Clustering

1 Fundamentals

1. Clustering: getting started
   (a) Get ready to use the same dataset as for the previous class. 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 `KMeans` algorithm, with 3 clusters.
   (c) Compute the centroid (mean values for each variable), and the size for each cluster. A flexible way to proceed is to extract the clusters (`predict`), add it as a new column in the dataframe, then compute statistics by cluster in the dataframe, for instance with `.groupby("cluster").agg(['mean',"count"])`
   (d) Remember that k-means is based on the notion of distance between points in the D-dimensional space, and thus is sensible to magnitudes. Standardize the data with `StandardScaler`, and re-run the clustering
   (e) To better observe the difference, we can plot (`sns.scatterplot`) the relation between a variable with large values (x="budget"), a variable with low values (y="vote_average"), with dot colors corresponding to clusters (hue="cluster"). Observe the clear difference between normalized and non-normalized versions.
   (f) If you had to give a manual label to those clusters, to describe the movies they contain, what would it be?

2. Evaluation and number of clusters
   (a) Compute the silhouette score using method `silhouette_visualizer` from package `yellowbrick`, plot the silhouette score and interpret it.
   (b) We would like to find the optimal number of clusters. Apply the silhouette score method: plot the relation between k and the silhouette score, and search for a maximum value, or a value where there is a "summit" (good luck).
   (c) Apply now the elbow method. The y axis can represent various scores that decrease with k, a simple one is to take the `inertia_` parameter of your k-means object. Look for an elbow to find the right number of clusters (good luck).

2 Advanced

(a) Apply DBSCAN method, with the default parameter eps=0.5. Compute clusters centroids and size. Remember the number of clusters is found automatically.
(b) Check which movies are in some of the small clusters. One way to proceed is to 1)create a dataframe filtered exactly like the one you used for training, but in which you retain the movie title. 2)Add the cluster column to that dataframe.3)Now you can filter on the cluster number to see the corresponding movies.
(c) To check that you understood correctly the eps parameter, make a guess: if you increase it, will you lower or increase the number of clusters? Check.
(d) Check the `BayesianGaussianMixture` class of sklearn, and try to understand its parameters `n_components, covariance_type,tol,max_iter,init_params`.
(e) Run it in different versions and explore the results

3 Going Further

(a) We could consider (although unlikely) that our clusters should correspond to different movie genre. Extract the first genre from the genre column for each movie. Considering that as our clustering ground truth, compute the AMI. Explore parameters of your favorite algorithm such as to find those that maximize the similarity between found clusters and genres.

(b) Going back to the Gaussian Mixture, run it with a small number of clusters, and for 2 features, plot the actual gaussian mixture (heatmap/kdeplot)