Dimensionality Reduction/Low dimensional Embedding

1 Fundamentals

1. Dimensionality reduction
   (a) Load the dataset of user votes (simplified I provide) from the class website. In the next questions, we would like to create a map of movies, in which similar movies are close in space. We can use for this the principle that two movies that are liked by the same people should be similar
   (b) Use `pivot` pandas function to create a matrix such as movie names are indices and users are columns, i.e., features. Replace nan by zeros
   (c) Use PCA (from sklearn) to embed directly this data in 2D, and check using `plotly` scatterplot that some similar movies are closeby in that space (check for instance movies from well-known series such as starwars, lord of the rings, harry potter, etc.
   (d) Do the same with tSNE (still from sklearn). Vary the perplexity and observe the differences
   (e) Now apply first a PCA in 50 dimensions, and then apply a tSNE on the result in two dimensions. The structure should be clearer

2. Correlation to network, network to graph embedding
   (a) Using sklearn `pairwise_distances`, compute the `cosine` similarity between movies. (Note: you could use another distance measure...)
   (b) Use a threshold to keep only strong values of similarity, for instance 0.5 to begin with. You will have to make several trials with the value after seeing the graph. (Note: you could use backbone extraction to get a better graph)
   (c) Using `networkx`, create a graph from the thresholded similarity matrix (e.g., `from_pandas_adjacency`. Don't forget to add the node names (you can get them from the pivoted matrix index, for instance, `set_node_attributes`)
   (d) Remove self-loops, and singletons (nodes without edges) with the corresponding functions
   (e) Plot it (better with Gephi, after export `write_graphml` and check that the edges make sense.
   (f) Using library `karateclub`, use `node2vec` to embed the graph in small dimensions, e.g., 8 dimensions
   (g) Use t-SNE or PCA to embed the 8 dimensions in 2, and use an interactive plot to check that you arrive back to a meaningful map of movies.

2 Going further

3. Coding your own PCA
   (a) One strength of PCA is that it is very simple to code using linear algebra. We will compute it manually and check that we get the same results as with the sklearn function
   (b) Start by centering the data, for instance using `df-df.mean()`
   (c) Compute the covariance matrix using `np.cov` (be careful to compute the covariance between variables, not elements !)
   (d) Compute the eigenvectors of the covariance matrix using numpy `LA.eig`
   (e) Check that the eigenvectors you obtained are similar to the `components` obtained by sklearn. Don't forget that we are looking for those associated with the highest eigenvalues !
   (f) You can recompute the final embedding in 2D using a simple matrix multiplication. The result should be identical to the one obtained by PCA