ESS201: Programming-II
Module: C++

Jaya Sreevalsan Nair
jnair@iiitb.ac.in
International Institute of Information Technology, Bangalore

Project Document (v2.0) on November 14, 2018
Submission Deadline: 11:59:59 pm IST, November 30, 2018 (Friday)

The project will be done by pre-determined groups of 6 members each. Given are 8 tasks where each group member will perform a task, different from the rest of his/her group. Each group will be given a unique combination of tasks which need to be integrated to give a common application.

Goal: Your group will be building a small-scale tabular data processing toolkit.

Data:

- We will be using multivariate, time-series, numerical data for regression tasks, from the UCI Repository[1].
- Each dataset has a $M \times N$ matrix, where M is the number of records/instances, and N is the number of numerical variables. Each row in an excel sheet corresponds to a record, and each column to a variable.
- In order to reduce the dimensionality of the dataset, one can convert the rectangular matrix to a symmetric square matrix of either M or N, using a symmetric pairwise relationship computed for either the records or the variables, respectively. In this project, we explore the pairwise relationships of correlation, covariance, and similarity.

  Following are the characteristics of such a matrix:

  - Such a matrix is positive semidefinite.
  - Such a matrix requires an ordering (or permutation) of records in the case of $M \times M$ matrix, and similarly, that of variables in the case of $N \times N$ matrix. This permutation will be applied along both rows and columns. Thus, the i^{th} row and the i^{th} column correspond to the same entity (i.e. a record or a variable), and matrix element at location (i, j) contains the value/score corresponding to the relationship between the i^{th} and j^{th} entities.
  - In this project, we will maintain that all matrix elements are real numbers between 0 and 1. (correlation values are from -1 to 1). In cases where the elements deviate from this rule, we can transform the matrix using *normalization* or taking *absolute value* of the matrix element.

[1] https://archive.ics.uci.edu/ml/datasets.html?format=&task=&att=num&area=&numAtt=&numIns=&type=mvar&sort=attUp&view=table

- A *feature vector* of a record/vector is a vector of the values of numerical variables (in this case), in a specific order of the variables.
- In this project, an output file of a matrix $M \times N$, referred to as *data matrix*, will use the following format akin to the csv format:

```
M, N
a11,a12,a13, ..., a1N
a21,a22,a23, ..., a2N
...
aM1,aM2,aM3, ..., aMN
eof
```

  where the matrix size is given in the first line, followed by M lines, one for each row. Each of the M lines contain N numerical values, separated by commas.

Tasks:

- Create a C++ class for the record. This must be generic, as your application should be able to read any file of this data type, irrespective of the number of columns (i.e. variables) or number of rows (i.e. records).
  - The file will be organized in a way that first n columns (from left-to-right) will contain the non-numerical variables, and the rest of the $(N - n)$ columns will contain the numerical variables. Hence, in a simplified version, expect the user to say what n to consider for an input file.
- Create a C++ class for the symmetric square matrix. This must be generic as well.
- Use object-oriented concepts (classes, composition, inheritance, polymorphism) to the maximum extent possible in this project.

The eight tasks, where Tasks 1-4 are to be done on a multivariate tabular data, and Tasks 5-8 are to be done on a matrix.

1. **Similarity Matrix and k-means Clustering:** Read from input data file. Compute pairwise similarities between records. Use cosine similarity of feature vector consisting of only the numerical variables, for similarity computation. Further, use the similarities in your implementation of k-means clustering to compute the user-input k clusters. Output the records along with an additional column of cluster ID, to a csv file. The order of the records in this output file must be the same as the input file. Output to a second csv file the similarity matrix of size $M \times M$, which captures pairwise similarities between records in the rows.

    - Clustering algorithms typically need distance matrices and not similarity matrices. Dissimilarity matrices may be considered analogous to distance matrices. Hence, for a similarity value in the range $[0, 1]$, the corresponding dissimilarity value is one minus the similarity value.

