Summary 3: Improving Language Models with `CheckList`

Sam Showalter
University of California, Irvine (showalte)
showalte@uci.edu

1 Content Summary

This is a reivew of (Ribeiro et al., 2020) which discusses a novel method of evaluating NLP models that provides a more granular review of their capabilities. The system, called `CheckList`, takes inspiration from behavioral testing frameworks common in software engineering. The authors feel that held-out test set accuracy is insufficient to evaluate LMs. Given the complexity of these systems and the set of capabilities the model needs to be holistically performant, `CheckList` appeared to be a logical and necessary extension.

`CheckList` is defined in terms of a matrix, where rows specify capabilities and columns, test types. In general, it is intended as an easy-to-use yet powerful software development tool to generate and evaluate series of *behavioral* tests. Many LMs, despite SOTA performance, have been noted to fail on elementary language and bias tests. This leads to two implications: First, LMs may find shortcuts to achieve performance instead of learning general heuristics. Second, even in well-tuned models, evaluation is lacking, leading to unnoticed bugs and biases.

To show `CheckList` is a general tool, the authors evaluate several NLP tasks including reading comprehension and sentiment analysis. They discover that even models with superhuman performance possess (sometimes egregious) failures in language understanding. To verify that `CheckList` is a generally useful tool, they worked with Microsoft's sentiment analysis system and team, where several bugs were found and developers stated `CheckList` valuable. In another survey, those with little experience were able to leverage `CheckList` to find more bugs and generate more test cases in SOTA NLP models, in part because of its templated, cloze-style test cases.

The authors close by noting that `CheckList` does not serve to replace the many additional robustness and capability tests for LMs, but complement them. They feel the tool effectively centralizes the evaluation of LMs holistically and facilitates better debugging as well as more granular model evaluation and comparison.

2 Analysis

This is one of the most exciting and compelling papers I have read in NLP. The authors are very clear about their research motivation as well as the intended application of their solution. Moreover, they conducted case studies where professionals utilize `CheckList`, something I have seen few authors do with their implementations. Their sections flowed smoothly and their proposal has been demonstrated to be easy to use as well as valuable for professionals and relative amateurs alike.

The contribution of `CheckList` fills a notable gap in NLP evaluation. In the last decade the capabilities of LMs have improved rapidly, leaving a strong need for equally granular evaluation. What's more, `CheckList` goes further to facilitate a smooth transition from evaluation to bug detection and patching. In turn, I see their contribution as an incremental insight, but a novel contribution in the form of their software solution. Empirically, the baseline results on bug findings in SOTA models were compelling. The automatic generation of test cases suppresses the risk of cherry-picked examples, though perhaps the impact of the lexicon-generating model (RoBERTa) should be taken into account. Qualitatively, a larger sample for the amateur case study would have been ideal.

However, my biggest criticism of this implementation is that the author's did not speak at length about the limitations of their solution. As with any software tool, there are fundamental limitations to what it can do. I imagine `CheckList` is no different, and there are probably a family of behaviors which it cannot evaluate well. Similarly, `CheckList` is also limited by the manner in which its test cases are generated. The authors demonstrated a preference for templated and generated solutions, but did not qualify what impact this practice may have on overall evaluation results. Robustness tests on `CheckList` by varying test-case generation methods/models seems like a promising area of future work. This can be done easily, because the authors provided easily understandable code for `CheckList` on Github.

References

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of nlp models with checklist. In *Association for Computational Linguistics (ACL)*.