Spring 2021

Source Code Comment Classification Artificial Intelligence

Cole Sutyak
cps46@zips.uakron.edu

Source Code Comment Classification Artificial Intelligence

Cole Sutyak
Williams Honors College

ABSTRACT

Source code comment classification is an important problem for future machine learning solutions. In particular, supervised machine learning solutions that have largely subjective data labels but are difficult to obtain the labels for. Machine learning problems are problems largely because of a lack of data. In machine learning solutions, it is better to have a large amount of mediocre data than it is to have a small amount of good data. While the mediocre data might not produce the best accuracy, it produces the best results because there is much more to learn from the problem.

In this project, data was collected from student comment code in computer science classes. This data was then sorted based on various tools in order to create automated source code classification. Various data categorization and sorting methods were explored, ultimately resulting in a process where assigned letter grade was used as a sorting label. Using python, CommentLabeler, and SortAndUnique tools were developed in order to automate the manual source code labeling process. State retention and error checking were also features that were added to streamline the process further.

The most important takeaway from this experience was that the amount of data is much more important than quality. In fact, mediocre data will provide better results with regard to machine learning because there is room for improvement and it proves machine learning as a solution.

TABLE OF CONTENTS

Page

INTRODUCTION

The classification of comments in source code is a time-consuming and laborious process if done manually. However, this task is essential to effectively managing data. The purpose of this project was to classify comments in source code using machine learning. The ultimate goal of this research was intentionally open-ended to allow for the exploration of multiple routes, applications, and solutions. The majority of the work was in the field of software engineering rather than researching potential methods to complete this task. The majority of the time spent on the project was spent in the development of a functional tool that could successfully classify source code and would improve as the result of machine learning. Therefore, the end result of this research project was a developed tool rather than definitive solutions to specific applications for this tool.

BACKGROUND

Following a conversation with Dr. Collard regarding potential project ideas as well as a personal interest inventory that I had conducted, Dr. Collard suggested that I take a look at a paper written by Kallis et al. entitled “Ticket Tagger: Machine Learning Driven Issue Classification.” This research project described a novel approach to the prioritization of comment code via machine learning. This interesting approach led to further conversations with Dr. Collard and the development of my proposal to create some form of AI in order to classify comment code. Following up with extensive research, I have not located any comment code text classification AI using deep learning.

However, a similar approach to this project was found using Java was done by Luca Pascarella, Magiel Bruntink, and Alberto Bacchelli as explained in their paper, “Classifying Code Comments in Java Software Systems.” In this approach, they use supervised machine learning testing two different classes of supervised classification methods: probabilistic classifiers such as naive Bayes or naive Bayes Multinomial, and decision tree algorithms such as J48 and random forest. Their approach largely mirrors this project because Fasttext is an expansion and optimization of the decision tree algorithm. However, while their work has to do with comment classification, their classification is vastly different. Their comment classification is explicitly concerned with the purpose of the comment, while this project is concerned with overall comment grade or health.

A comment grade, or health, is a general scoring of a code comment. Research projects performed by Tenny as well as Woodfield et al., indicate that code comments are integral for code readability. However, these studies fall short when deciding the question, what makes a comment good? A large part of this project was spent on this specific question.

For the purposes of this project, a good comment is defined as one that most efficiently completes a comment's purpose. This definition creates another question, what is a comment's purpose? This nuance could be debated, however for the scope of this project, the comment's purpose is "to make it easier for people to understand the code" This definition was obtained from research conducted by Yang et al. as reported in their paper, "A survey on research of code comment." This may seem intuitive, but it is a more advanced problem than it appears considering the code classification AI must label a comment good or bad based on these two disputed definitions. In summary, a good comment is one that most efficiently "makes it easier for people to understand the code." However, this definition has the potential to be quite subjective, varying from person to person and as a result, another problem is created. With this in mind, it is important to abstract the code classification AI will function based on the definition of what a good comment is.

Expanding on this premise there are often many comments which are necessary to the code base that aren't classified as good. One example of this is commented-out code. Commented-out code can provide a real benefit to code libraries that need examples of implementations or code lines that are frequently hot swapped. Examples of implementations could theoretically aid in understanding the code. This makes it, by one definition, potentially a good code comment, but for the simplicity of this project, since most comments that include this strategy do not aid in helping the reader understand the code, it is not categorized as such. Both of these examples of commented-out code are generally against proper coding practices, and while, by definition, they can theoretically be good comments, they are generally not, therefore the decision was made to omit these comments from the data.

