Autonomous Tagging of StackOverflow Questions - EE 628 Final Project

Rohan Chandra

Abstract

StackOverflow has become an essential place of discussion for programmers and has helped many engage in meaningful discussion and learning. An easy way to navigate the website is through filtering the various questions posted by their tags. Here I discuss a neural network that solves the problem of assigning tags to a newly encountered question.

1 Introduction

The quantity of information available on StackOverflow has increased tremendously, but there remains no efficient way of automatically classifying posts based on their content.Users are asked to tag their own questions which may not result in the best, most relevant viewership, nor the best categorization as users may not tag the questions in the intended way. A system that handles this for the user in a predefined and standardized manner would help get answers faster and maintain order in the database.This paper outlines a method that automatically allocates tags for a given question. A neural network is used to tag the questions. The remainder of this paper is organized as follows: Section 2 outlines previous work on text classification, Section 3 describes the methodology used here. Section 4 summarizes the results. Section 5 discusses key insights. Section 6 talks about possible future work.

2 Recent Work

Text classification is the method of classifying text into various classes or tags based on its context. It is based on the application

of natural language processing and machine learning algorithms. It is a multilabel classification problem. Work that has been done in this area is the motivation behind this paper.

Algorithmic Programming Language Identification based approach is one such popular technique. Various syntax highlighting tools such as Google Code Prettify will automatically highlight syntax given some code. However, these tools do not actually identify languages; instead, they use heuristics that will make the highlighting work well. In the case of Google Code Prettify, broad grammars (such as Clike, Bash-like, and Xml-like) are preprogrammed. These grammars are then used to scan code, and the best matching grammar is used in highlighting. Clearly, languages that share a grammar cannot be distinguished between. More relevant is SourceClassifier, which attempts to identify a programming language given some code. However, it relies on a simple Bayesian classifier. Its strength is therefore limited to the quality of training data, and it can easily be thrown off by strings and comments.

The other popular approach is automatic content tagging. In one paper [7], Xia et al. propose an automatic tag recommendation algorithm TagCombine. There are three components of TagCombine, each of which tries to assign the best tags to untagged objects: (1) multi-label ranking component, which predicts tags using a multi-label learning algorithm, (2) similarity based ranking component, which uses similar objects to recommend tags, and (3) tag-term based ranking component, which analyzes the historical affinity of tags to certain words in order to suggest tags. The recommendation algorithm methodically computes various weighted sums of the three components to attempt to find the best overall model. Another algorithm by Wang et al. In second paper [21] proposes a tag recommendation system dubbed En-TagRec. The proposed EnTagRec computes tag Algorithmic Programming Language Identification based approach is one such popular technique. Various syntax highlighting tools such as Google Code Prettify will automatically highlight syntax given some code. However, these tools do not actually identify languages; instead, they use heuristics that will make the highlighting work well. In the case of Google Code Prettify, broad grammars (such as Clike, Bash-like, and Xml-like) are preprogrammed. These grammars are then used to scan code, and the best matching grammar is used in highlighting. Clearly, languages that share a grammar cannot be distin-guished between. More relevant is SourceClassifier, which attempts to identify a pro-

gramming language given some code. However, it relies on a simple Bayesian classifier. Its strength is therefore limited to the quality of training data, and it can easily be thrown off by strings and comments.

The other popular approach is automatic content tagging. In one paper [7], Xia et al. propose an automatic tag recommendation algorithm TagCombine. There are three components of TagCombine, each of which tries to assign the best tags to untagged objects: (1) multi-label ranking component, which predicts tags using a multi-label learning algorithm, (2) similarity based ranking component, which uses similar objects to recommend tags, and (3) tag-term based ranking component, which analyzes the historical affinity of tags to certain words in order to suggest tags. The recommendation algorithm methodically computes various weighted sums of the three components to attempt to find the best overall model. Another algorithm by Wang et al. In second paper [21] proposes a tag recommendation system dubbed EnTagRec. The proposed EnTagRec computes tag prob-ability scores using two separate methods, Bayesian Inference and Frequentist Inference, and then takes a weighted sum of the probability scores. Bayesian Inference relies on a posts textual data to compute the probability that a given tag is associated with the post. EnTagRec formulates posts into a bagof-words model and then trains a Labeled Latent Dirichlet Allocation model which is used to compute tag probability scores for a post. The Frequentist Inference approach infers a set of tags after some preprocessing of a post, and then utilizes a network of tags to select additional tags that are similar to the ones in the set. The network of tags is constructed with the tags as nodes and weighted edges between two tags based on the Jaccard similarity of the posts that contain those tags.

3 Methodology

3.1 Dataset Description

The Stacksimple dataset is a set of StackOverFlow programming questions. It was compiled during 2006 and has a large subset of the entire database of questions on the website. It contains two files that are organized with columns containing the features common to each question. The dataset consists of 2 CSV files: questions.csv which Contains the Id, CreationDate, ClosedDate, DeletionDate, Score, OwnerUserId, AnswerCount and a second file that contains the question tags mapped to the question ID.

