Carolyn Cotterman
Stat 244
January 22, 2010

Assignment 1

1 Introduction

This assignment explores some important procedures in multivariate statistics: finding eigenvalues/eigenvectors of matrices and grouping data observations into sensible clusters. In exploring these procedures, I will compare some simple methods in C with those from standard routines and packages in R that have similar goals. My code for this assignment is contained in the following files:

- Stat244_hw1_main.c
- Stat244_hw1_header.h
- Stat244_hw1_R_functions

Output from my C and R code is captured in "hw1_c_out" and "hw1_R_out," respectively. While main findings are discussed below, please refer to these output files for additional details.

2 Power Method for Eigenvalues and Eigenvectors

In this problem, I find the eigenvalues and eigenvectors of a real symmetric 5 x 5 matrix using two different methods: first using the power method, which I coded both in C and in R, and then using the dsyevd function from LAPACK.

By definition, if $\underset{\sim}{x}$ is an (n x 1) eigenvector of (n x n) matrix A, then $A\underset{\sim}{x}$ equals $\lambda\underset{\sim}{x}$, where λ is the corresponding eigenvalue. For this problem, I restrict myself to A matrices that are real and symmetric since doing so not only simplifies the calculations, but also ensures that my eigenvalues are real, and that the eigenvectors will form an orthonormal basis for R^P. Now, if we combine all of the eigenvectors of matrix A to form another matrix, call it V, where each column of V represents an eigenvector, and we form diagonal matrix Ω whose diagonal elements are the corresponding eigenvalues, then $A = V\Omega V'$.

The power method allows us to find the matrices V and Ω, described above. This procedure consists of the following steps:

1. Form a (n x n) matrix X_0 that has orthonormal columns. One way to do this is to use the Gram-Schmidt algorithm on an arbitrary (n x n) matrix S. (The Gram-Schmidt algorithm is a "QR decomposition method" that will decompose S into an orthonormal matrix X and an upper triangular matrix R such that S = XR.)

2. Update matrix X_0 until convergence is reached. The matrix should be updated as follows, for i = 1, 2, ...

   (a) Let $T = AX_{i-1}$.

   (b) Find X_i such that $T = X_i R_i$ using a QR decomposition method.

   Convergence is reached when R_i is a diagonal matrix. (In practice, this means that the sum of the magnitudes of the non-diagonal elements of R_i should be less than some "small" threshold value.)

The columns of the resulting X_i matrix will contain the eigenvectors of A, while the diagonal of R_i will contain the eigenvalues.

After implementing the power method in C and in R, I verified my results by first confirming that $A = X_i R_i X_i'$, and then by checking to see that my results match those obtained through running the LAPACK routine called "dsyevd." Indeed, my results from all three methods were the same, aside from some sign differences that cancel out to maintain the $A = X_i R_i X_i'$ equality.

None of my implemented routines took more than a small fraction of a second to find the eigenvalues and eigenvectors of my 5 x 5 matrix. But, it should be noted that the run time for my power method implemented in C was about 11 times greater than my run time using the LAPACK routine (0.00012 versus .000011 seconds). If I were trying to find the eigenvalues and eigenvectors of a larger matrix and/or for many matrices, then the run time difference between these procedures could be an important consideration.

3 Cluster Analysis using the Leader algorithm

Clustering is an unsupervised learning method that consists of grouping "similar" observations together. For this assignment, I implemented a couple of different clustering procedures using a dataset containing crime statistics. More specifically, the data consists of seven crime measurements (e.g., rates of murder, rape, robbery, etc.) for sixteen American cities.

While other definitions exist, I will consider similar observations to be those that are near each other in Euclidean space. That is, I define the distance between observations i and j, which each have p measurements, as follows:

$$d_{ij} = (\textstyle\sum_{q=1}^{p} |x_{iq} - x_{jq}|^2)^{1/2}$$

where x_{iq} is the value of variable q for observation i after having been scaled by the standard deviation of variable q. If the variables are unscaled, then the clustering procedure will place greater importance on the variables that (a) have a larger standard deviation, and (b) are measured in smaller units. For example, without scaling, a difference of 1 meter between the lengths of two boats in a hypothetical boat dataset would appear larger to our clustering algorithm if boat length was reported in terms of centimeters, as opposed to meters.