Another problem with labeling a comment based on this definition is comment context.

The comments obtained for this project oftentimes did not have enough context to be labeled good or bad. The context, in this case, being the code surrounding the comments. The tools used for obtaining the comment data stripped all code from the comment, so some comments could be mislabeled because of lack of context. For example, if there were a comment such as “sets x to the area of a triangle”, and this had no relevance, meaning that the comment was not close to any x variable, or worse, x was not being set to the area of a triangle, then by definition, x is a bad comment. But without context “sets x to the area of a triangle” is a good comment because it efficiently explains to the reader what is in that comment section of code.

DESIGN

Once the background information was assimilated and a focus for the project had been established, it was time to make design decisions. All code for this project was developed in Python. This language was chosen because of the ease of using multiple libraries using pip. Pip is a package management system for installing and managing python. Prior to making the decision to utilize python, other options were explored, including C++ and C#. Although C# does not support Fasttext natively, it is a very robust and powerful language for developing windows applications in particular. However, this was not optimal for this project because it was decided in the early stages of development to go against the idea of creating a windows application. The reason is because much research and development would need to be done on best practices for a windows application which would defer the focus from the actual project.

There were two remaining choices, C++ and Python, both of which are languages that Fasttext explore natively. C++ was ultimately decided against, not because C++ is a bad language but because, in this case, Python is better. Python is better because of pip. Pip is a package-management system written in python for python that allows for easy installation and management of python packages. Installing C++ packages is not extremely complex, but with respect to research and development, python is much more suited to the job. Python's package manager allows the user to install and manage multiple complex packages quickly and seamlessly. This is more suited for this research project because multiple packages could be installed and tested to find optimal solutions.

In hindsight, C++ might have been a better choice because the focus of this project shifted from a solution explorer, to a software engineering one.

The tool used in this project was a python fast text classifier, "Fasttext". Fasttext is a

library for learning word embeddings and, more importantly in this case, text classification. Fasttext was created by Facebook's AI Research lab(FAIR) and has been shown to be on par in terms of accuracy with more complex solutions such as deep learning classifiers, as reported by Joulin et al. in their article, "Bag of Tricks for Efficient Text Classification."

Developing a unique deep learning application for this problem was also a solution explored. If the goal of this project is to find the most efficient solution to text classification of code comments, there is no real benefit in using deep learning models over Fasttext. This is because while deep learning models often are on par with Fasttext in terms of accuracy, they suffer in two main categories, complexity and speed. In the paper "Bag of Tricks for Efficient Text Classification" by Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov, they show multiple cases in which their Fasttext classifier is on par in terms of accuracy to other deep learning models. And while accuracy is virtually the same among both text classification solutions, Fasttext drastically outperforms deep learning solutions in terms of training speed. Oftentimes while a deep learning solution takes hours, a Fasttext one takes seconds. Fasttext is also a robust and maintained library that is fully implemented with python pip, so implementing it is extremely simple, whereas the complexity of creating a deep learning model for this problem would be beyond the scope of this research project.

IMPLEMENTATION

Data Source

For all machine learning problems getting mass amounts of data is a crucial step for accuracy. For supervised learning problems, oftentimes getting the data is not challenging; it is labeling the data that provides a hurdle. This project was no exception. I have come to understand that the best machine learning solutions are often ones in which mass amounts of data are easily obtained, often automatically, and then labeled. Keeping this in mind, the data for this project was obtained with the assistance of Dr. Collard. He was able to provide student-created comments from projects in his classes. This type of data was beneficial because the comments were created by students so there was a healthy mix of both good and bad comments.

However, this data source was not without drawbacks. The first of which was that since the data was all for assignments, the students were commenting on the same bodies of code. This meant that a lot of the data was nearly identical or default comments that were left by the professor himself. This presented a drawback because the comment classifier would pick up a word or phrase that is frequently used for a good or bad comment and might automatically mislabel it because of that word or phrase. In one perspective, this could be beneficial in this case because largely the comments that were written by the professor and thus were generally good comments. This means that this data could potentially be all labeled good and help train the system. However, to preserve data integrity it was decided to create a solution to this drawback rather than ignore it. The solution to this problem was quite simple, to get rid of duplicate comments.

