Query Tagging Using Linear Models For Stackoverflow

Nithin Krishna Gowda
Computer and Information Science
University of Oregon
Eugene, Oregon
ngowda@uoregon.edu
Github link - https://github.com/gnithink/ML572-final-project

***Abstract*—Text classification is one of the most fundamental concepts in Natural Language Processing. The concepts involved in text classification are very intuitive and easy to understand, yet it is so powerful and versatile that it has applications in almost all fields known to humans. In the agile environment of industry where customer feedback is the driving force behind innovation and improvement of products, text classification analysis of customer reviews completes the feedback loop between the customers and manufacturers. Massive amounts of text can be processed, tagged and evaluated to gain insights into the quality of the product at hand by a small team of people. This is the motivation behind exploring the concepts of query tagging of questions posted in stack overflow. Linear models like Logistic Regression, Support Vector Machines and Ridge Classifiers are used to tag questions posted by programmers on stack overflow and the performance of these models are evaluated.**

***Index Terms*—Natural Language Processing, query or question tagging, Term Frequecy Inverse Document Frequency(TF-IDF), Bag of Words, Support Vector Machines, Logistic Regression, Stemming, Lemmatization.**

I. Introduction

This paper is written to give a general idea of the steps involved in classifying text. Query Tagging can be considered as a multilabel classification problem where each query is given multiple tags based on the text in the question. This is similar to sentiment classification except that there are hundred tags or classes and each question can be tagged with multiple tags. We go through the entire methodology and pipeline starting from the raw text, methods to preprocess the text, convert the text to vectors which can be processed by the computer. Since, especially in NLP there is a tendency to generate sparse vectors of very large size, methods to reduce the size of these vectors and remove irrelevant features are also discussed. Finally some parameters to analyze the performance of the different linear models like precision, recall, accuracy and f1-score are also discussed. Linear models that can handle extremely sparse data are used here. In future work, the concepts involved in deep learning approach to classify text are also discussed where the extremely powerful method of convolution is used to extract features from word embeddings. This is also known to give better performance as it captures more properties of text like context, ordering etc. This method produces very dense vectors.

II. Dataset

The dataset is taken from kaggle [2] from the stackoverflow dataset. The consists of only two columns - the input is the query posted by the user and the output is the list of tags associated with the query. There are more The training, validation and the test set consist of 80k, 30k and 20k entries respectively. The text is in its raw form with all the typographical errors and incomplete entries. A lot of preprocessing needs to be done to convert the text into insightful feature vectors on which the different linear models can be used. This dataset is a good starting point to get a clear understanding of converting text to meaningful feature vectors on which different models can be trained and tested.

III. Methodology

All the concepts involved in the pre-processing of text will be convered in this section. As expected, text input from humans depends on the writing style of each user and same query or question can be asked in multiple ways. So by intuition if the model can predict similar tags for queries covering similar topics or similar questions that have been worded differently then our model is doing a good job. Just to give an overview of the methodology, the concepts of Tokenization, N-Gram Tokenization, stemming, lemmatization, bag of words and TF-IDF(Term Frequency and Inverse Document Frequency) will be discussed in relation to text processing. The methodology of creating feature vectors for deep learning approach is also discussed at the end in 'Future work' section. Logistic Regression will be explained briefly. The functions from the inbuilt libraries used to implement these operations will also be specified as and when required.

A. Tokenization

Text can be considered as a sequence of characters or sequence of words or sequence of phrases etc. Each meaningful unit of text is called a token and the process of converting a piece of text into a sequence of tokens is call tokenization. For the task at hand, each word can be considered as a token. The tokens can be extracted from the text by using many different approaches like the white space tokenizer, tokenzation based on punctuation or Treebank tokenization(inbuilt function in

NLTK library). These methods are briefly discussed but the TreeBankTokenizer is used in this method.

- **Whitespace Tokenization** is the process of extracting tokens separated by white space. The function WhiteSpaceTokenizer as part of the NLTK library can be used to achieve this process. However this approach leads to lots of unexpected tokens and meaningful information about the query is lost.
- **WordPunct Tokenizer** creates tokens separated by punctuation marks. This appraoch leads to separation of words with apostrophes into two individual tokens which is not required.
- **TreeBank Tokenizer** is part of the NLTK library which is written for tokenization with the rules or heuristics of English grammar in mind. This creates meaningful tokens which can be used for further analysis.