The leader algorithm can be used to quickly cluster data with minimal memory requirements. It works as follows for a dataset with n observations and p measurements:

1. Specify a threshold value, t, such that if the distance between object i and the cluster leader for the k^{th} cluster is less than t, and no other cluster leader is closer to i, then we assign object i to cluster k.

2. Designate the first observation as the leader of cluster 1.

3. Consider each subsequent observation (j = 2, 3, ... n) in sequence:

   - Determine observation j's distance to the leaders of all pre-existing clusters.
   - If the the minimum of these distances is less than or equal to t, then place observation j in the cluster with a leader closest to it.
   - If the the minimum of these distances is greater than t, then create a new cluster with observation j as its leader.

The total error of a cluster partition can be defined as:

$$err_s s = \sum_{i=1}^{n} \sum_{q=1}^{p} (x_{iq} - \bar{x}_q^{(l_i)})^2,$$

where case i belongs to cluster l_i and $\bar{x}_q^{(l_i)}$ represents the average of variable q across all observations belonging to cluster l_i.

To improve on the initial cluster solution, we can take each observation in turn, and move it to the cluser that minimizes total error. A technique known as "k-means" does this, iterating over the data again and again until moving an observation to a new cluster will not lower total error. To decrease run time, this method may use the following approximation to calculate the error improvment associated with moving observation i to cluster $m \neq l_i$:

$$\Delta err_s s = \frac{n_{l_m}(d_{im}^*)^2}{n_{l_m}+1} - \frac{n_{l_i}(d_{il_i})^2}{n_{l_i}-1}$$

While the k-means algorithm still does not produce a clustering result that is independent of the initial data ordering, it is at least better than using the leader algorithm alone.

Though I knew that the leader algorithm does not always succeed in minimizing the total sum of squares (error) for a given cluster count, the results still surprised me: before implementing this procedure, I had imagined that total error would monotonically increase as fewer clusters were allowed (e.g., as I increased my threshold value.) In fact, the total error when all observations were grouped into one cluster was approximately equal to the error when 8 or 9 clusters were formed, and was only 1/6 of total error when 4 clusters were allowed. It is clear from these results that the leader algorithm does not minimize total error for a given number of clusters. In fact, using different threshold values can result in the same number of clusters, but very different groupings and error levels. (For example, a threshold of 11 through 17 results in the formation of 3 clusters. But, if the threshold is at the lower end of this spectrum, then these 3 clusters generate a whopping error of 816, while if the threshold is 15 - 17, then error is only 296.) From the results of my leader algorithm procedure, it is difficult to determine the optimal number of clusters – none appear to be particularly "natural".

The "kmeans" procedure in R yields at least as good, and usually much better, error values for each cluster count, as compared to the leader algorithm alone. As expected, the kmeans procedure returns the same total error as the leader algorithm when each observation has its own cluster (i.e., we have 16 clusters), and when all observations are grouped into one cluster, since there is only one trivial way to cluster the data into 1 and into 16 clusters. For clusters of sizes in between, however, the kmeans procedure beats the leader algorithm by a large margin almost every time.[1] The results of the kmeans procedure follow the monotonic relationship we were missing from the leader algorithm results. Namely, adding clusters always leads to a lower total sum of squares.

4 Conclusion

In completing this assignment, I have learned that while I am capable of writing programs in C, it often pays to take advantage of standard routines (e.g., LAPACK) and functions (e.g., in R). Benefits of these established routines and functions can include shorter coding time, shorter execution time, and improved results. Still, being able to write my own C code ensures that I will not hit a wall – there may come a time when a built-in function or standard routine is not going to satisfy my statistical needs and I'll need to take matters into my own hands by exploiting the power of C and the knowledge I've gained in Stats 244.

[1]The only exception is when cluster count is two. In this case, my leader algorithm chooses a clustering scheme that yields a slightly lower error than the kmeans' clustering configuration.