Lab 3

25948127
UC Berkeley

October 17, 2017

Introduction

This lab assignment serves as a continuation of the linguistic data lab, and focuses on computation speed and efficiency. The logic here is that the first lab introduced clustering and machine learning methods for dimensionality reduction and analysis. These methods are inherently computationally intensive. When confronted with this sort of problem, common approaches include unlocking more of a local machine's core processing capabilities, offloading the job to a remote server, and writing code in C++. In this report, I use these tools to improve the clustering of the binarized response data from the last lab, and report my results.

Concretely, this report is divided into the following sections:

1. Parallelize the k-means clustering algorithm and run it on the SCF remote server.
2. Write C++ code to calculate a similarity measure.
3. Reproduce Figure 3 from the Ben-Hur (2001) paper.

Parallelize k-means

The first step in this assignment was parallelizing the k-means clustering algorithm, and then running it on the SCF remote server. My code is available in the accompanying "R/" folder in the project files. The basic process involved the following steps:

1. Remove the non-quantitative/non-answer variables in the "lingData" dataset.
2. Write a wrapper function that calculates and returns a kmeans cluster assignment that could be passed into a later function and for loop.
3. Write a function that calculates the Jaccard similarity between two dataframes. Other options included correlation and matching, and it would be easy to substitute these measures in later on if I chose to.
4. Write a function that takes two subsamples from the data, clusters them according to a pre-specified number of k-clusters, and then apply the Jaccard similarity function to the two subsamples.
5. Write a for loop that simulated this process 100 times for k-clusters = [2:10] (the numbers tested in the Ben-Hur paper)
6. Initialize extra CPUs, and then place the for loop from step 5 into a "%dopar%" wrapping so that the above steps are run in parallel

C++ Function

The next task is to write a C++ function that computes the similarity between two subsamples. The underlying reason to do something like this is because C++ functions run faster than R ones (because R is built on top of C). So, I wrote the equivalent Jaccard similarity function in C++, and then sourced it into the R environment. After doing this, I microbenchmarked the two measures and got the following results:

	min	lq	mean	median	uq	max	neval	cld
R	2564.7365	2649.136	2813.9280	2680.6053	2699.9675	4792.3081	100	b
C++	289.1152	299.165	305.4503	302.7028	306.1963	346.1311	100	a

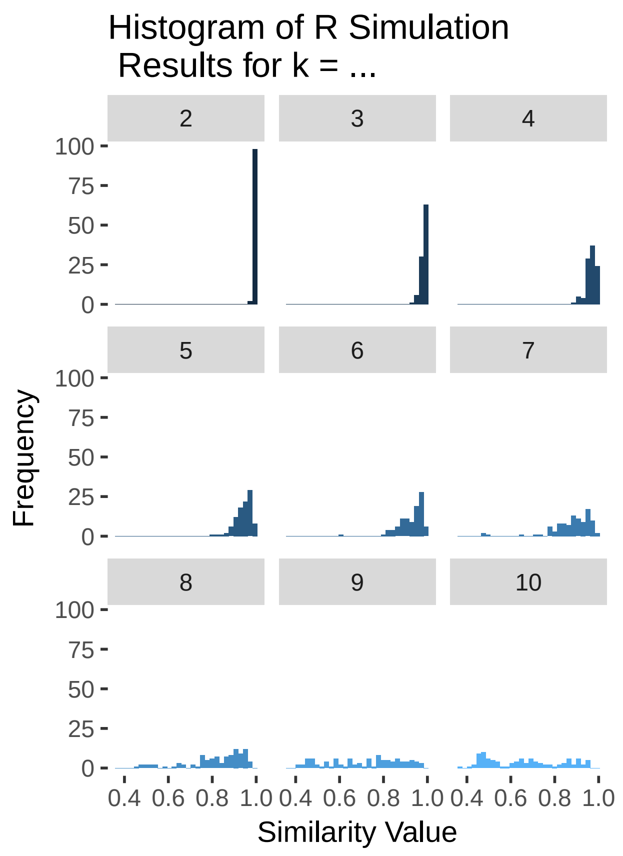
The C++ function seems to improve the speed by a factor of about 9. For something as large as the linguistic dataset, this is a boon for analysis.