B. Token Normalization

Token Normalization is process of extracting the root word from different versions of the word. Eg: the word 'talks' or 'talked' needs to be converted to the root word 'talk'. This can be achieved in many ways - the process of stemming and lemmatization are discussed here.

- **Stemming** is a process of removing and replacing suffixes to get to the root form of the word, which is called the stem. It usually refers to heuristic that chop off suffixes or replaces them. PorterStemmer is one of the oldest stemmer which uses regular expressions to build a sequence of steps through which the root word is extracted.
- **Lemmatization** refer to extracting the root word from each token properly with the use of vocabularies and morphological analysis. The root word returned is called as the lemma. The WordNet lemmatizer is a popular one available in the NLTK library.
- Further, we can normalize the text by converting capital letters to small letters. Capitalization in the middle of a sentence can be retained as it could be important information like acronyms or proper nouns. This can be achieved by writing more heuristics as required. **WordNet lemmatizer is used in this project**.

C. Token Vectorization

Once we have extracted the meaningful tokens from the text we cannot feed it to the model right away as mathematical models do not understand text. Each query should be converted into feature vectors of the same size. This can be achieved by using the **bag of words** model, **N-gram** approach to retain ordering of words information followed by **TF-IDF vectorization**.

- **Bag of words** model involves generating one hot vectors for each word and summing up the one hot vectors for each query. Bag of words is the bag of all the unique words that are available in the database of the training, test and the validation sets. For each query from stackoverflow we replace with ones in that vector. Say for example there are 100k unique words in the database and the query size is 50 words. Each vector will have a size of 100k zeros and replaced with 50 ones for that specific query. But by using the bag of words model, we lose the ordering of the words which is critical information.
- The ordering of words information can retained by using the **N-gram approach**. Here, we not only look for individual words, but also for existence of pairs of words or triplets or more. But doing so, the size of the feature vector grows exponentially and it becomes computationally very expensive or impossible to process this data.
- This problem can be overcome removing specific features in the vectors. These vectors are extremely sparse and lot of words like 'is', 'the', 'a' etc. can be removed from the text as they do not make any meaningful contribution since they appear in almost queries. These are called **stop words** which can be removed. Also there are some words which appear very rarely in the database which are basically typos and some errors. Even those features can be removed. We care about the **medium frequency terms**. So how do we determine the which are bad features and which are good features among these medium frequency terms ? This is where the measuring coefficient TF-IDF is used.
- **TF-IDF (Term Frequency - Inverse Document Frequency)** - As the coefficient name says, we measure the number of times each term appears in each each query. This is the term freqeuncy. Then we calculate the normaized term frequency. The inverse document frequency is the reciprocal of number of times each term appears in the entire database divided by the number of queries. TF-IDF value gives us a measure of how frequent the feature or token has appeared in each query and how rare the the feature has appeared in all the reviews or documents together. These are the distinguishing tokens for each query and help in determining the tags for that query. The *tfidfvectorizer* in the NLTK library performs all these operations.

tf(t, d) – frequency for term (or n-gram) t in document d
f - raw count of frequency in each query.
N $= |D|-$ total number of documents in corpus $|\{d \in D : t \in d\}|-$ number of documents where the term t appears

$$tf(t,d) = f_{\mathrm{t,d}} / \sum_{t' \in d} f_{\mathrm{t',d}} \quad (1)$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \quad (2)$$

- By performing all these operations, each query has been converted into a meaningful vector on which the linear models can be trained and their performance can be analyzed. Even now the vectors are extremely sparse and that is why linear models are chosen as they are extremely fast to train with sparse vectors. We will discuss logistic regression, multilabel binarization and one-vs-rest classifier in the Experiments section.

IV. Experiments and Model architecture

After pre-processing the text, the each input query or question is in the form of an extremely sparse vector. The output value for each vector is a list of tags for that specific query. We want to transform these tags into vectors. After this task, our data is in a format which can be used to train various models.

