Data Structures and Algorithms Final Project: 1000 Genomes Analysis

Luke Mueller: Team Members Sachin Patel and Divy Kangeyan

May 9, 2016

1 Context

The 1000 Genomes Project is the largest public catalogue of human variation and genotype data. These data were collected between 2008 and 2015, with the goal of identifying genetic variants with sufficient frequency (at least 1% in the populations studied). Our data was a subset of the full 1000 Genomes data, containing only a single chromosome. Specifically, we had around 800000 SNPs from 1094 subjects (2188 haplotypes).

2 Objective

Our task for these data was three-fold: first, we compress the dataset by taking advantage of the data structure (i.e., sparsity). Second, we identify the degree to which there exist distinct subpopulations within the data using Principal Component Analysis (PCA). Third, we identify a subset of genetic loci whose similarity has maximum distance (via matrix norms) to the overall similarity matrix (determined in task 2).

3 Compression

Importantly, we sought to achieve efficient, lossless compression for these data. The justification for this is the following: first, we had no prior knowledge about the data. *With* prior knowledge, we could make assumptions about structure in the data and thus make probabilistic imputations, in effect, achieve "lossy" compression. However, as identifying this structure was part of task 2, we did not want to make any false assumptions that would bias our future analyses. Second, we knew there were characteristics in the data we could use to our advantage *without losing* any data. For example, we know the entire data matrix consists of 0s and 1s, thus, if we identify the indices of all the 1s in the matrix, we can infer completely the indices of the 0s.

Storing only indices of nonzero values in a sparse matrix is closely related to *compressed sparse row* matrices (CSR). However, in CSR, the assumption of nonzero values is relaxed to include any real number (i.e., not just 1s). Thus a CSR matrix will store the indices of nonzero values, and their respective value. The `sparse` module of the `SciPy` library converts a sparse matrix to CSR with a single line of code. We tried compression with both methods, under the tight assumption of 1s being the only nonzero values in the matrix, and under CSR. We achieved compression from 3.6GB down to 0.98GB and 1.9GB respectively.

Not satisfied with these results, we attempted to take our compression a step further. In particular, since the pairwise, adjacent data matrix columns represented single individuals with 2 haplotypes, we could collapse these columns through summation and thus reduce the dimensions of the data to 800000 by 1094. This method is still "lossless" though it operates under slightly modified parameters to the original data matrix. Converting this matrix to CSR achieved compression down to 0.43GB.

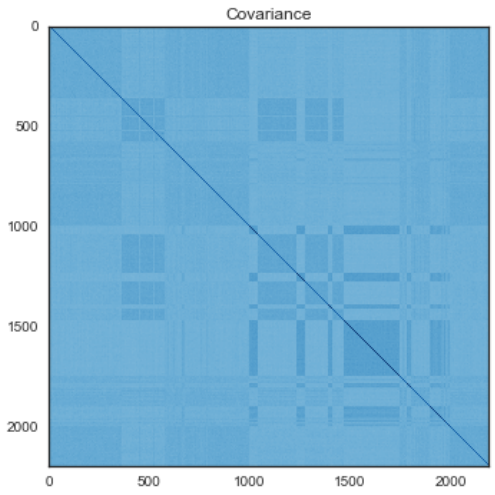

Figure 1: Covariance Matrix: All 2188 Haplotypes.

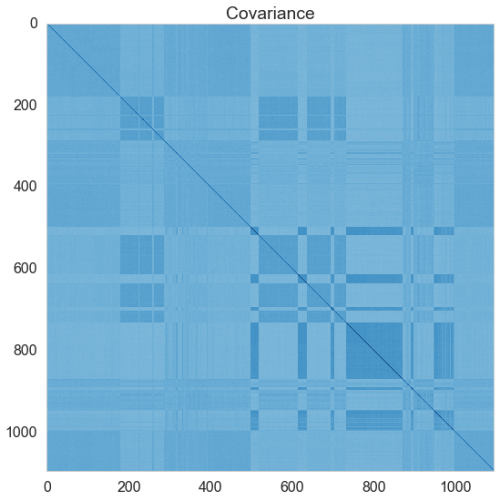

Figure 2: Covariance Matrix: Collapsed Over Individuals.

Finally, we realized our storing was primarily inhibited by use of 16 bit integer values (since each row/array contained up to 1094 elements). By partitioning the rows into blocks of size ¡ 256, we could store indices at uint8 data type. Using dictionaries for each partition, and repeating the process over all 800000 rows, we achieved compression down to 0.16GB.

