Lab 3 - Linguistic Survey
Stat 215A, Fall 2015

Emin Arakelian, Fadi Kfoury, Kuan-cheng Lai, Lei Kang

October 8, 2015

1 Lab Introduction

In this lab k-means will be applied on the binary linguistic data set that has been encountered in earlier labs. The purpose of this lab is to evaluate the stability of different clustering structures generated with different k-clusters. To assess stability three metrics will be utilized (Ben-Hur) :

1. Correlation Metric
2. Matching Coefficient
3. Jaccard Coefficient

In what follows we define the basic notations for computing those measures.

Let $\mathcal{L}$ be the the labels vector of a dataset X clustered into k - clusters $\{S_1, S_2, ...S_k\}$. We define the following pairwise $n \times n$ similarity measure matrix C , with:

$$C_{ij} = \begin{cases} 1 & \text{if } x_i \text{and } x_j \text{ belong to the same cluster with } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Such that $x_i$and y_i belong to $\mathcal{L}$.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the labeling vectors of two different clustering structures. We define SM as the similarity score between $\mathcal{L}_1$ and $\mathcal{L}_2$.

1.1 The Correlation Metric Similarity Score:

The correlation metric similarity score is given by:

$$SM_{\text{correlation}} = \frac{\mathcal{L}_1.\mathcal{L}_2}{\sqrt{(\mathcal{L}_1.\mathcal{L}_1)(\mathcal{L}_2.\mathcal{L}_2)}} \tag{2}$$

1.2 The Matching Coefficient Similarity Score:

The correlation metric similarity score is given by:

$$SM_{\text{matching}} = \frac{N_{11} + N_{00}}{N_{11} + N_{00} + N_{10} + N_{01}} \tag{3}$$

where, N_{ij} is the cumulative number of matches ,with i,j $i, j \in \{0, 1\}$, where $\mathcal{L}_1$ and $\mathcal{L}_2$ are i and j respectively.

1.3 The Jaccard Coefficient Similarity Score:

The correlation metric similarity score is given by:

$$SM_{\text{jacard}} = \frac{N_{11}}{N_{11} + N_{10} + N_{01}} \tag{4}$$

1.4 Stability Algorithm (Ben-Hur)

As suggested by Ben-Hur's stability algorithm, for every k two random samples $\{S_1, S_2\}$ are drawn from the complete data set, such that every random sample contains a pre-set proportion m of the data. The two samples are then clustered separately and the intersection between them is found along with its corresponding labeling vectors $\mathcal{L}_1$ and $\mathcal{L}_2$. The three previously mentioned similarity scores are then computed between $\mathcal{L}_1$ and $\mathcal{L}_2$. The latter sampling is repeated n-times eventually producing a distribution of similarity scores for every k. The utilized coefficients have been summarized in table .

Parameter	Value
k_{min}	2
k_{max}	10
m	0.35
n	100

Table 1: Ben-Hur Stability Algorithm Utilized Parameters

2 Lab Results

2.1 Encountered Memory Issues

To compute the similarities, we first attempted to store the entire similarity matrix SM in memory then compute the scores based on the matrix. Because of the large size of the SM, running the program has crashed a laptop with 16 GBs of memory. Note that the similarity matrix is a k by k matrix with k being the size of the intersection. To avoid storing the entire matrix in memory, an iterative approach has been taken. The latter loops across all possible pairs of the L matrices and increments N11 N00 N10 N01 on-demand. The iterative