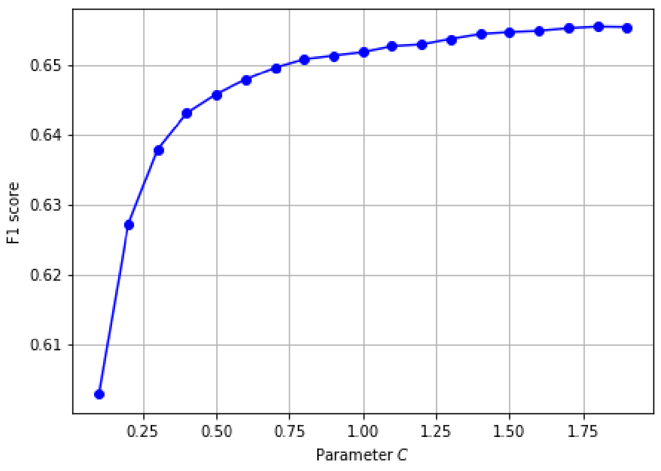
- **Multi Label Binarizer** - The task of converting outputs tags to vectors can be achieved using the Multi Label Binarizer from the sklearn library. Since there are hundred different tags in the output, the output for each input vector is converted to a 1 x 100 vector with ones for the tags associated with that query.
- **One Vs Rest Classification** - Since this is a multilabel classification problem, we use the one vs rest classification model to classify text. This approach involves fitting one classifier per class. For each classifier the class is fitted against all the other classes. i.e. for each input vector, when training the classifier, the prediction for one specific class is considered positive (eg: if the prediction is tag 'python' and that tag is part of the actual output for the input vector, then it is considered a correct prediction and all other class predictions are considered negative). This approach is also computationally efficient and can be easily interpreted. Since each class is represented by one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier.
- **Logisitic Regression** - Since logistic regression and other linear models are excellent at handling sparse vectors, it is chosen as the primary classifier for the one vs rest classification model.
- In summary, One vs rest classification model used with Logistic regression as the primary classifier to predict tags for stackoverflow queries. The input vectors are created using bag-of-words and TF-IDF values. The model is trained using both the formats and the results are summarized for the validation and the test set.

V. Tuning Hyperparameters

The hyperparameter C is the inverse of regularization strength and it must be a positive float value. Smaller values specify higher regularization. We trained the model with values of C in the range of 0.1 to 2.0 and chose the value with the highest F-1 score. The chosen value for C was 1. The graph for parameter C against the F-1 score is given below.

VI. Results and Analysis

Assigning the right tags to new incoming queries optimizes the search process when users are looking for answers to a specific question and the search results are more relevant. This



(a) C vs F-1 Score For Logisitic Regression

is the main reason why stackoverflow is the default website for programming queries. Considering the dataset at hand, each input query has multiple tags. If our model can predict at least some relevant and correct tags for the incoming new queries this would improve the search process by many fold and aid in finding relevant search results.

- Empirical observation of the predictions and the actual values of the test set shows that relevant tags are assigned to the queries. However, in some cases, out of the many tags assigned to each query, only some of those tags have been assigned. And some of the queries do not have any tags if the model was not sure about what tags to assign. This empirical observation hints that even if the accuracy results are low for the model which is about 0.36, they have a good precision and recall score which can be inferred from the F-1 score of 0.66.

A. *Parameters for Analysis*

The general parameters to determine the performance of a model are **accuracy, precision, recall, F-1 score** and **ROC curve**. All these parameters are given a brief overview to help the reader get a better understanding of these concepts.

- **Accuracy** - Number of correct predictions among the total number of predictions.
- **Precision** - Intuitively, it refers to the percentage of the results that are actually relevant.
- **Recall** - Intuitively, it refers to the percentage of the actual relevant results that are correctly classified by the algorithm.
- **F-1 Score** - Harmonic mean of the precision and the recall. Maximizing this value should be the objective of any model. This can be referred as an indicator of the performance of the model.
- **ROC curve** - It is a plot of the True Positive Rate on the Y-axis and False Positive Rate on the X-axis. The Top Left point on the curve is the ideal point. We want to try to maximize the steepness of the curve (maximize

the True Positive Rate) and minimize the False Positive Rate.

