PubMed Clustering Project

Contents

1. Clustering algorithm

1.1 Pre-processing

Reading input: The data is read and put in the right data structure using the function def ReadData(filename) where the file is opened and is split according to the whitespaces. Then only the first column is added to the list because it represents the gold id.

Creating input data: In the training set, a HashMap is created where the key is the gold id which is the first column and the columns other than the first are appended as value. This is done in the function def label(filename) where the input is the file.

Vectorization: In the function def truthTable, a set of unique labels are made, fetching the labels from the HashMap created in above step. There a HashMap true_pred is made, where the key is the gold id and the value is the true label which is the index of the disease in the set.

Example: Triple negative breast cancer becomes Triplenegativebreastcancer and subsequently all the diseases are added and made to a set with unique disease name.

Normalization: The ids are summed up and all the ids are normalized using the sum. It helps in better visualization of the result and to normalize the feature.

1.2 Algorithm used

K means algorithm was employed for the problem. The following reasons for choosing the algorithm:

1. K means is the simplest algorithm. It would work perfectly for a small dataset that we are dealing with.
2. We have an idea of K as we know the training dataset, so it is better to employ k means rather than hierarchical clustering.
3. The algorithm produces tighter clusters than hierarchical clusters.
4. The algorithm is fast and efficient.

5. The clustering results are easier to understand.

1.3 Parameters

1. **K (Number of clusters)**: The number of clusters are chosen randomly or can be decided based on the data. It can also be decided using silhouette analysis.
2. **Choosing centroid**: It can be considered randomly. We decided it randomly from a uniform distribution.
3. **Stopping criteria**: It is decided if the centroids no longer changing or max iteration is reached, we stop the algorithm.
4. **Max iteration:** If the algorithm is implemented efficiently the number of iterations does not matter because after a point of time, the centroid will get stuck. That is why we implemented the stopping criteria. Thus, our algorithm has that advantage of hard convergence.

1.4 Similarity metric

1. **Euclidean distance**: K-Means is based on pairwise Euclidean distances between data points, because the sum of squared deviations from centroid is equal to the sum of pairwise squared Euclidean distances divided by the number of points**.**

2. **Manhattan distance**: The Manhattan distance is based on absolute value distance. Absolute value distance should give more robust results, as opposed to Euclidean distance that would be influenced by unusual values.

3. **Jaccard similarity**: The Jaccard coefficient measures similarity between sample sets
4. **Cosine similarity**: Cosine similarity takes the dot product of the vector.

1.5 Quantitative measure

In order to quantitatively measure how our algorithm performs on the dataset, we devised a simple way. We perform kmeans on the validation/testing dataset. Our aim is to cluster the same ids in the same cluster. The index of the cluster does not matter. The aim is to cluster the same ids together. So, to observe how many ids are clustered together. We define a function 'quant' which creates tuples of the length of elements in a cluster. Since, the which id goes to which cluster cannot be determined since it is an unsupervised task. We can just see if the same ids are clustered together. We also plot the true_pred id vs the cluster as well as the predicted vs cluster for visualization.

1.6 Complexity of the code

Calculation of distances: To calculate the distance from a point to the centroid, we can use the squared Euclidean function. We perform two subtractions, one summation, two multiplications and one square operation.

Comparisons between distances

Calculation of centroids

So, the total number of operations for an iteration
= distance calculation operations + comparison operations + centroids calculation operations
= 6*[k*m*n] operations + [(k-1) *m*n] operations + [k*((m-1) + 1) *n] operations

If the algorithm converges after I iterations, then total number of operations for the algorithm
= 6*[I*k*m*n] operations + [I*(k-1) *m*n] operations + [I*k*((m-1) + 1) *n]
= [6Ikmn + I(k-1) mn + Ikmn] operations
≈ O (I*k*m*n)

1.7 Result

Parameters	Grouping
1. Similarity index: Euclidean, K = 6	{0: (32, 33), 1: (6, 6), 2: (16, 24), 3: (23, 30), 4: (17, 2), 5: (9, 8)}