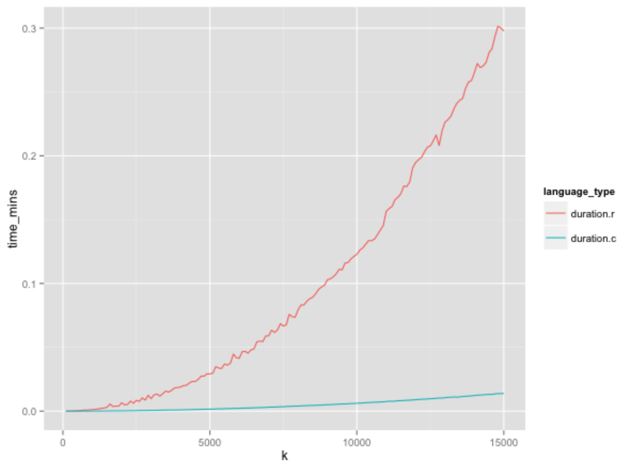


Figure 1: C++ versus R Time Complexity of the Similarity Function

approach has yielded reasonable resources requirements and running times in both R and C++. It should be noted that only the similarity score function was coded in C++, so we imply by "C++ running time" as the running time resulting from a hybrid code of both R and C++.

2.2 Running Time Differences

The C++ code has yielded around 50% reduction in running time as compared to its R counterpart. The running times of the R and C++ codes are summarized in table 2.

Utilized Language	Running Time
C++ / R Hybrid	25.98 mins
R	51.25 mins

Table 2: C++ Running Time Comparison

To further zoom on the true difference between the R and C++ approach, we simulated the running time of just the similarity function implementation on both languages with different input vectors (of size k). The simulation results are summarized in Figure . The results show that while for k sizes between 0 - 15,000, the complexity grows exponentially for the R version while it is relatively linear in the C++ version. It is clear that for large k's the C++ code is orders of magnitude faster than the R code.

2.3 Further Optimizing the Code

2.3.1 C++ Code Tweaks

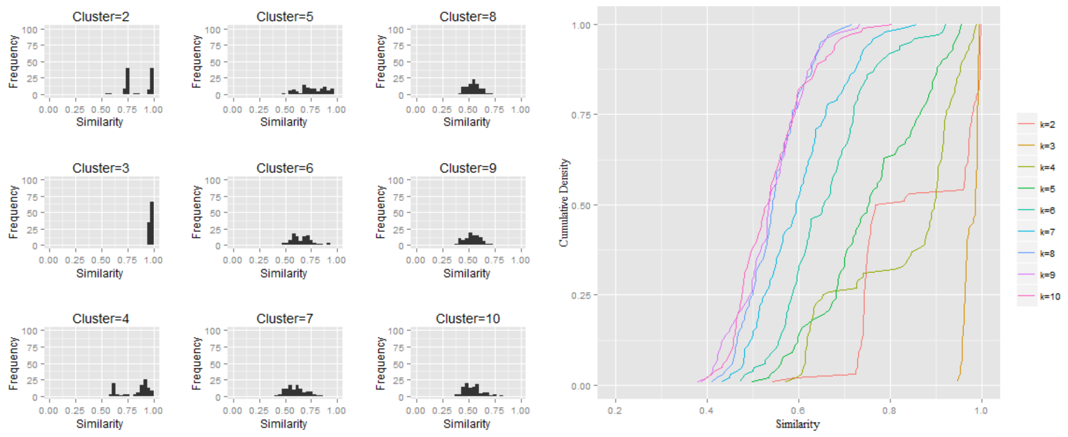
To further optimize the C++ code we tried to cut down the number of nested if's that we have in our current implementation. The simulation results yielded slight to no improvement. Because such improvements rendered our code hard to read, we decided its best to omit that changes and maintain the readability of the code, given that the performance improvements were insignificant.