2. **Row Ranking and Distance Matrix:** Read from input data file. Compute the Euclidean distance of each record from a *hypothetical average* record, which uses the average of each numerical variable in its feature vector. Normalize the distance values. Sort the records based on increasing order of normalized distance to the hypothetical record. Now compute the Euclidean distance between records taken pairwise. The distance matrix is assigned based on the computed ordering of the records, it has 0 in diagonal elements, and the corresponding distance values in the non-diagonal elements. Output a vector of the ordering of the M instances using its index in the original $M \times N$ data matrix. Output to a csv file the distance matrix of size $M \times M$, which captures pairwise distances between the records.

    - Normalization of a set of values implies transforming the values in order to confine the range of the transformed values in $[0, 1]$. Hence, compute the minimum and maximum of the values, and compute the normalized value as $\frac{value-min}{max-min}$.

3. **Variable Filtering and Correlation Matrix:** Read from input data file. Compute auto-correlation for each numerical variable. Sort the correlation values, which are in the range $[-1, 1]$. Select the variables which have the absolute value of the autocorrelation value greater than 0.5. Let the number of such variables be n, such that $n \leq N$. Filter out $(N - n)$ variables, whose correlation values are in the range $[-0.5, 0.5]$. Thus, now we have $M \times n$ matrix of data. Compute correlation between the n variables, taken pairwise at a time. Sort the numerical variables based on their auto-correlation values. Use this order of numerical variables to assign a correlation matrix, where the diagonal elements are the autocorrelation values, and non-diagonal ones are the correlation values between the corresponding numerical variables. Output a vector of the ordering of the n variables using its index in the original $M \times N$ data matrix. Output to a csv file, the correlation matrix of size $n \times n$, which captures the correlation values between highly correlated numerical variables.

    - Filtering out implies discarding. This can be used to reduce the dimensionality of the dataset.

4. **Column Ranking and Covariance Matrix:** Read from input file. Compute the variance of values of each of the numerical variable across all the records. Sort the numerical variables using the normalized variance values. Then compute the covariance matrix by computing pairwise covariance between two variables. The diagonal elements will be the variances of the variables corresponding to the index of the diagonal element. Find the minimum and maximum values of the elements in the covariance matrix, and normalize the matrix elements using these extrema values. Sort the variances in the descending order to find a permutation or ordering of the indices of the variables. Reorder the columns and rows of the covariance matrix using the permutation, such that the diagonal elements are in a descending order. Output a vector of the ordering of the N variables using its index in the original $M \times N$ data matrix. Output to a csv file, the normalized covariance matrix of size $N \times N$, which captures the covariance values between the numerical variables.

    - The correlation coefficient computed in Task 3 is different from normalized (co)variance values computed in this task. In the latter, normalization is done based on the set of covariance values of the numerical variables, and in the former, the coefficient is a dimensionless value, computed using the variance and standard deviation.

5. **Eigenvalue Computations for a Symmetric Square Matrix:** Compute the first p eigenvectors of a symmetric square matrix (e.g. covariance matrix, similarity matrix, correlation matrix) of size P, using the algorithm of Power Iteration method as given in `[HK02]`. Use these eigenvectors to compute the largest p eigenvalues. Print out the p eigenvectors and eigenvalues, using the formula of eigenvalues and eigenvectors $Ax = mx$, where A is the matrix, m is the eigenvalue, and x is the corresponding eigenvector. Consider p to be a user input, where k is less than the size of the square matrix.

6. **Minimum Degree Reordering of Symmetric Square Matrix:** Implement a modified version of the "basic minimum degree ordering" algorithm given in `[GL89]`. The nodes referred to in the paper correspond to the objects in the matrix, of size P. The algorithm for this involves the following:

   (a) Degree of an object, in our case, is the sum of the non-diagonal elements in the row. The definition of degree of an object is the only modification in this project.

   (b) Once the object with minimum degree is found then the index of the object is added to the vector of new permutation and the corresponding row and column are removed, giving a submatrix which is a square matrix of size $(P - 1)$.

   (c) The node of minimum degree is then computed in the submatrix, and the process is implemented recursively, until no more elements are remaining of the input square matrix.