Max_iteration: 10	
Similarity index: Euclidean, K = 6 Max_iteration: 100	{0: (9, 35), 1: (32, 3), 2: (6, 22), 3: (16, 36), 4: (17, 5), 5: (23, 2)}
Similarity index: Euclidean, K = 4 Max_iteration: 10	{0: (32, 2), 1: (23, 8), 2: (17, 50), 3: (6, 43)}
Similarity index: Euclidean, K = 4 Max_iteration: 100	{0: (23, 23), 1: (9, 37), 2: (32, 9), 3: (16, 34)}
Similarity index: Manhattan K = 6 Max_iteration: 100	{0: (9, 50), 1: (17, 6), 2: (32, 2), 3: (23, 2), 4: (16, 43), 5: (6, 0)}
Similarity index: Jaccard K = 6 Max_iteration: 100	{0: (9, 103), 1: (23, 0), 2: (17, 0), 3: (32, 0), 4: (16, 0), 5: (6, 0)}
Similarity index: Cosine K = 6 Max_iteration: 100	{0: (23, 103), 1: (6, 0), 2: (32, 0), 3: (16, 0), 4: (9, 0), 5: (17, 0)}

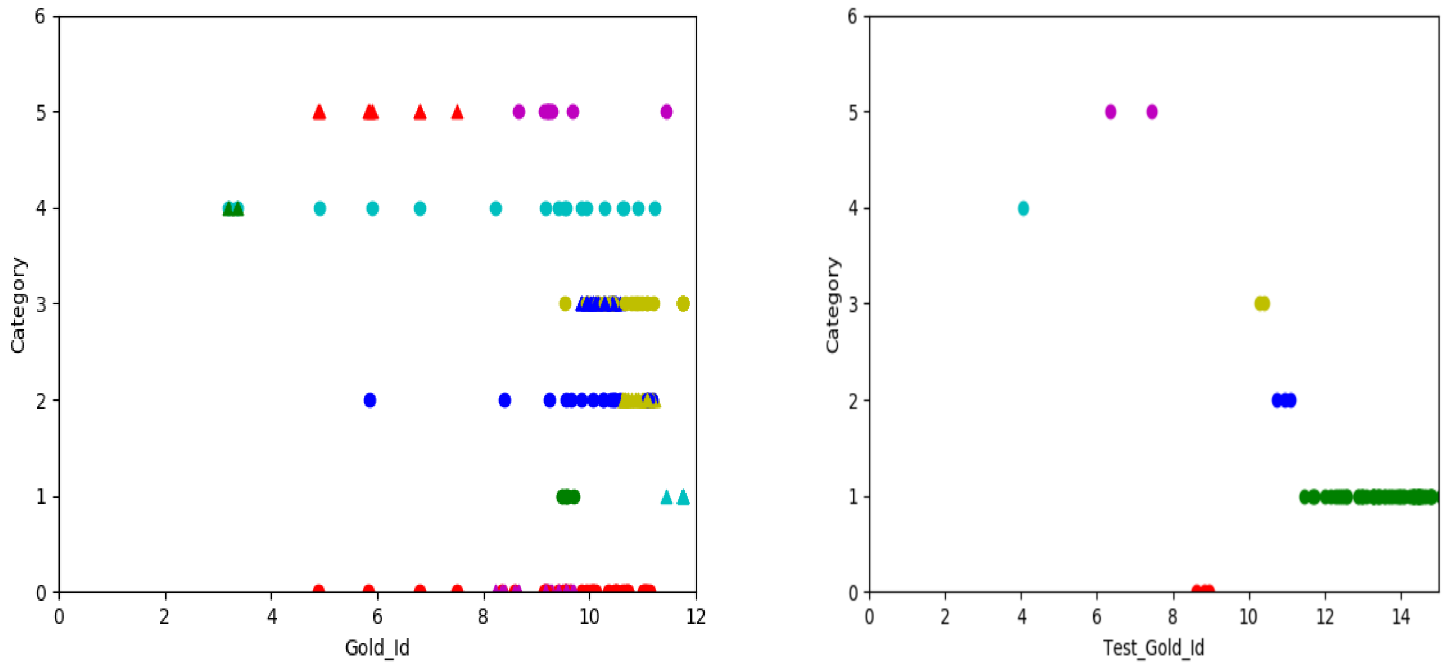

Figure 1Similarity index: Euclidean, K =6, Iteration = 10

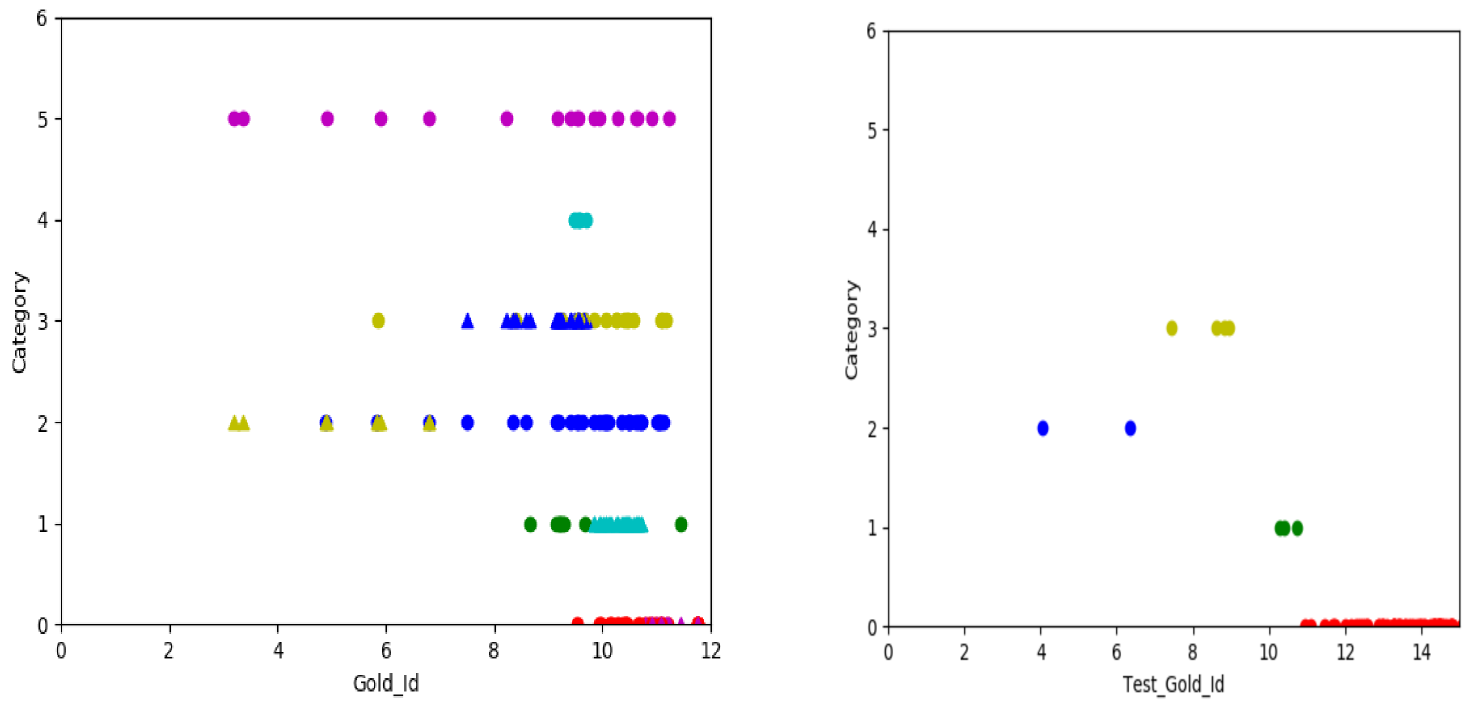

Figure 2 Similarity index: Euclidean, K =4, Iteration = 100

