12/11/23

DS 210

Designing and Performing a K-means Clustering Algorithm to Analyze John Snow's Cholera 1855 Report

by Will Shore

For my DS 210 final project I chose to analyze this data set that John Snow collected in 1855 to track the spread of cholera deaths across London. In the mid 19th century, London was facing a cholera epidemic and at one point there were over 600 deaths a week related to the disease. Snow, a physician operating in London, answered the call to help and created an early mapping system to track the spread of the virus. After filtering neighborhoods by where they access their water, he discovered that the most affected portions of the city were sourcing their waters close to sewers. In tracking the spread of disease, Snow became the founding father of epidemiology. Snow's set contains 1,852 buildings in the Soho area with a varying number of deaths reported for each home. In all, 661 deaths are reported in this data set, and my plan is to use the available data to predict deaths across clusters of houses.

To accomplish my project I am going to split the set into training and testing sets, then create a k-means clustering algorithm that would cluster similar homes together in the training sets. After doing so I would collect the deaths associated with these clusters and use the resulting averages to predict deaths in the clusters of the test set. Additionally, to provide a visual element to the project I wanted to create a graph to plot the individual clusters/homes in the data set.

Methods:

Processing

Before analyzing the data, I need to process it in rust. To do this I created a module called graph.rs that reads the csv file from its file path and turns each building/item into a vector of records. Once I had the data processed into a vector of records, I stored its results in a new variable called graph_data. Here is an example of what the graph_data looks like: Record { data: [1577.0, 0.0, 0.0, 0.0, 0.0, *234.96*, *5.43*, *169.55*, 0.0, 529564.9309, 181044.652] }. The features of the data set are in the order of: ID, relevant deaths (cholera deaths), non relevant deaths (other causes), total deaths, presence of a pestfield, distance to pestfield, distance to sewer, distance to broad street pump, a death dummy variable, X coordinates of the house, and Y coordinates of the house.

However, there are only 3 important features for clustering each item, those being the the distance to pestfield, the distance to sewers, and the distance to the broad street pump. Supporting Snow's analysis, these variables will tell us how likely these homes will be exposed to cholera based on their distance to risky points of access. The broad street bump itself is Snow's claim to fame, for after completing his mapping he was able to isolate the broad street pump as the origin of many cholera cases, and after ordering the city to close the pipe the rates of cholera in the area dropped significantly.

With my relevant data identified, I need to split it, process it, and cluster it. To do this I created another module called kmeans.rs that will be the home for my processing functions. One such function is called extract_features that reads each record and stores the 3 relevant features in a vector of vectors. Afterwards I scaled the data using a min_max_scaling function that normalizes the data based on the largest and smallest values of the feature (Fig. 1). This keeps every data point within the range of 0 to 1, which makes it easier to process for comparison and graph later on. There are many options to feature scale data, but I wanted each variable to be weighted equally while still preserving its variance.

Clustering

Next we need to review our training data and choose the best training iteration of k-means. In this case I am going to create 9 clusters to represent the 9 unique death values from the data set (0,1,2,3,4,5,6,8,12). The k-means function will first choose a random data point within the training set to initialize its centroids, and over 100 iterations will move those centroids closer towards similar values by taking the mean of nearby data points. However, since our data has no ground truth to compare itself against, I will calculate a silhouette score for the clusters. The silhouette score is a measure that quantifies how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to 1, where a high value indicates well-defined clusters, a score near 0 suggests overlapping clusters, and negative values indicate that data points may have been assigned to the wrong cluster (Fig. 2). Through each initialization the program is calculating and comparing the results to find the highest silhouette score. To provide the user a visual representation of the clustering algorithm, I have included print statements that showcase how many nodes are assigned to each cluster and what that iteration's silhouette score is. Once the best iteration is identified, the program will store its characteristics outside of the iterations loop. The most important of these characteristics is a vector called cluster_to_death, which stores the average deaths associated with each cluster.