3.1.1 Preprocessing

The data files were read into a script and the columns 'score', 'owner_id', 'ans_count' and 'tag_id' are extracted from the questions.csv while the corresponding tags are simultaneously extracted from the other file. We only use the 50 most commonly encountered tags, so those are extracted from the data. The 'score' and 'ans_count' columns are added to the first two columns of the input, and the 'owner_id' is split and added character-wise to the input list, along with a padding of zeros.

3.2 Model

The model used is a neural network implemented in tensorflow with input_size: 50000, 4 hidden layers with hidden_layer_sizes: [12,14,26,46]. The net is initialized as usual. MSE is used to calculate the loss. The initial few attempts resulted in errors, and the structure was modified according to [16]. Various combinations of hidden layer sized were used, accounting for the fact that the result was required in 5000 epochs. The final learning rate chosen was 0.001. The final accuracy achieved was 0.80 for train and 0.80 for test after 5000 epochs, a huge step up from any other combinations of hidden layer sizes. The model works by initializing a set of parameters for each layer such as weights and biases that are multiplied with the pre-processed input vectors. The results are passed from one layer to another until the output layers outputs the final results, which in this case is a dictionary containing the predicted targets.

4 Results

The loss halves in the first 500 epochs and slows down, and continues to reduce more and more gradually all the way to epoch number 5000. The metric used was Mean Squared Error and the Optimizer was Adam. The model is fairly straightforward as compared to standard and older machine learning models used in the past by other authors. The training speed was impressive as well. However, the entire dataset was not considered and the impact of scaling the model on it's training speed is yet to be tested.

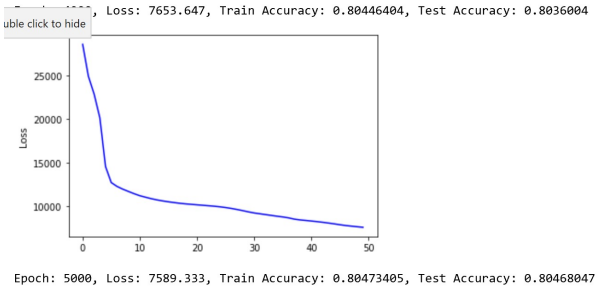

Figure 1: Accuracy of the neural net

5 Conclusion

With the advent of deep learning libraries such as mxnet, tensorflow and so on it has become increasingly simple to implement very powerful text classification algorithms with very little know-how or engineering effort. However, feature engineering for these models is cumbersome and a significant overhead in most cases.

6 Future Work

This project used only the top 50 tags. In the future, I would like to use more tags to further improve the model and evaluate it more accurately. Also, a more detailed description of what is being processed and a better display of the results. Such as, displaying a sample question and it's corresponding tag predictions alongside the actual tags. This could be done with colorful plots, with a good readability. Different preprocessing techniques and standard machine learning models are in the pipeline as well, along with a more thorough analysis and examination of the source data, as paucity of time did not permit that this time.

Acknowledgments

I would like to thank my Professor Sergul Aydore for giving me the go-ahead to do this project, and for all the guidance and support she has offered me throughout the semester. I would also like to thank all my classmates, who helped me understand the topics being discussed that much better.

References

[1] www.kaggle.com/stackoverflow/stacksample. 2016.

[2] github.com/bassirishabh/stackoverflow. 2018.

[3] www.stackoverflow.com. 2018.

[4] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

[5] Loren Collingwood, Timothy Jurka, Amber E Boydstun, Emil-iano Grossman, WH van Atteveldt, et al. Rtexttools: A supervised learning package for text classification. 2013.

[6] Chris Ding, Tao Li, Wei Peng, and Haesun Park. Orthogonal nonnegative matrix t-factorizations for clustering. pages 126– 135, 2006.

[7] M Eric, A Klimovic, and V Zhong. ml nlp: Autonomous tagging of stack overflow questions, 2014

[8] Amir Fallahi and Shahram Jafari. An expert system for detection of breast cancer using data preprocessing and bayesian network. International Journal of Advanced Science and Technol-ogy, 34:65–70, 2011.

[9] Ingo Feinerer. Introduction to the tm package text mining in r. 2013-12-01]. http://www, dainf, ct. utfpr, edu. br/kaestner/Min-eracao/RDataMining/tm, pdf, 2017.

[10] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. pages 137– 142, 1998.

[11] Thorsten Joachims. Transductive inference for text classification using support vector machines. 99:200–209, 1999.

[12] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. 14(2):1137– 1145, 1995.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Ima-genet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[14] M Eric, A Klimovic, and V Zhong. ml nlp: Autonomous tagging of stack overflow questions, 2014.

[15] Logan Short, Christopher Wong, and David Zeng. Tag recom-mendations in stackoverflow. Technical report, San Francisco: Stanford University, CS, 2014.

[16] https://github.com/erohkohl/question-tagging

[17] https://github.com/Utkarsh4430/Autonomous-Tagging-Of-Stack-Overflow-Questions/blob/master/predicting-tags.ipynb

[18] https://github.com/Drag97/Autonomous-Tagging-of-Stack-Overflow-Questions/blob/master/code.ipynb