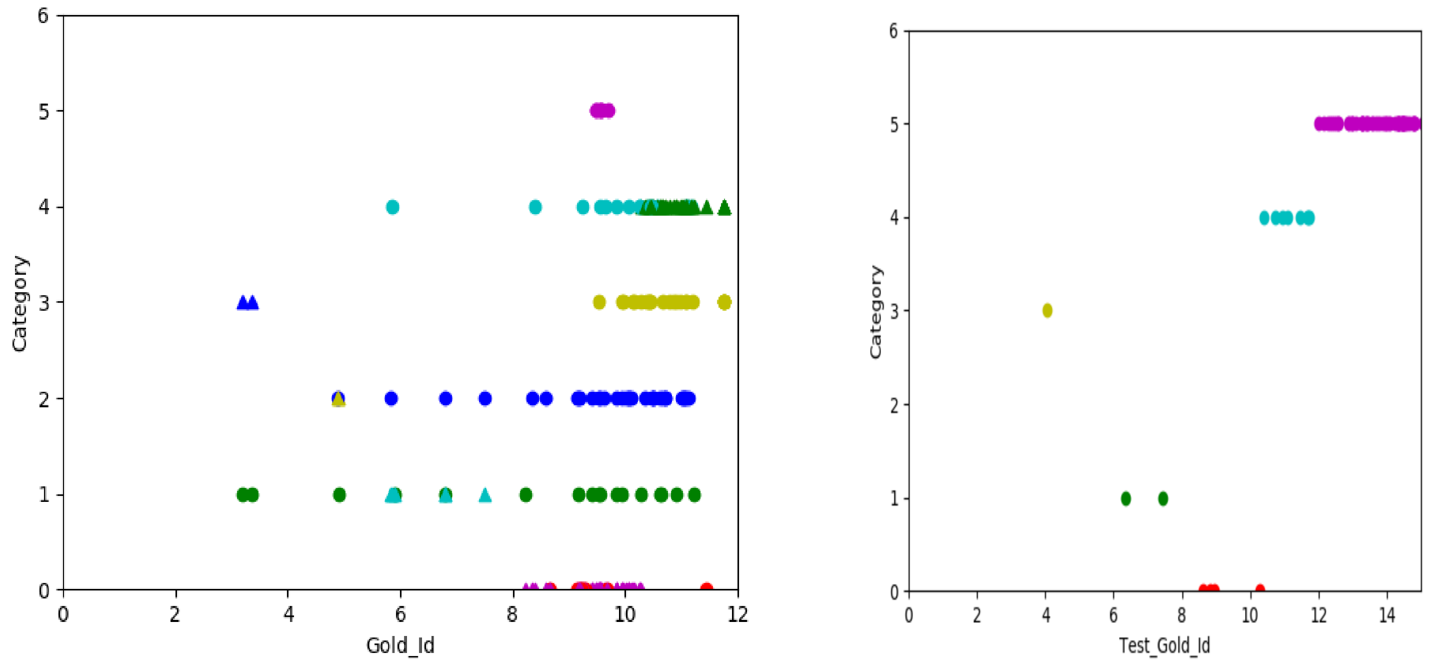

Figure 3Similarity index: Manhattan, K =6, Iteration = 100

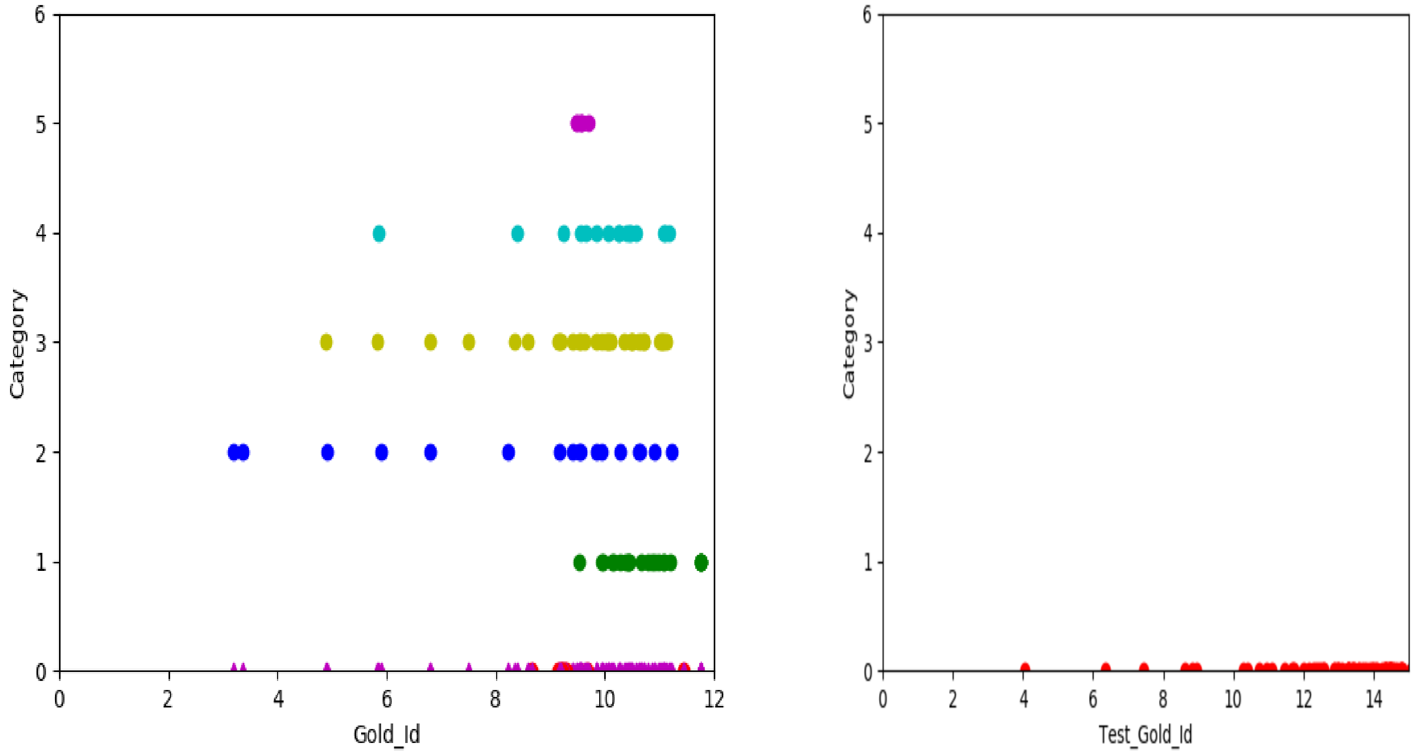

Figure 4 Similarity index: Jaccard, K =6, Iteration = 100

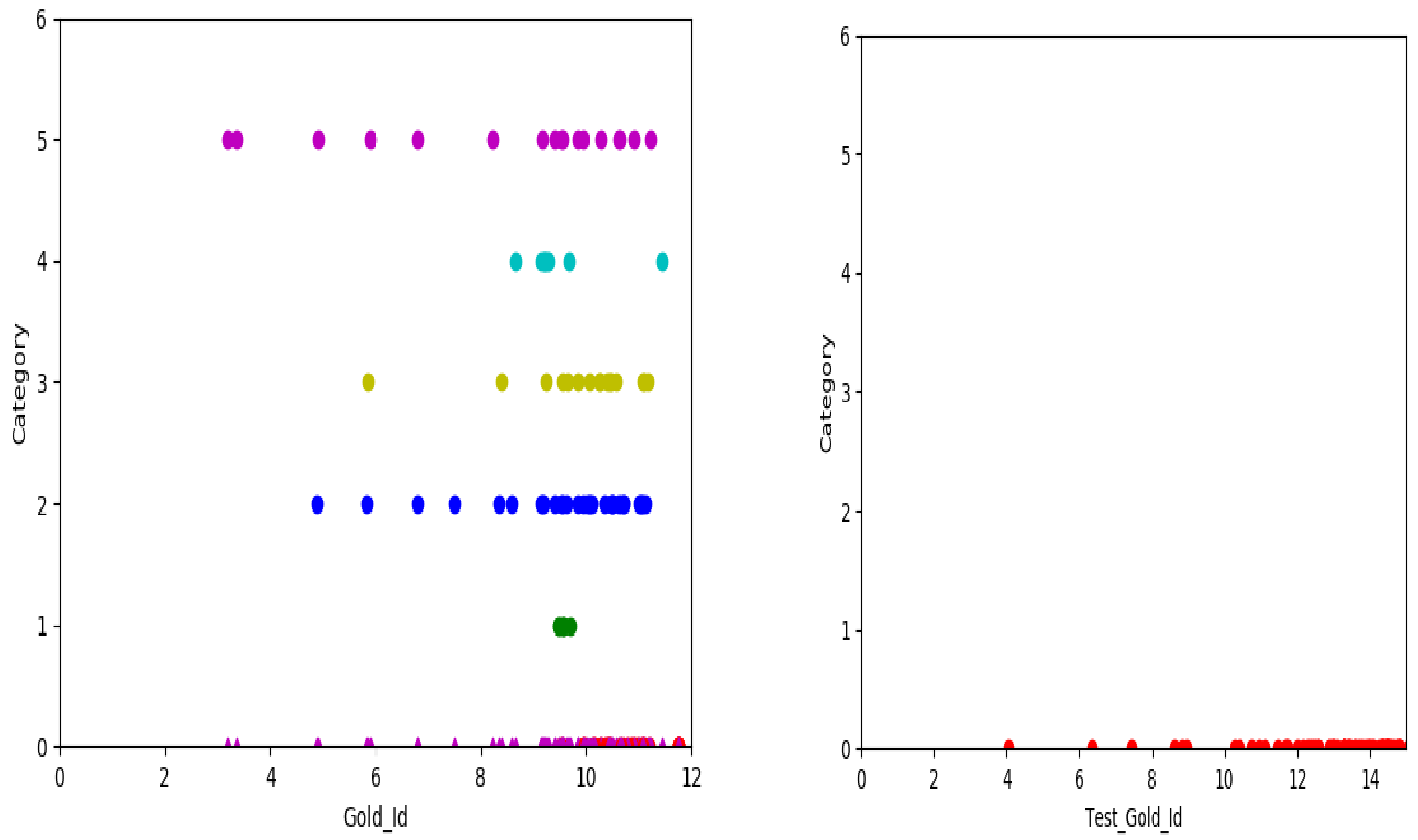

Figure 5 Similarity index: Cosine K =6, Iteration = 100

1.8 Discussion

1. We tested different values of K since observing the training data, we had 6 different classes based on observation. We noticed that the training and validation set performs well for K =6. But the testing data has a bias towards one class. Thus, proving that K =6 might be a more clusters being made than needed for the data. Even for K=4, we see that bias of the data for one class stating that for our test data K<4 is an optimal choice.
2. We tested different similarity index. Euclidean and Manhattan performed well. Jaccard and cosine similarity did not perform well. They classified all the dataset in one cluster. Thus, for our data Euclidean would be the best choice for similarity index function.

3. In case of number of iteration, we noticed that with 10 was not able to cluster well. But, 100 number of max iteration for our data is an optimal choice.
4. We also noticed that outliers in case of Kmeans affects the result.
5. Normalization of the data was found out to be beneficial. It puts the data within a range that also helped in deciding the initial centroid values.