With our variables prepared, we now need to compare it against our test set. Similarly, the program will run 10 iterations of k-means on the test split, but this time when it clusters the nodes together it is also multiplying them by the average deaths associated with each cluster (which is stored in cluster_to_death) to form a prediction for the number of deaths in the iteration. The sum of these predictions are then compared against the actual number of deaths recorded in the test set and I store the differences across each iteration. I similarly included print statements that showcase the clustering results. The test iteration that has the lowest difference between the predicted deaths and actual deaths gets its values saved and has its nodes and centroids graphed. Additionally, I include another silhouette score analysis on the test set to ensure that nodes are properly assigned.

In all, this program is iterating through the training set to find a clustering output with the most well defined clusters, and after doing that it calculates the average deaths associated for each cluster. The program takes those averages and multiplies them with the nodes for each cluster in the test set to form our predictions. After seeing which iteration produces the lowest difference between the predicted deaths and actual deaths, it graphs the outcome using rust plotters.

Plotting

Rust plotters is a library for rust that allows coders to create 2d and 3d charts for visualizing data. A website for its documentation can be found here. This program will output 2 graphs, one for the best iteration of the training set, and one for the best iteration of the test set. The colors for the clusters are randomly generated each time to meet the demand for any number of clusters the user may input. These colors are composed in an RGB fashion, with each randomly generated number fulfilling the r, g, and b values. However, I warn against implementing too many clusters or the colors may begin to look similar. To read the graph, each circle represents a node in 3d space that is plotted from its normalized features, and the triangle marker represents the centroid of the cluster.

Tests

To validate the functionality of my programs, I have incorporated three tests in the main.rs file. The initial test evaluates the effectiveness of my plotting function by checking if it successfully generates a graph. In this test, the function is supplied with generic values, and its acceptance of these values is confirmed by the successful execution leading to the final Ok() return. This outcome serves as an indicator that the function is producing the expected output. The second test is designed to ascertain the proper functioning of the k-means algorithm on a given dataset. After creating a fundamental dataset and specifying variables such as the number of clusters, the function is executed. The test then validates that the number of cluster assignments matches the count of the original data points, and ensures that the number of centroids aligns with the specified number of clusters. The final test guarantees the program's ability to accurately partition a dataset. By providing it with ten data points, the test instructs the program to split the data according to a 7:3 ratio. Subsequently, the lengths of the split sets are examined, and assertions are made to confirm that they equal their designated lengths and collectively add up to the length of the original dataset.

Afterword

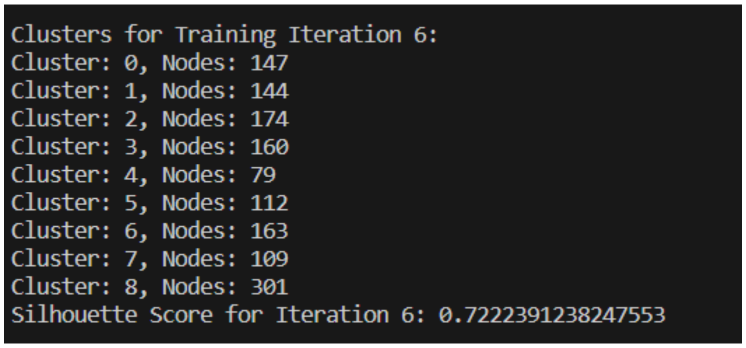
At its best, I have had predictions that had 0 differences between predicted deaths and actual deaths, and at worst I have had predictions that are over 100 deaths off of the mark. Since the number of nodes in each cluster changes with each call of k-means, I wanted multiple iterations through both the test set and training set so that I can pick out the top performers of each iteration. Without a true ground truth to compare to, there is only so much I can do for forming a predictive model, but even then the average of my differences tend to hover around 24 deaths. If Snow had a program like this in the mid 1800s he could have plotted every house in Soho and predict how many deaths will occur in these clusters. Who knows how many lives this could have saved, but because of Snow there is a whole field of epidemiology where scientists are simulating the spread of disease on increasingly sophisticated models. Thanks to the efforts of one man we are saving more lives than he could possibly imagine.

