Lab 3 - Parallelizing k-means Stat 215A, Fall 2017

Hector Roux de Bézieux

October 23, 2017

1 Similarity of clustering

First, we implement the computation of the similarity matrix in. We compute the *Jaccard coefficient* for the two labeling from kmeans. The *Jaccard coefficient* is $\frac{N_{11}}{N_{11}+N_{10}+N_{01}}$. The algorithm is efficient in memory usage since we never store any similarity matrix . We only compute the similarities on the spot and store the result in the summary statistics N_{11}, N_{01}, N_{10}. Memory usage is in O(n) instead of being in O(n^2). In R, this is computationally slower than matrix product since R is optimized for such operations, but the reduced memory usage makes up for it. We can see that, for 9000 samples, the algorithm run in 1s (see the table below).

We can also code the same algorithm in C++ and compare the relative speed. The similarity algorithm in C++ is stored in the **extra/** folder, in the **Similarity.cpp** file. We select 20% of the data (around 9000 samples) twice independently, run k-means on each set with $k = 2$ and compare the two labeling with the two similarity algorithms. We verify that both give the same result. Then, we use the *microbenchmark* package. Both algorithm are ran 100 times and we compare the time it takes to run them both.

```
## Unit: milliseconds
##                            expr         min          lq        mean
##  similarity_cpp(L1_com, L2_com)    1.818082    1.828008    1.996282
##    similarity_R(L1_com, L2_com) 1514.735300 1582.496765 1634.459596
##      median          uq         max neval
##     1.86099    2.074798    3.237957   100
##  1606.36964 1650.118171 2420.142180   100
```

The R algorithm run takes roughly 10^3 more time than the C++ one.

2 Parallel Computing

First we can code the inner loop. For given K, N and m, we sample twice a fraction m of the data set, run the k-mean algorithm with k clusters, compute the similarity score and repeat those steps N times. All that is done in a function called **SubSample** that takes as input k, N, m and the data set X and returns a $N \times 1$ vector of scores.

Then we loop over the number of clusters, k, from 2 to 10. This is the part that we parallelize. The code is present in the .Rnw file with the option *eval = FALSE* which means it is not run when the pdf is compiled. The real code that was use to produce the latter output is in the **R/** folder.

3 Results

First we can plot the density kernels for all the similarity scores of given number of clusters. This is Fig 1. A first comment is that even for $k = 2$, the maximum score, 0.29, is quite low. This probably mean that our clustering is not very strong. However, we can already see a clear cut between $k = 2$ and $k = 3$, between $k = 3$ and $k = 4$ and between $k = 4$ and the rest. This would lead to choosing either $k = 2$ or $k = 3$. With only that plot, $k = 3$ might be a more reasonable choice since the score is really consistent across all subsamplings.

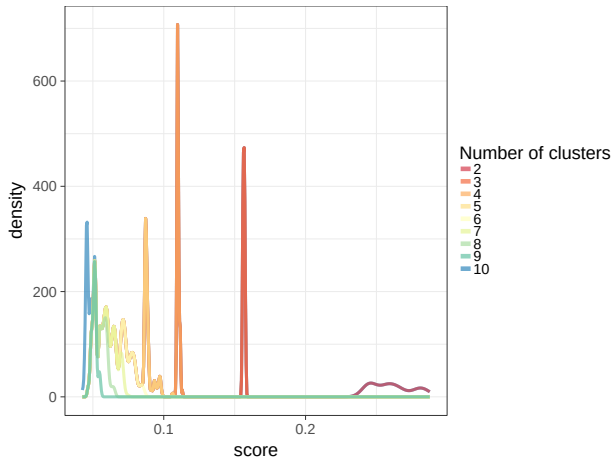

Figure 1: Kernel density plots of the similarity score over 100 subsamplings, for various number of clusters

We can also reproduce the figure 3 of the Ben Hur paper:

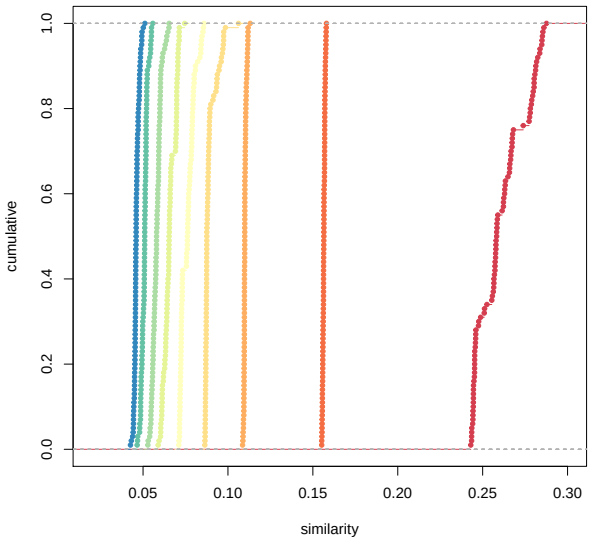

Figure 2: of the cumulative distributions for increasing values of k

As advised in Ben-Hur et al. [2001]., we want to select a cluster with very stable and very high similarity score. For $k = 3$, the score is very constant. We can imagine that, each type, a specific subset of points are correctly clustered while the rest of the points are not. So a small portion of the dataset clusters in 3 groups

while the rest doesn't cluster at all. Therfore, selecting $k = 3$ could make sense.

However, suppose we have k clusters assigned at random. For n large enough, we have $N_{11} = n \times \mathrm{P}(2 \text{ points are assigned the same cluster twice}) = \frac{n}{k^2}$. Likewise, $N_{01} = \frac{n(k-1)}{k^2} = N_{10}$. So the Jaccard Coefficent is $\frac{1}{2k-1}$. Therefore, when assigning at random, the Jaccard Coefficent has an expected value of 0.2 when $k = 3$. This is higher than what we have here!! However, since the similarity score is very stable, we can assume that the clusters are partly not assigned at random but given that the score is lower than its expected value for random assignments, the clustering is probably quite poor.