Getting comments in this way also caused another drawback. Because the comments were assignments given by the professor, oftentimes, comments were instructional. This means

that the comments were instructions to tell the students how to do the assignment. The solution to this problem was not as simple and was more time consuming as well. It required the manual omission of this type of data.

Labeling

The main challenge, after gathering the data, was to obtain labels for the data. Fasttext does offer unsupervised learning options for text classification, but since the content of the comments will never feature whether it is a bad comment (unless if the comment were to literally say “this is a bad comment”), this method is not appropriate. With this in mind, there were two approaches utilized to obtain labels for the project.

The first method was to manually label data on a three-point scale. One being good, two being bad, and three being omitted comments. The comments labeled three were either commented-out code or comments that required more context to be labeled. This solution was not only short lived, but it was generally bad. The problem with this solution was that there was so much subjectivity to the “good” comment that inconsistency in data labeling was often rampant, creating many mislabeled comments. On top of that, getting enough labeled comment data was extremely time consuming.

In an attempt to streamline this process, a python script was created to automate this as much as possible. This tool was eventually developed into a full shell tool. The two main features of this shell tool that aided in comment labeling was the SortAndUnique and CommentLabeler tools. The SortAndUnique tool was created to solve the problem of data repetition as listed above and was not initially intended to solve this problem. This tool sorted the data and then ran a unique function that omits any repeating comments. The data was sorted to make it easier to run the unique function. It is easier to run the unique function on sorted data

because an unsorted unique function would need to check the whole list for every line thus making it an extremely costly O(n^2) algorithm. While the sorted unique algorithm utilizes quicksort, which is a O(nLogn) algorithm and then it would need a unique function step which would be O(m), so the final runtime of the algorithm would be O(nLogn * m). This tool made it easier to manually classify data because when the data was sorted like phrases were all grouped together. Meaning multiple blocks of good and bad comments were generally grouped together, making labeling conceptually easier to comprehend.

The other python script was the CommentLabeler. This was a tool developed for this project to label comments efficiently. This tool automatically prompts the user with comments to be labeled using the command line. This tool provides three main functions, error handling, state retention, and automatic prompting. The user defines the potential labels in a config file, and if the user does not enter a proper label, the tool does not label that comment and pre prompts the user with it; this is to prevent user error and preserve data integrity. Another function the tool provides is state retention; when the user is finished commenting, and they are not finished with the file, they are able to exit the tool, and it will automatically save the state in which the user was at so that they can resume where they left off. The last main feature provided is automatic prompting; the user is automatically prompted with each comment in the data set one after the other every time they label a comment. This tool is simple in concept, but it goes a long way.

To see why this initial solution was not optimal, it is necessary to observe the dataset. The dataset used for this project was 30,000 comments, and after sorting was 5,000 comments. Hypothetically, if it were to take someone an average of 30 seconds to discern a good or bad comment, to label all 5,000 comments, it would take that person 150,000 seconds or ~41 hours to completely label the data set. If that person were to attempt to label the full data set, it would

take them ~6x longer. So, with this information, this solution is bad mainly because it is not scalable. Good machine learning problems are ones that have easily scalable data and label solutions. The best machine learning problems are ones that have automatically scalable data and label solutions.

The main and final solution found in this project for data labeling was utilizing another portion of the data, the grade of the assignment itself. Instead of manually labeling the data on a three-point scale, we used a typical grading scale (A, B, C,D, F) with “A” being the highest and “F" being the lowest. The labels for the data would be from the assigned grades.

This solution was not without its drawbacks though, while it was a much more scalable solution, it also still required a certain amount of manual labeling in that the professor had to look up the grades and label those students’ code comments appropriately. This could be done semi automatically as opposed to the other solution, which was nearly completely manual. This solution provided ~2600 labeled comments as opposed to the manual solution, which provided ~500. This solution also has the problem that the grading scale does not necessarily determine whether or not the students' comments are “good” or “bad” based on the previously established definitions. This also raises another question, should the comment classification AI just label the comments “good” or “bad” or should it label it on some sort of spectrum? This project focused on a spectrum identical to that of the grading scale. The best comments were Labeled A, and the worst were labeled F using the semi-automatically generated data from the assignment.