How to Run:

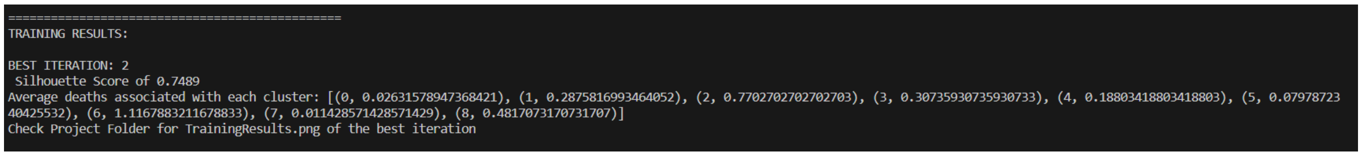
Provide the program a file path to read the csv file "john snow." After doing that, specify the number of clusters you want k-means to produce, and the number of iterations you want to create. Additionally you will have to adapt the plot.rs to properly see the resulting png files. If the graphs are not changing with each call of the program, go into the plot.rs module and change the root path to match the location of your project folder. With each run the png files will be overwritten, so if you wish to save them separately I recommend making a copy of them. Additionally, feel free to experiment with the number of nodes and iterations. These variables are located near the top of the run function in main.rs.

Output Code:

Example Train Results



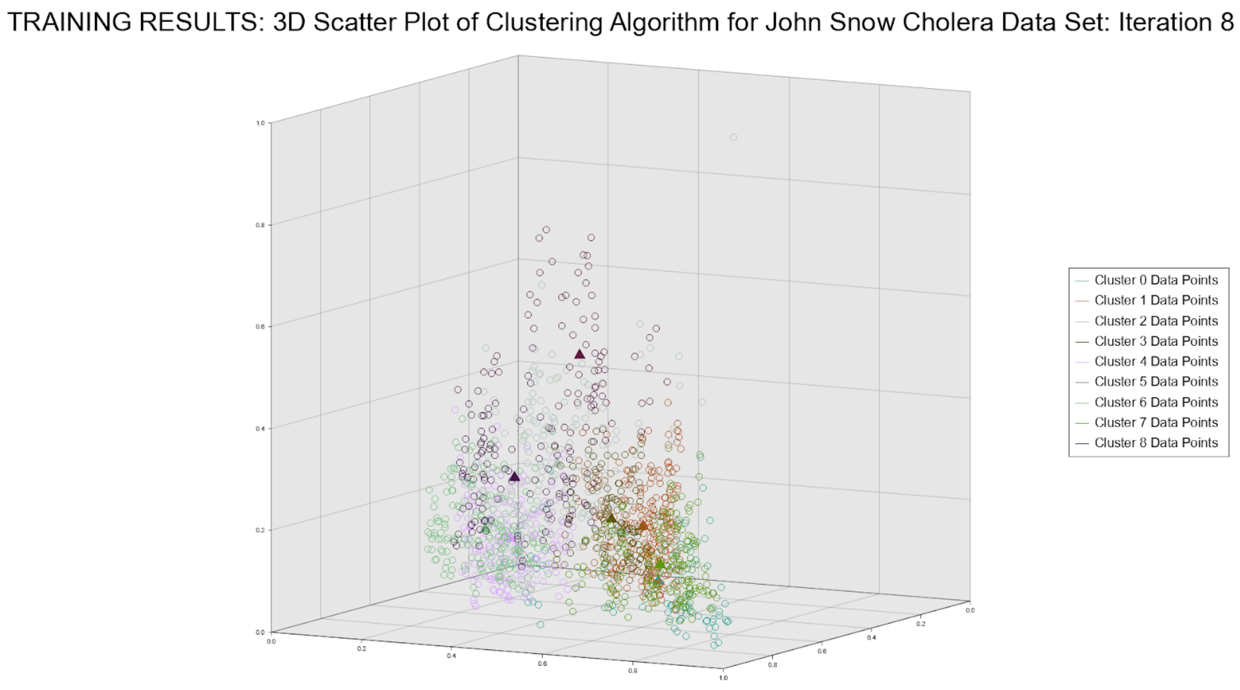
```
Clusters for Training Iteration 6:
Cluster: 0, Nodes: 147
Cluster: 1, Nodes: 144
Cluster: 2, Nodes: 174
Cluster: 3, Nodes: 160
Cluster: 4, Nodes: 79
Cluster: 5, Nodes: 112
Cluster: 6, Nodes: 163
Cluster: 7, Nodes: 109
Cluster: 8, Nodes: 301
Silhouette Score for Iteration 6: 0.7222391238247553
```