   The output is the new permutation of the "objects" in the matrix.

7. **Reverse Cuthill-McKee (RCM) Reordering of Symmetric Square Matrix:** Implement a Cuthill-McKee algorithm as given in `[CM69]`, which is based on Breadth First Search of a graph, whose adjacency matrix is the sparsified version of the input square matrix. Finally reverse the ordering from CM to get RCM ordering.

   Sparsify the input square matrix, whose values are in range $[0, 1]$, by forcing values less than 0.75 to be 0. Instantiate an empty queue Q and empty array for permutation order of the objects R.

   The CM algorithm is as follows[2]:

   - S1: Using the afore-mentioned definition of degree of an "object" (node in the graph), find the object with minimum degree whose index has not yet been added to R. Say, object corresponding p^{th} row has been identified as the object with minimum degree. Add p to R.
   - S2: As an index is added to R, and add all neighbors of the corresponding object at the index, in increasing order of degree, to Q. The neighbors are nodes with non-zero value amongst the non-diagonal elements in the p^{th} row.
   - S3: Extract the first node in Q, say C. If C has not been inserted in R, add it to R, add to Q the neighbors of C in increasing order of degree.
   - S4: If Q is not empty, repeat S3.
   - If Q is empty, but there are objects in the matrix which have not been included in R, start from S1, once again.
   - Terminate this algorithm once all objects are included in R.
   - Finally, reverse the indices in R, i.e. `(swap(R[i], R[P-i+1]))`.

   The output is the new permutation of the "objects" in the matrix.

8. **Robinsonian Reordering of Symmetric Square Matrices:** Use the given square matrix as a distance matrix in the algorithm for Robinsonian reordering `[Rob51]` in `[BBR+16]` as a "bipolarization algorithm." The output is the new permutation of the "objects" in the matrix.

[2] http://ciprian-zavoianu.blogspot.com/2009/01/project-bandwidth-reduction.html

While each student within a group has been assigned a different task, the responsibility of the group is to integrate these different tasks to build a cohesive application. Consider sequentially implementing at least two tasks together wherever possible.

- Tasks 1-4 is the input data files in csv files, where the first line is field names.
- Tasks 1-4 compute a symmetric square matrix in the form of covariance matrix, similarity matrix, correlation matrix, outputted in the format prescribed in this document.
- Tasks 5-8 can be implemented on the symmetric square matrix, inputted using the format given above. The second input to these tasks may be the permutation of the indices of the "objects" defining the matrix (e.g. either records or numerical variables, from the original data matrix). This index ordering will enable you to track the objects in the original data matrix, after the permutation involved in the Tasks 5-8.

Sample Datasets:

1. Air Quality: 9358 records, 15 attributes (2 time variables, 13 numerical variables)
   https://archive.ics.uci.edu/ml/datasets/Air+Quality
2. Appliances Energy Prediction: 19735 records, 29 attributes (1 time variable, 28 numerical variables)
   https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction
3. Beijing PM2.5: 43824 records, 13 attributes (5 time+non-numerical variables, 8 numerical variables)
   https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data
   Note: This dataset contains invalid values for some combinations of records and variables.

References:

[BBR+16] Behrisch, Michael, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and JeanĄ]Daniel Fekete. "Matrix reordering methods for table and network visualization." In Computer Graphics Forum, vol. 35, no. 3, pp. 693-716. 2016.

[GL89] George, Alan, and Joseph W. H. Liu. "The Evolution of the Minimum Degree Ordering Algorithm." SIAM Review, vol. 31, no. 1 (1989): 1-19.

[HK02] Harel, David, and Yehuda Koren. "Graph drawing by high-dimensional embedding." J. Graph Algorithms Appl. 8, no. 2 (2004): 195-214.

[CM69] Cuthill, E. and J. McKee. "Reducing the Bandwidth of Sparse Symmetric Matrices." The 1969 24th national conference 1969, ACM. (1969): 157-172.

[Rob51] Robinson, William S. "A method for chronologically ordering archaeological deposits." American antiquity 16, no. 4 (1951): 293-301.