RESULTS

As expected, the manually labeled comments produced poor results. At several stages of labeling the data, a fasttext model was run on manually labeled test data and it performed poorly. Since there was so little labeled data, and the data variation was so small (the data variation was small because all comments are labeled in order and the data is sorted), fasttext ~74% of the time chose the category that was used the most, in this case one. The accuracy of this was manually measured on 100 comments and was found to be 64%. However, this accuracy was largely due to most of the comments being good comments. If this was run on 50 bad comments and 50 good comments, the accuracy drops to 57%, and considering good and bad comments are 50-50 split, it is just slightly better than guessing.

The graded comments produced a much better result; however, with more data, the accuracy could be increased. To measure accuracy, 100 comments were taken from the test data set and manually labeled, 50 good comments and 50 bad comments were used. They were then run through Fasttext, and if they were labeled A or B, they were good, and if they were C, D, or F, they were bad. The accuracy of this was then measured with respect to how often these two matched. This produced a result of 71%, which was much better than expected.

CONCLUSION

This project presented a great opportunity to explore machine learning in a practical manner. It was an interesting way for me to explore my concentration interests within my major to be able to implement a practical solution that could be helpful. The project itself was very engaging. Despite the pitfalls of quantity and quality of data and methodology, I was able to arrive at worthwhile solutions that will help to create opportunities for future growth. The most important conclusion of this is that it is better to have a large amount of mediocre data than it is to have a small amount of good data for machine learning solutions, in particular Fasttext solutions. This is because it provides room for improvement which is the purpose of machine learning. The field of AI and machine learning is quite expansive, and the opportunity to develop cutting-edge technology in this area is only on the surface. I look forward to continuing to develop strategies within this realm.

FUTURE WORK

A great deal of future work could be done with respect to this project. Since it would be possible to skip the intermediary step of determining accuracy using A-B and automatically convert the labels to 1-2 based on the A B = 1 and C D and F = 2 scale, efficiency and potentially accuracy would be improved.

Obtaining more data would be a huge help to accuracy. As is generally accepted, the larger the sample size, the better the results. With this in mind, the more classification data that is available, the better the results will be. One possibility for obtaining more data in the labeled form would be to get data from multiple professors. Another possibility for gathering mass amounts of unlabeled data would be to create a scraper tool to scrape GitHub or another mass code base and extract the comments from there.

Testing out Fasttext unsupervised models would also be interesting to see the classifications Fasttext comes up with. This would especially be true if this method were used on the potential scraper tool, as mentioned earlier.

Lastly, a deep learning model could be used instead of Fasttext. It would be interesting to see how a traditional deep learning model would compare to Fasttext in terms of accuracy. In this way, the optimal process could be determined for future applications.

BIBLIOGRAPHY

Athiwaratkun, B., Wilson, A. G., & Anandkumar, A. (2018). Probabilistic fasttext for multi-sense word embeddings. *ArXiv:1806.02901 [Cs, Stat]*. http://arxiv.org/abs/1806.02901

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. *ArXiv:1607.01759 [Cs]*. http://arxiv.org/abs/1607.01759

Kallis, R., Di Sorbo, A., Canfora, G., & Panichella, S. (2019). Ticket tagger: Machine learning driven issue classification. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 406–409. https://doi.org/10.1109/ICSME.2019.00070

Pascarella, L., Bruntink, M., & Bacchelli, A. (2019). Classifying code comments in Java software systems. *Empirical Software Engineering*, *24*(3), 1499–1537. https://doi.org/10.1007/s10664-019-09694-w

Tenny, T. (1988). Program readability: Procedures versus comments. *IEEE Transactions on Software Engineering*, *14*(9), 1271–1279. https://doi.org/10.1109/32.6171

Tenny, Ted. (1985). Procedures and comments vs. The banker's algorithm. *ACM SIGCSE Bulletin*, *17*(3), 44–53. https://doi.org/10.1145/382208.382523

Woodfield SN, Dunsmore HE, Shen VY (1981) The effect of modularization and comments on program comprehension, pp 215–223

Yang, B., Liping, Z., & Fengrong, Z. (2019). A survey on research of code comment. *Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences - ICMSS 2019*, 45–51. https://doi.org/10.1145/3312662.3312710