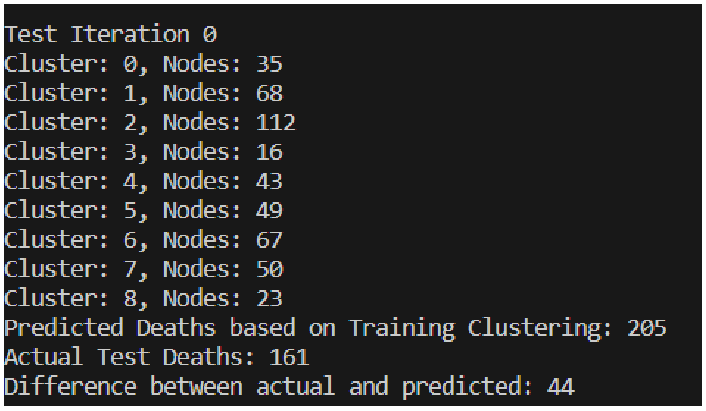
```
===============================================
TRAINING RESULTS:

BEST ITERATION: 2
 Silhouette Score of 0.7489
Average deaths associated with each cluster: [(0, 0.02631578947368421), (1, 0.2875816993464052), (2, 0.7702702702702703), (3, 0.30735930735930733), (4, 0.18803418803418803), (5, 0.07978723
40425532), (6, 1.1167883211678833), (7, 0.011428571428571429), (8, 0.4817073170731707)]
Check Project Folder for TrainingResults.png of the best iteration
```

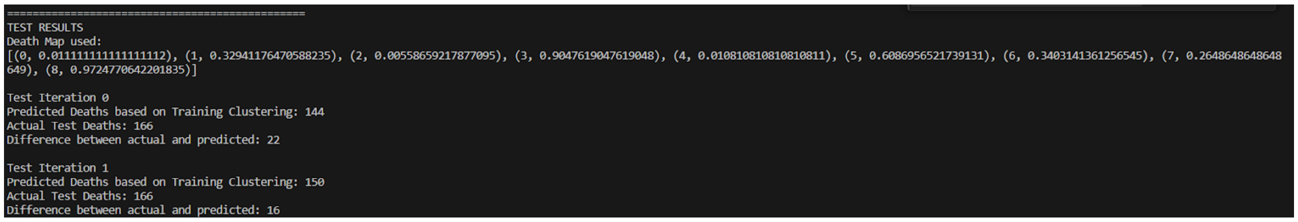
Example Train Graph:



Example Test Results



```
Test Iteration 0
Cluster: 0, Nodes: 35
Cluster: 1, Nodes: 68
Cluster: 2, Nodes: 112
Cluster: 3, Nodes: 16
Cluster: 4, Nodes: 43
Cluster: 5, Nodes: 49
Cluster: 6, Nodes: 67
Cluster: 7, Nodes: 50
Cluster: 8, Nodes: 23
Predicted Deaths based on Training Clustering: 205
Actual Test Deaths: 161
Difference between actual and predicted: 44
```

