Hackathon – IML

Task 2 – To Each his Code

Source code identification is a known and well-researched problem. We started by looking for metrics for this problem and encountered an interesting method using n-grams (contiguous sequence of n letter)[1].

This method learns by constructing a list of the n-grams in each project sorted by their frequency. We keep only the L most frequent for each project i (from now denoted as P_i). This data is saved in a file.

The prediciton is done by decomposing the given code to its n-grams and computing the intersection of those n-grams with each P_i. The project with the largest intersection will be chosen as the prediciton.

The initial implementation of this method is quite simple, but we had to choose both n and L and decide how to break ties.

We chose n and L first by what was recommneded in the papers. Unfourtunately, they based their research on bigger test codes.

So, we tried a few of those hyperparameters and managed to deacrease the error-rate.

We found that most of the prediction erros are caused by ties (where we just choose the project with lowest index). Those ties are a result of many reasons:

1. shared libraries (sush as Apache), where many projects use and therefore use the same commands and comments.
2. Short single lines like "})" which ends code blocks and is shared with most programming languages.
3. n-grams that don't show up in any P_i (all intersections are of size 0).

One of the solutions to resolve those ties was to use another n as a tie breaker.

We chose to first test with $n_1 = 14$, which has a very good success rate, but sometimes results with a tie that hurts it. The tie was caused by an empty intersection with all P_i because it might be a too long sequence. So we took the tied projects and then checked with $n_2 = 6$ and chose accordingly. Because of the smaller size, it had a larger intersection size.

For choosing L (the number of n-grams in each P_i), we tried a a large range of numbers: from 20 to ∞ (keeping all n-grams). Ideally, ∞ would be best, but it takes a lot of time and space. We found $L = 50,000$ to work fast and well.

We tested the algorithm by dividing the given code into 70% training set, 15% validation set and 15% test set.

We managed to reach an error-rate of 15~20% (a random predictor has an error rate of 85%).

The main cause for errors were single lines which are shared by all programming languages, and therefore can't be propely identified. The same for file description in comments.

[1] Frantzeskou, Georgia, et al. "Effective identification of source code authors using byte-level information." Proceedings of the 28th international conference on Software engineering. ACM, 2006.