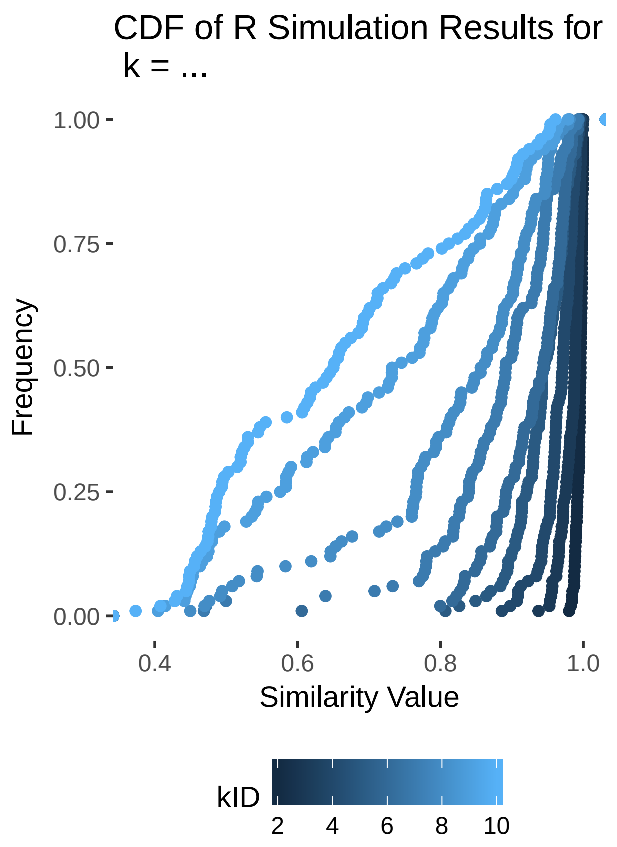
Reproduce Figure 3

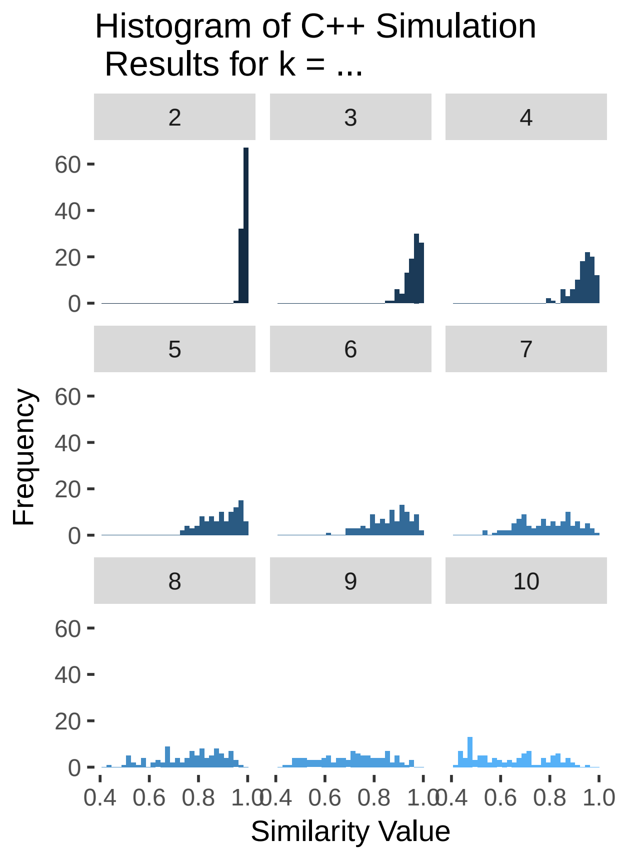
The final task was to reproduce Figure 3 from the Ben-Hur paper. In the paper, Ben-Hur et. al. provide a series of histograms and CDFs to illustrate how they made their choice of k. They selected a .8 sub-fraction (m) of each sub-sample, and performed 100 simulations (n) to generate these plots. I struggled to consistently achieve a $m = .8$, so I chose $m = .65$ instead, but still used the same n. Basically the process was:

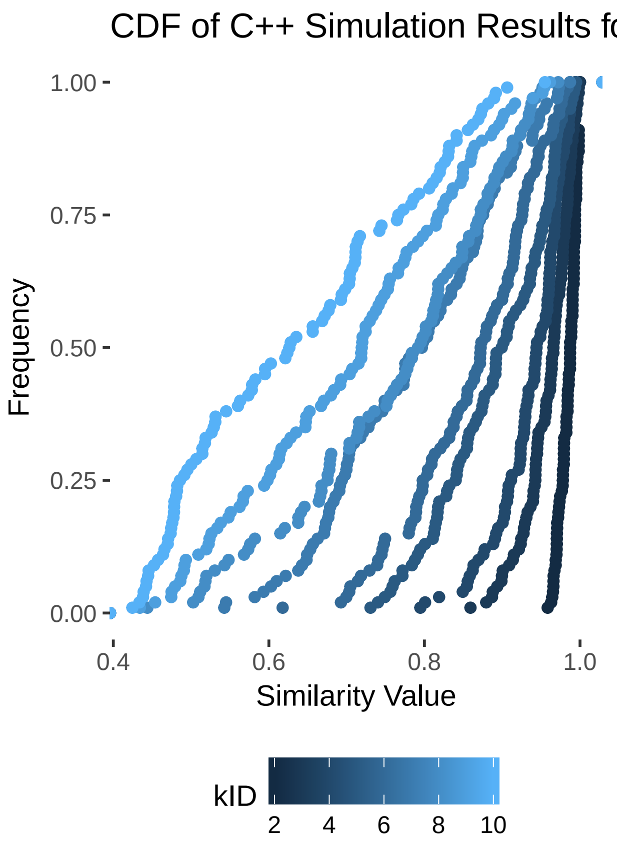
1. Initialize the clusters
2. Initialize an empty list that will contain dataframes for each k
3. Run the parallel for loop
4. After the loop ends, row bind all of the dataframes into a single data frame
5. Save a csv for the dataframe
6. Plot histograms and CDFs, grouping by the choice of k

I reproduce these plots below:









Above, I plot histograms for k = [2:10], and the CDFs next to them. One of the critiques I have of the original paper is that in the histogram plot, they change the scales of each successive histogram, which distorts the shape of the data and makes it difficult to compare across clusters. I use the same scales here, which better illustrates the tradeoffs between frequency and similarity. The CDFs do a much better job illustrating this, and I would probably choose k=6 based on the large gap between k=6 and k=7 (.8 vs. .6).

Conclusion

Overall, this lab illustrated the utility of making use of computational tools. Parallelizing the R code substantially improved the speed of the Jaccard matrix computation, which makes sense given the iterative nature of analysis (testing out k-means sequentially). Kicking the similarity calculation to C++ was similarly helpful as it was 10x faster than the R version.