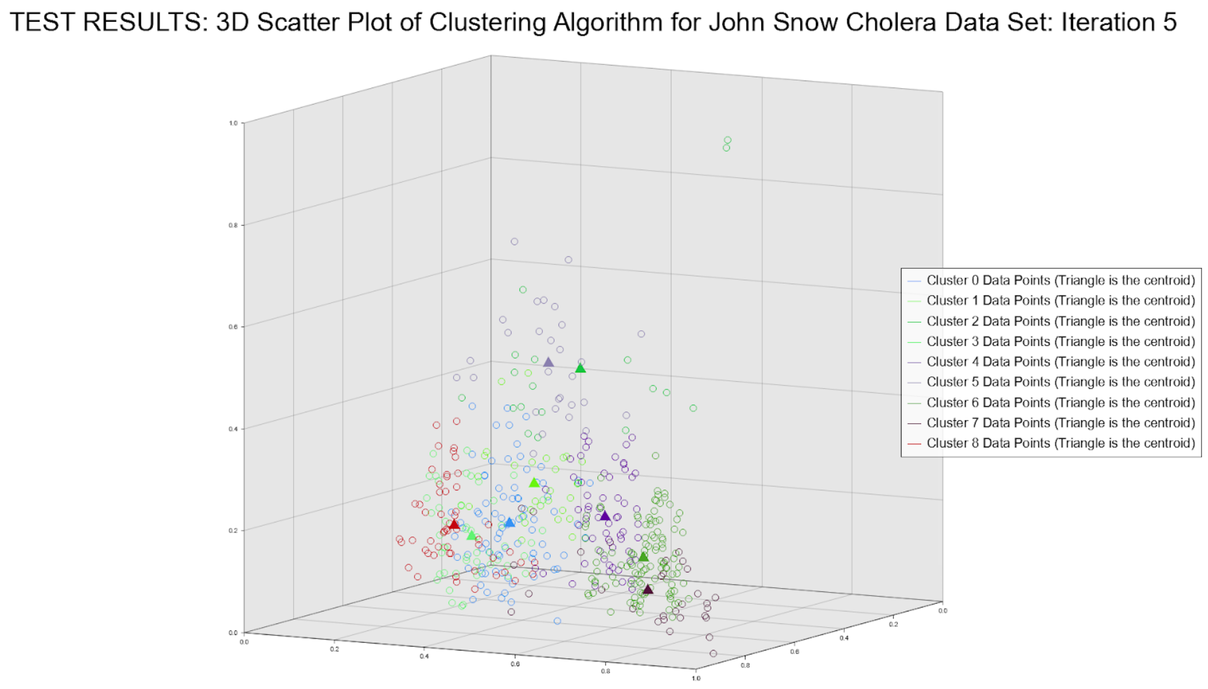


```
===============================================
TEST RESULTS
Death Map used:
[(0, 0.011111111111111112), (1, 0.32941176470588235), (2, 0.00558659217877095), (3, 0.9047619047619048), (4, 0.010810810810810811), (5, 0.6086956521739131), (6, 0.3403141361256545), (7, 0.2648648648648
649), (8, 0.9724770642201835)]

Test Iteration 0
Predicted Deaths based on Training Clustering: 144
Actual Test Deaths: 166
Difference between actual and predicted: 22

Test Iteration 1
Predicted Deaths based on Training Clustering: 150
Actual Test Deaths: 166
Difference between actual and predicted: 16
```

Example Test Graph:



Figures

Figure 1:

Min Max Scaling: $X\_scaled = (X - X\_min) / (X\_max - X\_min)$

Source:

https://jonmce.medium.com/min-max-your-way-through-pandas-data-frames-18dc40b6b12f

Figure 2:

Silhouette Score: $S(i) = (b(i) - a(i)) / max\{a(i), b(i)\}$

Source: https://tushar-joshi-89.medium.com/silhouette-score-a9f7d8d78f29