Eye Tracking in Software Engineering

Xiaozhe Yao

October 27, 2019

Eye tracking, the technique for measuring the movements of gaze, is promising in the research of visual systems, such as human-computer interaction, psychology etc. It has now been widely used in software engineering. As stated in (Sharafi, Soh, & Guéhéneuc, 2015), between 1990 to 2014, there are 36 papers related to the use of eye-tracking in software engineering.

In (Rodeghero, McMillan, McBurney, Bosch, & D'Mello, 2014), they proposed an eye-tracking study of 10 professional Java programmers in which they read some Java methods and wrote summaries. Afterwards they analysed the eye movements and gaze fixations of the programmers to identify where the programmers focus on when reviewing and writing summaries. Then they concluded that the VSM tf/idf approach roughly approximates the list of keywords when programmers read, and they found the priority when programmers read, i.e. method signature > invocation keywords > control flow keywords. Based on the findings, they proposed an approach for extracting keywords by modifying the weights of different keywords according to the priority.

(Rodeghero, McMillan, McBurney, Bosch, & D'Mello, 2014) provide an example of using eye-tracking to source code summarization. The approach can also be used in similar fields, such as automated code complexity analysis by recording the gaze fixation time (*Hypothesis*: The longer programmers stay in a method, the more complex this method would be). Indeed, some researchers use eye-tracking to measure the complexity of websites, such as (Wang, Yang, Liu, Cao, & Ma, 2014), but not source code complexity right now. Meanwhile, The comprehensive experiments is convincing, but I would recommend explorer deeper with the data. For example, as stated by the author, the Pearson correlation is negative for seven of 53 methods, then what's the reason behind for the negative correlation? Pearson correlation varies between *-0.28* and *0.94*, what is the reason for the variation? These questions makes sense because the code traits may heavily influence how programmers read it, and can be used to adjust the weights accordingly. The dynamic weight adjust might have a positive impact on the performance of the tool.

Prior to (Berkovsky et al., 2019), traditional methods rely on self-reports and self-perceptions, which might be unreliable or biased due to privacy concerns, misunderstandings, etc. To avoid these issues, they proposed a framework for detection personality traits by analysing the persons' response to external stimuli. They first use external stimuli to trigger physiological responses, and then capture the physiological responses from participants. Afterwards they use the general pipeline of machine learning to handle the data, i.e. data processing, feature extraction and then supervised learning. It is a classification task and the trained classifier can be used to predict the trait values from the eye data and determine the trait class label for a new subject.

Compared with (Rodeghero, McMillan, McBurney, Bosch, & D'Mello, 2014), (Berkovsky et al., 2019) not only assess the performance of different classifiers, but also try to explain why Naive Bayes outperforms other classifiers. The assessment across different classifiers and different stimuli are thorough, well-explained and convincing. It is clear that video better performed than images regarding to the trait detection task, but I am also curious about the effect of different duration of video since the video used in their study is much shorter than previous study, as claimed by the authors (*Hypothesis*: It might looks like Gauss distribution). Besides, as stated by the author, they use "off-the-shelf" classification algorithms, and they explain NB's performance is because its assumption that the features are i.i.d, there are some other methods that could contribute to the improvements of accuracy, such as MultiNomial Naive Bayes, Random Forests, Bayes Network etc. With a looser assumption, it might outperform Naive Bayes in some cases. It is hard to say if it will work better but it is anyway worth a try.

(Rätsch, n.d.) is a good resource for learning basic machine learning, which introduced supervised classification, KNN, LDA, Decision Tree and Neural Networks, as well as SVM and Boosting.

References

Berkovsky, S., Taib, R., Koprinska, I., Wang, E., Zeng, Y., Li, J., & Kleitman, S. (2019). Detecting Personality Traits Using Eye-Tracking Data. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*. ACM Press. https://doi.org/10.1145/3290605.3300451

Rodeghero, P., McMillan, C., McBurney, P. W., Bosch, N., & D'Mello, S. (2014). Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. ACM Press. https://doi.org/10.1145/2568225.2568247

Rätsch, G. A Brief Introduction into Machine Learning.

Sharafi, Z., Soh, Z., & Guéhéneuc, Y.-G. (2015). A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, *67*, 79–107. https://doi.org/10.1016/j.infsof.2015.06.008

Wang, Q., Yang, S., Liu, M., Cao, Z., & Ma, Q. (2014). An eye-tracking study of website complexity from cognitive load perspective. *Decision Support Systems*, *62*, 1–10. https://doi.org/10.1016/j.dss.2014.02.007