Apriori HW Write-Up

My chosen threshold for the report was 600: with that I got about 130 different patterns — I chose this number because the amount of patterns was more easily readable for me, while still allowing a lot of multi-key patterns to slip in. Any number higher than this would begin to significantly compromise the amount of knowledge we can actually get from generating this pattern set.

We can see a couple interesting things present in the existing set: firstly, we can have a lot of confidence that these are actually tweets about the flu, as “flu” is the foremost keyword with 26,000 support. Furthermore, we can conclude that getting a flu shot is a very ‘tweetable’ event — the patterns “flu get shot”, “got flu shot”, “getting flu shot”, and other variations, have an immense amount of support and fill the top spots of the result file. We can also see some side-effect reporting happen: “sore” is occurring at 675 support, with nearly all of it also co-occurring with “shot” or “flu shot” (both at 636). We can use this number to estimate that “sore flu shot” (636) / “flu shot” (23000) results in about 2.7%. This number could be an accurate estimate of side-effect prevalence, if only Twitter reports weren’t subject to self-selection bias.

The program I have written uses a number of algorithmic and data-representation improvements to optimize runtime. The major one among these aims to replace the traditional pattern of “searching the entire dataset to determine the support of each pattern” by instead storing references to each supporting entry — in this way, when trying to combine two patterns to create a large one, we can take the inclusion of their supporting sets. I created a custom inclusion algorithm designed to take advantage of the fact that supporting entry ids were assigned in sorted order, which sped up that runtime from O(n^2) to just O(n). I used Google’s recent Swiss-Table model of the common HashMap to allow multi-threading of some of the iteration. Summed up, my program takes 6.9 seconds to find threshold 2 on the full dataset provided, and the attached output.txt took .17 seconds to generate at threshold 600.

This assignment has taught me a couple things. I got a lot of insight (as was my intention) into building programs with Rust, especially things I never touched before like creating my own data structures, thread safety, and lots of new syntax. From an NLP side, I realized that performing this operation on raw keywords like we do will actually lead to missing potentially significant knowledge — a better solution would be to perform *stemming* and *lemmatization*, which would not only reduce the amount of unique patterns we have, but give us a better picture of what these tweets are actually about.