NegSelReport

Name: Theo Farrell User-name: nchw73

Two-letter code for your chosen negative selection algorithm: VD

Note – DR = detection rate (%), FAR = false alarm rate (%)

For parameter tuning, I used a grid search approach to explore various parameter combinations. This involved exploring wide intervals for each parameter at first, and then systematically narrowing them to investigate promising results. Within every interval, I sampled 10 evenly spaced values. Each parameter combination was repeated 3 times, and the results were averaged to account for randomness. For example, my first grid search checked 10^4 combinations of 10 equally spaced values for each parameter within the following intervals:

- c0 and c1 – [0.75, 0.95]. Note that the possible values for c0 and c1 were the same, but every c0 was tried against every c1 so we get 10^2 combinations from this interval.
- Threshold – [0.01, 0.1]
- N – [100, 1000]

This was surprisingly quick, because most **(c0, c1)** pairs where one was below 0.9 generated very low detector counts. If the threshold was small enough (less than 0.04) then there was always at least 1 detector produced in my tests for any (c0, c1) pair. As such, the following results were taken when threshold = 0.02 and N = 100. In one set with only one detector, DR was 3.93% with a FAR of 0.03%. The general trend was that a (c0, c1) pairs closer to (1, 1) produced more detectors and gave better results. E.g. (0.975, 0.975) produced 39 detectors. This wasn't surprising, as c0 and c1 are meant to represent coverage. Somewhat surprising however, was that c1 was much more significant than c0 for the output quality. An example of a typical (high c0, low c1) pair is (0.975, 0.85) which produced 8 detectors (with low DR and FAR). A typical (low c0, high c1) pair however looks like (0.825, 0.975) which gave 36 detectors with DR = 22.32% which is much better! Taking this observation further, I tried (0.775, 0.9999) which at one point gave 434 detectors and DR = 62.83%! The runtime was also significantly longer since much more detectors were generated.

Despite this observation, (high c0, high c1) pairs continued to give the most *consistently* high-quality results (and longer runtimes). Narrowing the interval for possible (c0, c1) pairs and repeating the grid search led me to finally choose c0=c1=0.9999. Beyond this, set generation consistently took longer than 13 seconds which is too long. Assume from now that c0=c1=0.9999.

As for **thresholds** on [0.005, 0.05], DR was > 70% on [0.005, 0.035) and FAR was < 10% on [0.025, 0.05]. This suggested [0.025, 0.035) as the optimal range. Beyond 0.05, there were a significant number of empty detector sets produced. By repeatedly narrowing this interval, I settled on 0.027 as my threshold, which gave DR = 78.38% and FAR = 6.07%.

For **intended number N of detectors**, a larger value resulted in longer runtimes but generally enabled higher quality detector sets. The latter part is due to the fact that the highest quality sets required small thresholds (as discussed above), which in turn required a larger N. Given my threshold = 0.027, I set N = 600 since it took longer than 13 seconds to generate more than this, at which point the runtime is too long.

Enhancement - When one detector is placed next to another, they touch at most once, which is only when the distance between them = threshold. In this case, there is a gap of potentially non-self between the curves of the detectors, where no new detector can fit. Noticing this, I decided to slightly extend the radius of detectors at the moment they are added to the detector set to cover this gap. I added an **alpha** scaling factor, which is multiplied by the threshold and added to the detector radius. This significantly increased detection rate while FAR remained tolerable (<10%). For c0=c1=0.9999, threshold=0.27, N=600: when alpha = 0 (i.e. the vanilla version) I averaged DR = 73.77 and FAR = 4.4 across 10 detector sets, whereas when alpha = 0.36 I averaged DR = 81.05 and FAR = 8.95, as well as ~70 less detectors. Alpha = 0.36 was the best value I found using a similar iterative approach to finding the threshold.