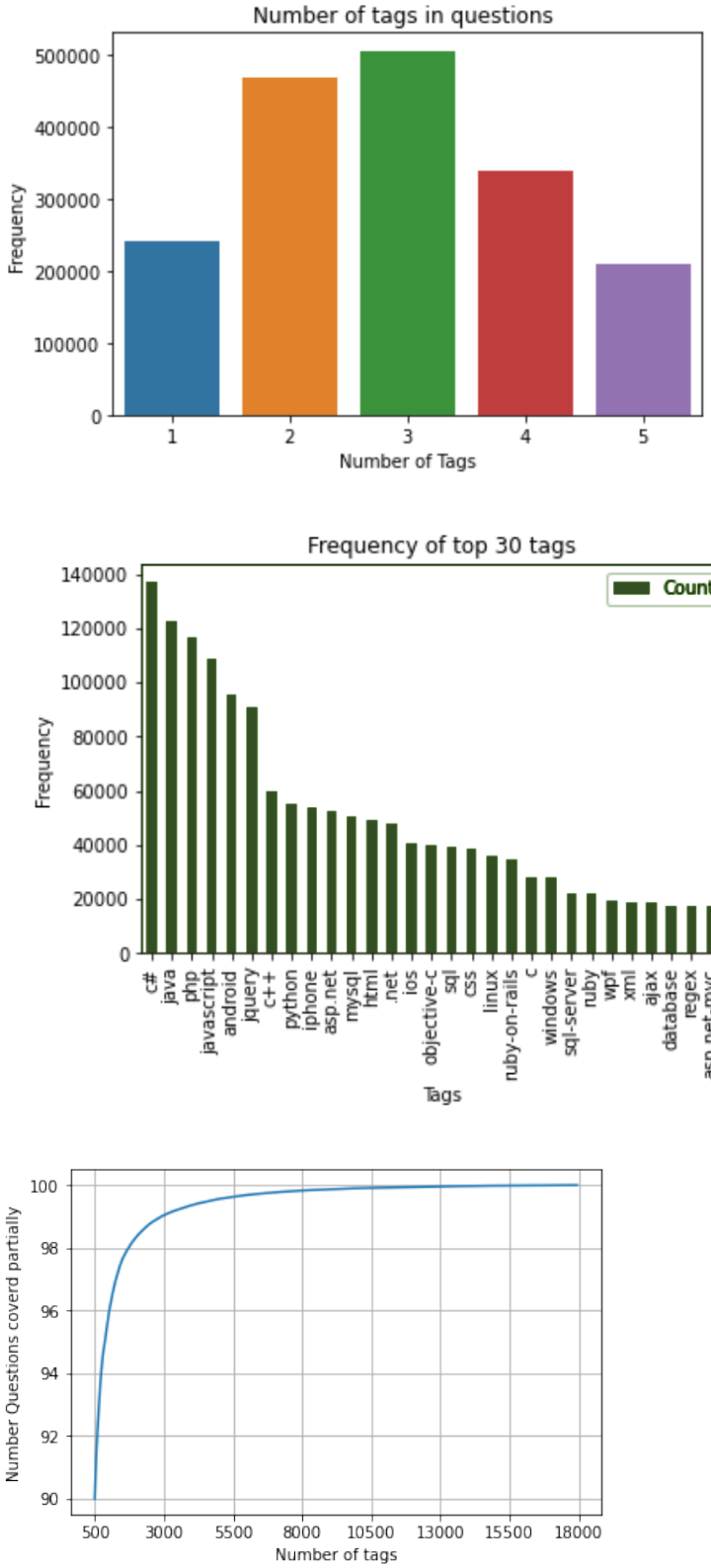

Fig. 1. Number of tags vs Number of Questions covered

V. Methodology

A. Data Collection

Firstly we needed to collect data to start working. We visited the different website over the internet to find the data of our need. Finally, we were able to find the data on Kaggle.

B. Data Preprocessing

We then moved on to processing our data to remove unwanted symbols, spaces, duplicate entries etc. We applied multiple preprocessing techniques, as mentioned above in the data preprocessing section.