2.3.2 Taking Advantage of Symmetry

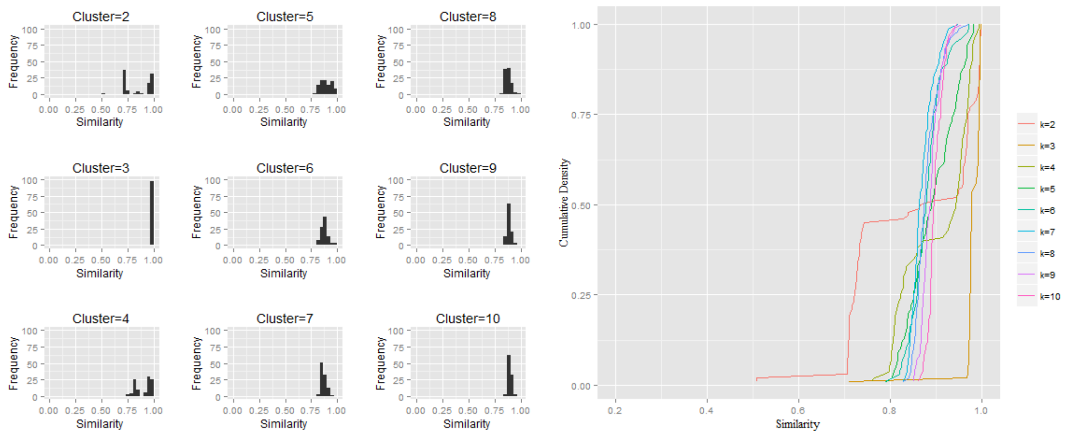
By construction, the similarity matrix ,SM, is symmetric. In order to take advantage of the symmetry, we propose to loop only across the upper part of the matrix. In code the latter enhancement simply means starting the index of the second nested loop from the current value of the outer loop index so that only unique combinations of i and j will be considered. This is expected to cut down the complexity of the entire mechanism by half for both the C++ and the R code.

2.4 Inferring the Stable K

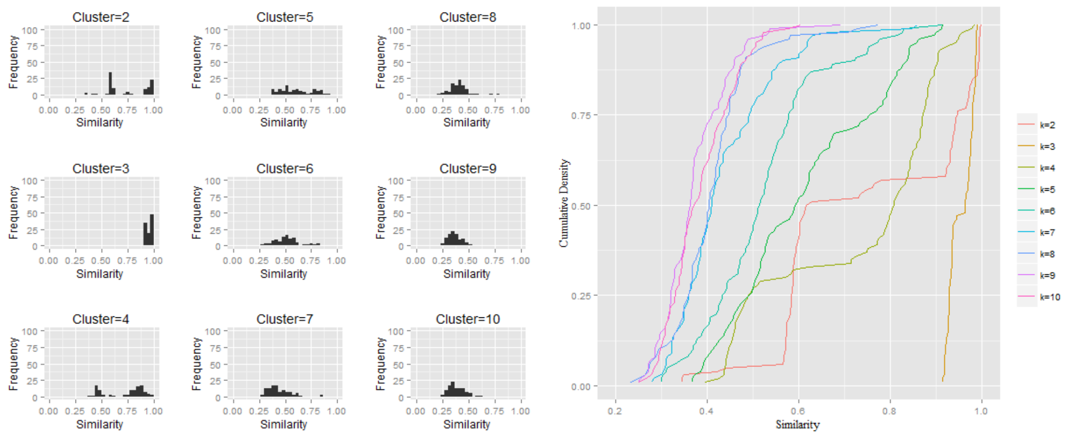
A major purpose behind any clustering implementation is unveiling the "inherent structure" of the data. Ben-Hur implies that the inherent structure of the data should be "stable with respect to sub-sampling." Hence, a good choice of k would be the largest-stable K given a certain similarity score. To infer such K we have plotted the similarity histograms along with cumulative distribution plots of different k distributions with each of the previously discussed similarity scores. The plots are shown in figures 3-(a to c). It can be infered from the plots that after k exceeds 3, the clustering stability is lost (i.e. the similarity distributions tend to have a larger spread). Hence, we can conclude that k=3, is the optimal number of clusters for performing k-means on the data set under study.



(a) Correlation Similarity Distributions



(b) Matching Similarity Distributions



(c) Jaccarad Similarity Distributions

Figure 2: Similarity Distributions For Different Values of k