Dimensionality Reduction/Low dimensional Embedding

1 Fundamentals

1. Getting started: PCA
   (a) Get ready to use the movie dataset. Keep columns [budget, popularity, revenue, runtime, vote_average, vote_count] as numerical columns. Clean them by removing rows having undefined (na) values. Remove 0 in Budget, Revenue, runtime and vote_count.
   (b) Using sklearn library, apply `PCA` algorithm in 2 dimensions
   (c) Create a dataframe with 3 columns: "d1" and "d2" for each of the 2 new dimensions, and "cluster", in which you insert the values of the clusters obtained in the previous class with k-means (3 clusters on normalized data, for instance)
   (d) Plot a scatterPlot of this data, with colors corresponding to clusters.
   (e) Standardize the dataframe and do the same process. You should observe a clear difference between the 2.
   (f) A strenght of PCA is that it is interpretable: each new dimension is a linear combination of original ones. Print using `pca.components_` the coefficients of this linear combination, with and without normalization, and try to understand what it means. In particular, what d1 mostly capture that d2 do not capture ?
   (g) Compute a correlation coefficient between d1 and d2 to check that PCA indeed captures uncorrelated features.

2. Comparing and making sense of embeddings
   (a) Using sklearn, compute TSNE and IsoMap in 2 dimensions, and create dataframes with 3 columns as before.
   (b) To make sense of the results, we need to be able to explore which movie is where in the embedding. To do so, add to each dataframe a fourth columns, which contain the title of the movie. To do so, you can do something like `df_pca["title"]=list(df.loc[numerical.index]["title"])`.
   (c) You can now create an interactive scatterPlot, using function `scatter` from library `plotly.express`, with the title in the parameter `hover_data`.
   (d) Explore the interactive plot for the different embeddings, and check if you can make sense of it, in particular checking movies at the extremities of the plot (you might want to search some of them on Wikipedia to see if they have obvious characteristics).
   (e) An objective of embedding is that elements that are more distant in the original space should be more distant in the reduced space, i.e., it somewhat preserves the order of distances. To check this, compute the correlation coefficient (Pearson/Spearman) between the distances before and after applying the dimensionality reduction. You can use `cdist` from scipy to compute distances, `flatten` to transform a matrix into a single dimensional vector, and `triu_indices` to keep the upper triangle of the matrix (the distance matrix is symmetric, and keeping both triangles would artificially raise the correlation)

2 Advanced

3. Correlation to network, network to graph embedding

   (a) 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 persons should be similar. On the website of the class, you can find a link to a vote file, which is a simplified version of the one available on Kaggle. Load this dataset.
   (b) Use `pivot` pandas function to create a matrix, and replace nan by zeros
   (c) Using sklearn `pairwise_distances`, compute the `cosine` similarity between movies
   (d) Use a threshold to keep only strong values of similarity. You will have to make several trials.
   (e) Using networkx, create a graph from the thresholded similarity matrix. Don't forget to add the node names (you can get them from the pivoted matrix index, for instance)
   (f) Remove self-loops, e.g., `g.remove_edges_from(nx.selfloop_edges(g))`
   (g) Plot the graph and check that edges seem meaningful.
   (h) Using library `karateclub`, use `node2vec` to embed the graph in small dimensions, e.g., 8 dimensions
   (i) Use t-SNE or PCA to embed the 8 dimensions in 2, and use an interactive plot to check that you now have a meaningful map of movies.

4. 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