C. Working with Different Machine Learning Models

Next, we started training our model on the data set. We have vectorized both training data and class labels in the preprocessing of the data, and we are training our models on these data.

D. Classifiers

We ran all the models mentioned below for ten different data sizes, and then we reported Accuracy, Macro F1 score, Micro F1 score and hamming loss for each size. We then obtained following results.

1) Logistic Regression:
We used the sklearn library for implementing One-vs-Rest and logistic regression and grid search. We have applied grid search to find the optimum value of learning rate for each data size.We then obtained following results.

Analysis				
Size	Accuracy	Macro F1	Micro F1	H loss
10000	0.1652	0.2201	0.3810	0.0033
20000	0.1671	0.2209	0.3903	0.0033
30000	0.1699	0.2188	0.3821	0.0033
40000	0.1754	0.2158	0.3739	0.0032
50000	0.1820	0.3583	0.2001	0.0032
60000	0.1829	0.3783	0.2162	0.0032
70000	0.1825	0.3023	0.3087	0.0032
80000	0.1824	0.2127	0.3787	0.0032
90000	0.1632	0.13	0.2751	0.0032
100000	0.1570	0.0393	0.2354	0.0032

Table 1: Analysis for Logistic Regression

2) SVM:
We used the sklearn library for implementing One-vs-Rest and SVM. We then obtained following results.

Analysis				
Size	Accuracy	Macro F1	Micro F1	H loss
10000	0.1932	0.2234	0.3987	0.0031
20000	0.1948	0.2346	0.4013	0.0031
30000	0.2031	0.2563	0.4071	0.0030
40000	0.2113	0.2603	0.4139	0.0029
50000	0.2189	0.2521	0.4201	0.0028
60000	0.2249	0.2464	0.4239	0.0028
70000	0.2254	0.2455	0.4189	0.0028
80000	0.2236	0.2442	0.4137	0.0028
90000	0.2201	0.2454	0.4141	0.0029
100000	0.2167	0.2588	0.4207	0.0029

Table 2: Analysis for SVM

3) Decision Tree:
We used the sklearn library for implementing One-vs-Rest and decision tree and grid search. We have applied grid search to find the optimum value of depth for each data size. We then obtained following results.

Analysis				
Size	Accuracy	Macro F1	Micro F1	H loss
10000	0.1521	0.2671	0.3853	0.0039
20000	0.1586	0.2784	0.3957	0.0039
30000	0.1578	0.2983	0.3966	0.0039
40000	0.1551	0.3008	0.3972	0.0039
50000	0.1601	0.3032	0.3999	0.0038
60000	0.1657	0.2952	0.4030	0.0038
70000	0.1632	0.3012	0.4027	0.0039
80000	0.1594	0.3019	0.40173	0.0040
90000	0.1601	0.2999	0.4023	0.0039
100000	0.1612	0.2981	0.4038	0.0039

Table 3: Analysis for Decision Tree

4) Bernoulli Naive Bayes: We used the sklearn library for implementing One-vs-Rest and BernoulliNB. We have applied grid search to find the optimum value of depth for each data size. We then obtained following results.

Analysis				
Size	Accuracy	Macro F1	Micro F1	H loss
10000	0.1061	0.0132	0.1767	0.0034
20000	0.1180	0.01557	0.1979	0.0043
30000	0.1175	0.0164	0.1988	0.0061
40000	0.1163	0.0268	0.2100	0.0066
50000	0.1153	0.0294	0.2178	0.0063
60000	0.1120	0.0375	0.2268	0.0082
70000	0.1068	0.0382	0.2129	0.0091
80000	0.1043	0.04611	0.2063	0.0101
90000	0.153	0.1635	0.3618	0.0061
100000	0.2167	0.2588	0.4207	0.0029

Table 4: Analysis for Bernoulli Naive Bayes

VI. Result and Analysis

We evaluated our problem using 4 different score metrics namely Accuracy, Macro F1 score, Micro F1 score, hamming loss for 5500 tags over diffferent datasize.
The table of different accuracy is as follows