To summarize, we achieved lossless compression of the original dataset down to 0.16GB. Though we lost information regarding the haplotype phase, the data from each locus is kept entirely. The algorithm runtime complexity for traversing the original matrix was at worst, $\mathcal{O}(n^2)$. After storing, we could recreate the original matrix relatively quickly with complete accuracy.

4 Calculating the Covariance Matrix

The covariance matrix was used as a genetic similarity matrix to identify subpopulations. Besides being useful in its own right, upon calculating the variance covariance matrix, we can easily extract Principal Components via eigendecomposition. We used the following formula to calculate the covariance:

$$Covariance(\mathbf{X}) = \frac{(\mathbf{X} - \mathbb{E}(\mathbf{X}))(\mathbf{X} - \mathbb{E}(\mathbf{X}))^T}{N-1} \tag{1}$$

Because the data matrix was so large, we needed several additional steps to perform the covariance calculation. First, we used the CSR formatted data matrix to subtract the column mean from each column. Since the data matrix was now ”dense”, we could not calculate dot products directly without consuming substantial memory. Thus, second, we partitioned the full data matrix row-wise by an arbitrary number, and saved these submatrices to hard-disk space. Third, we loaded each submatrix back into memory, performed the dot product and divided by $N-1$, and appended the resulting matrix to a new array. Then summed over these submatrices to obtain the full covariance matrix, which is represented visually through heatmaps in figures 1 and 2.

5 Subpopulation Identification

Next we performed PCA by extracting the eigenvalues and eigenvectors of the covariance matrix. The complexity of PCA involves two main tasks: the covariance matrix computation $\mathcal{O}(p^2n)$, and eigendecom-

position of the covariance matrix $\mathcal{O}(n^3)$. Hence the total time complexity of PCA is $\mathcal{O}(p^2n + n^3)$. In our case, the PCA was incredibly fast since we operated on a drastically reduced n space (via partitions).

The principal components were features/columns with the largest eigenvalues. We can determine an optimal number of principal components for analysis by minimizing both reconstruction error and number of principal components. Ideally, the total variance of the data can be explained by a few principal components (figure 3).

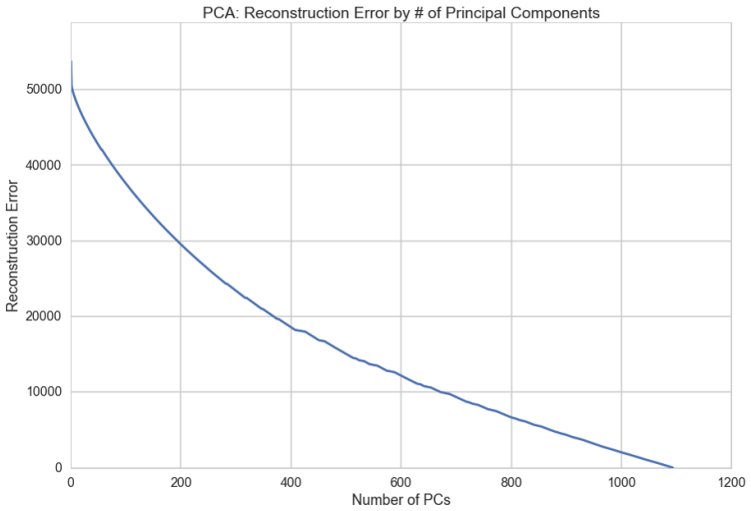

Figure 3: Principal Component Analysis

We decided to continue analysis using the first 2 principal components, as they explained a substantial amount of variance in the data, relative to the other eigenvectors. Observing the data on its principal axes revealed several distinct subpopulations - mostly related to subject demographics (figure 4). In particular, we discovered the subjects were roughly divided by continent.

We thus proceeded to identify subsets of 1000 genetic loci within each subpopulation: the Americas, Asia, Europe, and Africa. By partitioning the original data matrix by these subpopulations, we hoped to find distinct clusters of genomes within each subpopulation. First, we attempted to use the `DBSCAN` clustering algorithm (from the `sci-kit learn` python library), as it takes CSR matrices as input, is scalable to large samples, allows uneven cluster sizes, and permits specifying a minimum neighborhood size for each cluster (i.e. 1000). Unfortunately, since the complexity of `DBSCAN` is $\mathcal{O}(n)$, we ultimately decided this method was infeasible.

