CS 4650 Project Proposal

Recracking the Coding Interview

Rohit Das
rdas49

Vinnie Khanna
vkhanna32

Raj Srivastava
rsrivastava60

1 Motivation

Our problem is defining the best conceivable runtime (BCR) of software engineering interview questions. Essentially, coding problems such as the ones found on LeetCode and Hackerrank are commonly used in software developer interviews, and in order to create the most efficient solution possible, interviewees try to find the BCR of the problem. There may be words present in problem statements of LeetCode problems that can help determine the BCR. An NLP model can possibly provide guidance on what keywords and key phrases indicate different BCRs. This problem can also be extended into using problem statements to predict either the data structures and algorithms required to solve a LeetCode problem or the difficulty of the problem. Overall, this problem has a lot of potential to be useful to software engineer interviewees as we use NLP to investigate how much info about a problem can be gleaned from the problem statement.

Other researchers have worked on similar problems. Zhou and Tao (2020) used multi-task BERT and other competing models in order to predict the difficulty of LeetCode problems. Kolak (2020) uses DeepWalk, an unsupervised deep learning technique, to predict how efficient code snippets are in order to better tell developers what they should optimize. While both of these papers relate to our problem, neither explicitly looks at the idea of relating problem statements to Big-O runtimes. However, the results from Zhou and Tao (2020) do show promise for our problem by demonstrating that LeetCode problem statements holds enough info to be useful in text classification. Furthermore, the results of Kolak (2020) show potential in using unsupervised methods on code snippets to predict code efficiency. While we will be focusing on problem statements, it is highly likely that problem statements and the code snippets used to solve the problems are correlated.

2 Goals

Our main goal of this project is to derive insights into the coding interview process that have been unforeseen. Specifically, we aim to:
1) Create a classifier to identify optimal time complexity for solution given problem description
2) Create a classifier to identify difficulty or acceptance rate of problems given problem description
3) Cluster types of problems to identify trends
4) Analyze the results to generate insights on preparing for coding interviews
5) **If possible** automatically generate new coding problems given difficulty level, target time complexity, and problem category (more information in Section 3.2).

3 Plan

The project will take place in multiple stages:

1) Data collection with webscraping from leetcode.com (2 weeks)
2) Modeling (4 weeks)
3) Analysis (2 weeks)
4) Final paper (1 week)

All stages will be split fairly equally among group members, despite some being more comfortable with certain tasks than others. Web scraping will mostly be done independently, as it will be fairly coding intensive but does not require much planning. The second and third stages will be much more group-oriented, as we will need to discuss in great detail which models we may or may not combine before our final classification.

3.1 Web Scraping

From our research, there are no existing datasets mapping leetcode problem descriptions to identifying characteristics such as runtime, difficulty level, or problem approach. As a result, we will be aggregating our own data from existing web sources, namely LeetCode and HackerRank. Scraping will be done with Python using common libraries such as Beautiful Soup and Urllib. We will construct a dataset with problem descriptions and difficulty levels as features from the websites' respective 'problems' pages, and we will scrape runtime as labels from either the 'solution' section - if available - or 'discussion' section. As a stretch goal, we would also extract the problem approach from keywords appearing in the 'discussion' section. From our initial analysis, we will have roughly 1500 problems that are publicly available on LeetCode alone, which should provide plenty of training samples for our purposes.

3.2 Modeling

The modeling will aim to answer questions 1-3. We will each attack one of the questions, likely using a BERT classifier combined with another layer to generate the outputs. For questions 1 and 2, this final layer will be a classifier. For question 3, this final layer will be a clustering algorithm, and will require some further analysis to determine cluster cohesiveness evaluate whether the clustering algorithm performed well or not.
Finally, after this work is done, if time permits and if our other experiments generate good results, we will move onto question 5. For this, we will likely have to use a model like GPT-3 and our results from all our previous questions to help create this final model. We suspect that we will not have the best results for this model given our small data size, but it will be fun to see the results nonetheless. Google Colab and personal computers should be sufficient for the task at hand.

3.3 Analysis

By setting up our infrastructure and attaining some initial results in advance of the end of the semester, we will have adequate time to review and make any necessary adjustments. This includes any hyperparameter tuning and/or further research to properly explain our findings. As with modeling, this will require more group meetings and discussion over actual individual coding.

References

Sophia Kolak. 2020. Detecting performance patterns with deep learning. In *Companion Proceedings of the 2020 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, SPLASH Companion 2020, page 19–21, New York, NY, USA. Association for Computing Machinery.

Ya Zhou and Can Tao. 2020. Multi-task bert for problem difficulty prediction. In *2020 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pages 213–216. IEEE.