Analysis				
Size	Log. Reg.	SVM	Bernoulli NB	Deci. Tree
10000	0.1652	0.1932	0.1061	0.1521
20000	0.1671	0.1948	0.1180	0.1586
30000	0.1699	0.2031	0.1175	0.1578
40000	0.1754	0.2113	0.1163	0.1551
50000	0.1820	0.2189	0.1153	0.1601
60000	0.1829	0.2249	0.1120	0.1657
70000	0.1825	0.2254	0.1068	0.1632
80000	0.1824	0.2236	0.1043	0.1594
90000	0.1632	0.2201	0.153	0.1601
100000	0.1570	0.2167	0.2167	0.1612

Table 4: Overall Analysis

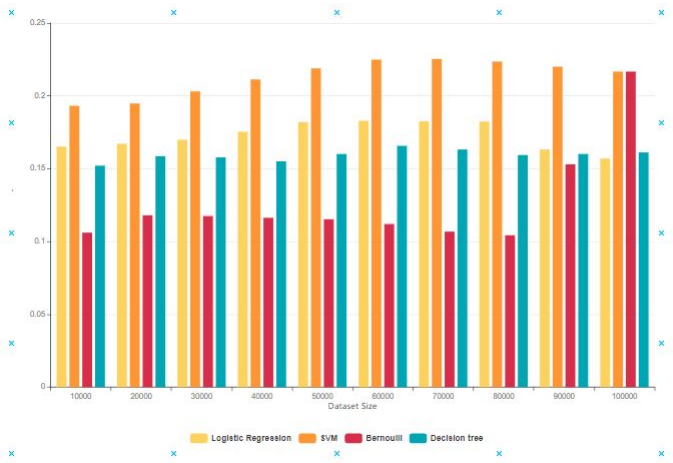

Fig. 2. Histogram of accuracy of different models is as follows

We can easily observe from the above histogram as well as other parameters(F1 score and hamming loss) that the best performing model of all of them is SVM.
As the size of data becomes 100000, accuracy of SVM and Bernoulli Naive Bayes became almost equal. Thus, we can say that as the datasize increases, performance of Bernoulli Naive Bayes also increased.
Performance of Decision tree and Logistic regression's performance remained independent of the datasize that we used for the analysis

VII. Conclusion

A. Learnings from the project

- We learned how to handle multi-label and multi-class dataset.
- We got to understand various text pre-processing techniques like lemmatization, tokenization etc. work.
- We learned new ways to visualize and understand large datasets using graphs and plots.
- We also learned new methods to analyze errors in prediction. For example: hamming loss.

B. Work Left to do

- Dive into more pre-processing strategies that could help us in interpreting data in better form for the machine learning models.
- Trying more machine learing models that work on text classification to acheive better and wide score metrics.
- Further, working with the current models to perform more hyperparameter tuning to get more precise and better results.

C. Contribution

- Nitin: Literature review, data visiualization, feature extraction, working with Support Vector Machine(SVM), report documentation.
- Sandeep: Data visiualization, data preprocessing, feature extraction, working with Logistic Regression, report documentation
- Navya: Literature review, data preprocessing, working with Bernoulli Naive Bayes, working with Decision Tree, report documentation.

References

[1] **Predicting tags for stack overflow questions using different classifiers** - https://ieeexplore.ieee.org/document/8389059
[2] **Predicting Tags for StackOverflow Questions** - http://cs229.stanford.edu/proj2013/SchusterZhuCheng-PredictingTagsforStackOverflowQuestions.pdf
[3] **Predicting tags for the question in StackOverflow** - https://medium.com/datadriveninvestor/predicting-tags-for-the-questions-in-stack-overflow-29438367261e
[4] **agStack: Automated System for Predicting Tags in StackOverflow** - https://www.researchgate.net/publication/338370635_TagStack_Automated_System_for
[5] **Sklearn One Vs Rest Classifier** - https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html
[6] **Sklearn Support Vector Machine and Logistic Regression** - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[7] **Sklearn Bernoulli Naive Bayes** - https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html