- **Micro and Macro Weighted ROC curve** - Micro-Average-Weighted ROC curve is a preferred metric for multilabel classification problem rather than the Macro-Average-Weighted ROC curve. The former will aggregate the contributions of all classes to compute the average metric for the F-1 score to overcome class imbalance (few specific classes have more input values).

$$Precision = \frac{TruePositive}{ActualResults} \tag{3}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{4}$$

$$Recall = \frac{TruePositive}{PredictedResults} \tag{5}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{6}$$

$$Accuracy = \frac{TruePositive + TrueNegative}{Total} \tag{7}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

B. Obtained Results for Model Performance

The accuracy, average-precision and f-1 scores are similar for TF-IDF and bag-of-words format of the vectors for **one vs rest classification using logisitic regression**. The results are tabulated below:

TABLE I: Performance Metrics For Multi-label Classification - Validation set

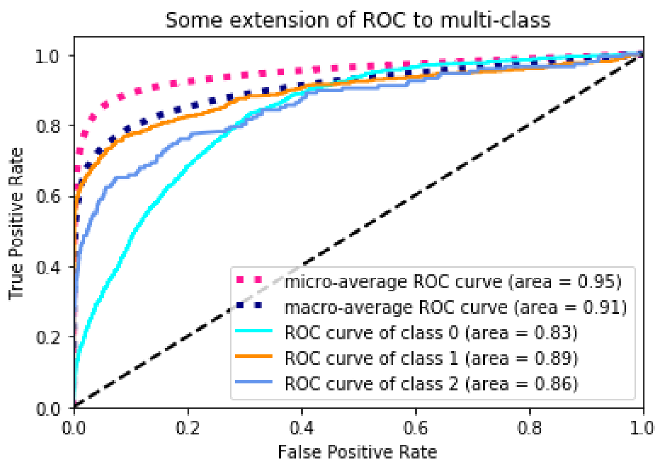
Metric	Bag-of-words	TF-IDF
Accuracy Score	0.37	0.38
F-1 Score	0.66	0.67
Average Precision Score	0.37	0.37

TABLE II: Performance Metrics For Multi-label Classification - Test set

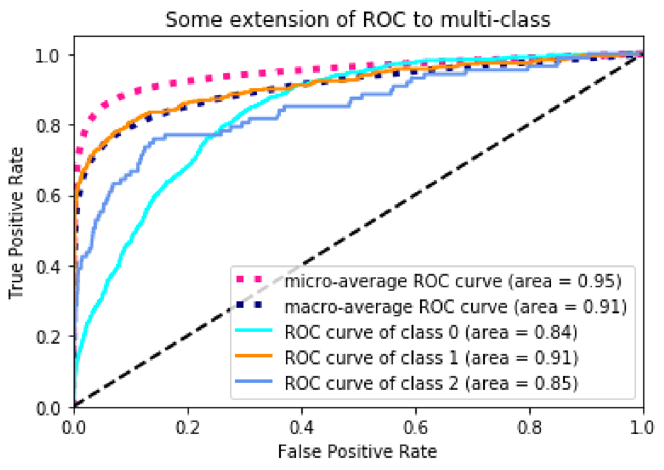
Metric	Bag-of-words	TF-IDF
Accuracy Score	0.37	0.38
F-1 Score	0.66	0.67
Average Precision Score	0.37	0.38

C. ROC - curve

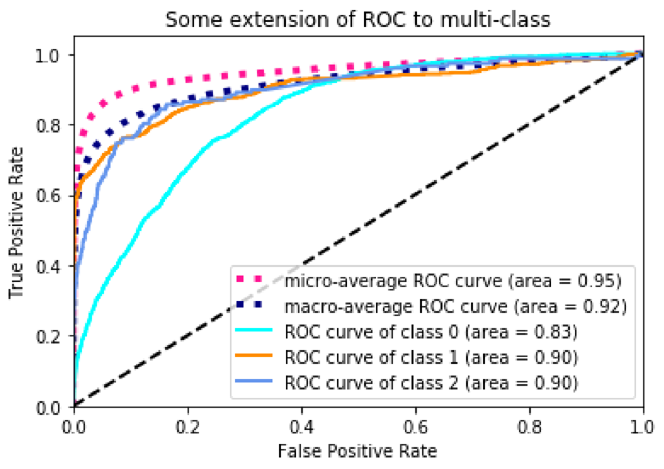
The ROC curve indicates that our model seems to perform well as the curve is steep which represents a high true positive rate.



