CS846-001 Week 6 Reviews

Jun Qing Lim
University of Waterloo
jq3lim@uwaterloo.ca

1 Sentiment Polarity Detection for Software Development

1.1 Problem Being Solved

The paper presents a classifier (Senti4SD) to support sentiment analysis in software development artifacts with a high accuracy of sentiment analysis. It proposes this classifier because the current sentiment analysis tools are not suited to the current software development context and contains many misclassifications. The authors claim that the proposed classifier could outperform the previous state-of-the-art and help reduce sentiment polarity misclassifications with the use of machine learning algorithms and natural language processing.

1.2 New Idea Proposed

The authors trained the classifier with a gold standard of posts mined from Stack Overflow and well-balanced posts of positive, neutral and negative emotions. Features for emotion polarity classifier are then added with lexicon-based, keyword-based, semantics representations to address biases in off-the-shelf sentiment analysis tools. The model is then piloted, trained and tested, and the following findings were concluded:

- reduces misclassifications of positive, negative and neutral biases
- outperforms SentiStrength on technical texts (higher performance)

The emotion polarity dataset built upon the Stack Overflow corpus is then made available for the public while serving as a valuable resource for future research in software development artifacts.

1.3 Positive Points

Firstly, the authors did not just propose a classifier with a set of evidences and arguments but also tested and verified the accuracy of the model, this serves as an important point of credibility for readers to cite the work of accuracy and efficiency improvement. Secondly, the paper introduced a state-of-the-art approach of combining emotion polarity with NLP and ML processing, which differs than the previous sentiment analysis tools (i.e., authors introduced a novel idea). Thirdly, the paper not just gathered a set of data mined from an online forum (Stack Overflow) but also made this dataset available to the public, this serves as one of the building blocks and a valuable resource for both developers and researchers to conduct study in this area.

1.4 Negative Points

Firstly, the authors only utilized Stack Overflow corpus to build a sentiment polarity classifier; as such, the model built may not be applicable (or easily generalizable) to other domains or applications. Secondly, the performance and accuracy of the classifier is only evaluated against the same dataset (as 70% training and 30% testing), as such the testing outcome may not be fully precise and further evaluation on other datasets is needed.

1.5 Future Work

An extension would be conduct further testings on other domains to determine its accuracy and applicability. Apart from that, we could also expand the research to incorporate "non-direct speech" - statements that were commented but imply alternative meanings (irony posts), this could help raise the accuracy further for incorrect classification. We could also expand the research to handle a set of language corpus tao tackle sentiment analysis for different languages (grammar, structure), improving its generalizability of the sentiment polarity detection system.

1.6 Rating

4 out of 5. Although the paper is extensive, but it was detailedly written and proposed a new approach to sentiment analysis, which is quite a research milestone for this area.

1.7 Discussion Points

- Are there any existing tools of sentiment analysis which have similar functionalities?
- What are some common natural biases that could be induced by any classifier?

2 Listening to Programmers — Taxonomies and Characteristics of Comments in Operating System Code

2.1 Problem Being Solved

The paper presents the outcome of a comprehensive study on comments of operating systems code to understand their characteristics and purposes. It aims to address the issue of improving software reliability and innovations by listening to the thoughts expressed by programmers through code comments, which could provide guidance for development of new techniques. Through this study, the authors hope bridge the gap between programmers' actual needs and tool usability to better improve code and project maintainability of software systems and productivity.

2.2 New Idea Proposed

The authors gathered 1050 comments that are randomly sampled from latest versions of operating systems: Linux, FreeBSD and OpenSolaris and studied the comments in its source code. The comments were then classified into four taxonomies of **WHAT** comment is that, **WHO** wrote the comment, **WHERE** are the comments and **WHEN** the comments were written, and subsequently obtained the following findings:

1. On **WHAT**: comments that were written usually compose of the following areas:
   - comments that describe the usage and meaning of variables
   - comments that describe code relationship between variables and/or functions
   - comments that describe code evolution aspects such as TODOs, deprecated code or cloned code
   - comments related to low-level ideas such as locks and synchronization
2. On **WHO**: comments are written by wide-variety of people not just general programmers but also well-known experts.
3. On **WHERE**: certain types of OS (such as OpenSolaris) were commented less than its other counterparts (such as Linux) as they reflect different programming needs and modern requirements
4. On **WHEN**: comments were written as early as when the project was first started and these comments were evolved over time. Some modifications were made to the comments to suit current programming needs while some comments remained unchanged.

2.3 Positive Points

Firstly, the paper proposed a set of taxonomy of comments, which provides readers an initial understanding of how the research was conducted and makes it easy for readers to understand the various types of comments found in the operating system code. Secondly, the paper provided deep explanations and insights over the characteristics of comments (and its realms), such as comments distributions (its statistics), code entities, purposes, etc.; this helps to serve both researchers and programmers to better understand software comments and its impact in the software development field. Thirdly, the paper is relevent to the current state of software as it involved code comments up to recent years, providing useful ideas towards the software community.

2.4 Negative Points

Firstly, the research only focused on operating system code and not other software systems, this may limit the generalizability of the findings. Secondly, the paper lacks in-depth explanations of its methodology (such as how are the percentages of comments calculated/computed - i.e., statistical explanations).

2.5 Future Work

As illustrated as part of its negative points, one extension that we could make is extend the study to other software systems source code to to better understand the overlapping insights and its differences. Furthermore, one other extension that we could do is to extend the study of comments to code reviews (instead of only source-code based), this allows us to develop new understanding on how comments can be used to improve the process of code reviews whilst improving the overall programmers' work efficiency.

2.6 Rating

2 out of 5. In general, the paper serve very little to no impact towards software industry (i.e., it lacks the deep explanations on how studying comments improve the software industry - though I may be bias). A higher rating would be provided if the paper was written with slightly deeper explanations.

2.7 Discussion Points

- How did the comments evolve over time and so how did the programmers' need evolved?
- Is there any tool which allows us to automate the process of parsing comments to highlight important points in the source code?
- How useful are comments towards the actual software community?