In our second approach, for each locus, we calculated the difference of means between the subpopulation of interest and the total population. We then ranked loci for each population, and selected the top 1000 loci (largest mean difference) to generate a new 1000 by 1094 matrix. Finally, we performed PCA on these matrices in hopes of identifying clusters of genomes that segregated subpopulations neatly. Though there was definitely evidence of clustering over the first two principal components, it was not quite what we expected (figure 5). In a future analysis, we might refine this method to introduce standardization over the means so our results were less susceptible to high variance.

Finally we were able to calculate matrix norms for these submatrices. Using the Frobenius norm (default for the `numpy.linalg` module), we calculated the following values: full data matrix: 13482.9, AMR: 1344.9, EUR: 1350.4, AFR: 1254.9, and ASA: 1234.1.

Genome ancestry Continent.png

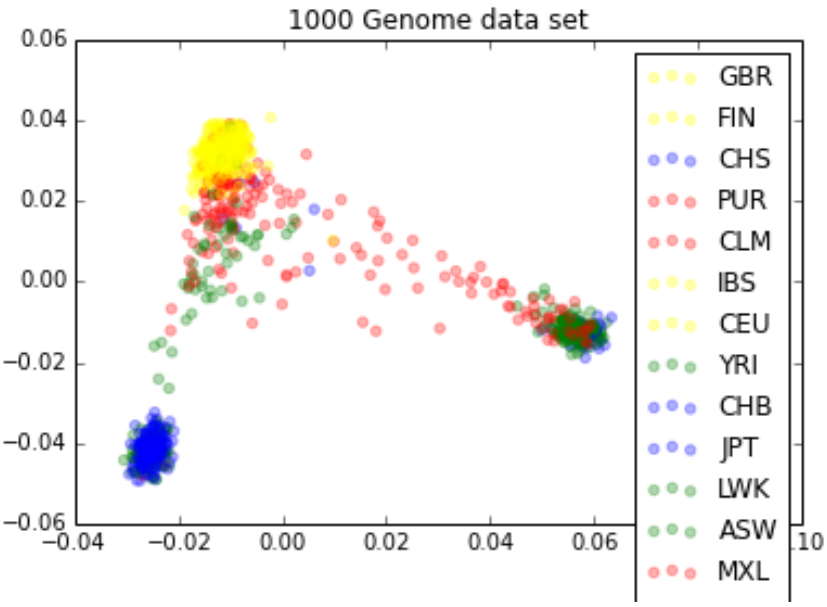

Figure 4: PC1 vs. PC2: Subject Demographics

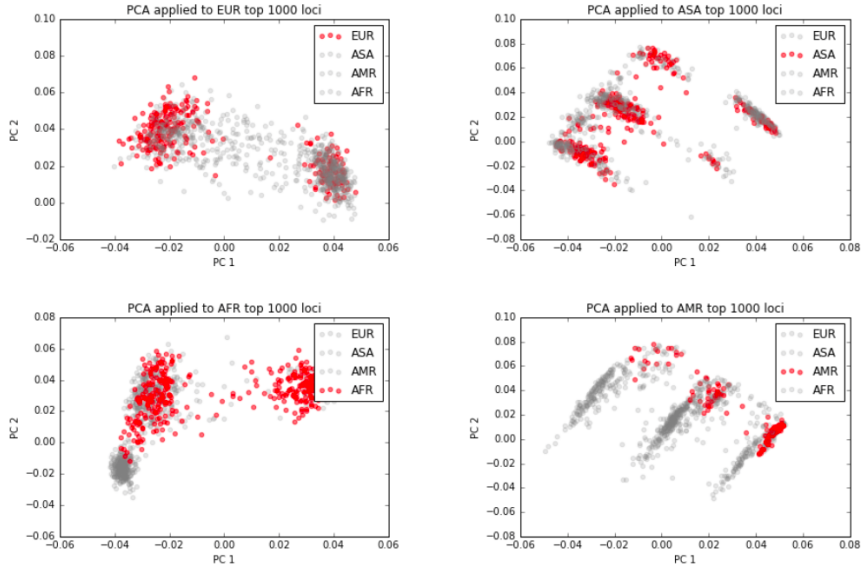

Figure 5: Subpopulation PCA