(a) Bag-of-Words Validation Set



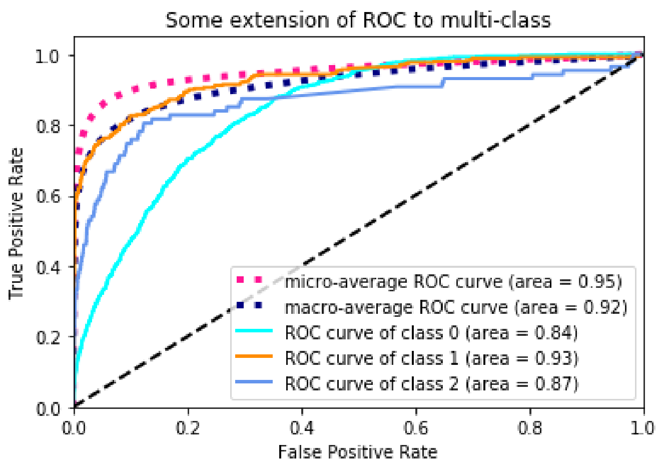
(b) TF-IDF Validation Set



(a) Bag-of-Words Test Set

VII. Future Work

Similar performance metrics can be obtained for different models like SVM and Naive - Bayes classifiers as the primary classifiers for one vs rest classification model. The primary



(a) TF-IDF Test Set

classifier can be chosen depending on the model that gives the best performance (best F-1 score and steeper ROC curve). Here we would like to give a brief overview of different approach to text classification using Multi-Layer Perceptron. This architecture is well suited for dense vectors.

A. Text Classification using Multi-Layer Perceptron

The word embeddings of the words using **word2vec** [9] can be used to generate dense vectors. The embedding for each word in the query is stacked vertically on top of each other. Then we can use 1D (One-Dimension) convolution operation to generate values n-grams. Depending on the size of the vector we want to generate and the important n-grams that we are looking for, we can create as many convolution filters as we want. For a specific convolution filter traversed through the entire query, we choose the maximum value among all the values generated by the convolution filter (maximum value represents that n-gram or n-gram with similar meaning is present in the query). Say we use 300 convolution filters, repeating this process on all the queries in the dataset generates 1 x 300 size vector for each query. This data can then be used to train a multi-layer preceptron with hidden layers. Empirical results show that dense vector representation models perform better than linear models.

VIII. CONCLUSION

In this paper we got a chance to explore the entire text classification pipeline starting from the raw text to actual prediction and analysis of the results. We also saw a brief overview of the dense vector method to classify text in the future work section. Similar text classification method can be used in different domains wherever multilabel text classification is needed.

REFERENCES

[1] Denzil Correa, Ashish SurekaIndraprastha,Institute of Information TechnologyIIIT-Delhi “Fit or Unfit : Analysis and Prediction of ‘Closed Questions’on Stack Overflow,” Publication:COSN ’13: Proceedings of the first ACM conference on Online social networksOctober 2013 Pages 201–212 https://doi.org/10.1145/2512938.2512954.

[2] https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/overview/

[3] Constantinos Boulis Mari Ostendorf, Text Classification by Augmenting the Bag-of-Words Representation with Redundancy-Compensated Bigrams, Proceedings of the Workshop on Feature Selection for Data Mining: Interfacing Machine Learning and Statistics, April 23, 2005 .

[4] Wen Zhang, Taketoshi Yoshida, Xijin Tang, A comparative study of TF-IDF, LSI and multi-words for text classification, Expert Systems with Applications 2010.

[5] Qiaozhu Mei and Yi Zhang. 2008. Automatic web tagging and person tagging using language models. In Advanced Data Mining and Applications, pages 741–748. Springer.

[6] Sanjay Sood, Sara Owsley, Kristian J Hammond, and Larry Birnbaum. 2007. Tagassist: Automatic tag suggestion for blog posts. In ICWSM.

[7] Clayton Stanley and Michael D Byrne. 2013. Predicting tags for stack-overflow posts. In Proceedings of ICCM 2013.

[8] Zhiyuan Liu, Xinxiong Chen, and Maosong Sun. 2011. A simple word trigger method for social tag suggestion. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 1577–1588. Association for Computational Linguistics.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Efficient Estimation of Word Representations in Vector Space, https://arxiv.org/abs/1301.3781