INDIAN INSTITUTE OF TECHNOLOGY DELHI

COL761
Data Mining
Assignment 2

Abstract

In this assignment, we experiment with frequent subgraph mining, subgraph isomorphism and k-means clustering.

Aniket Gupta 2019CS10327

Devanshi Khatsuriya 2019CS10344

Prabhakar Chaudhary 2019CS10381

Contents

1 Q1

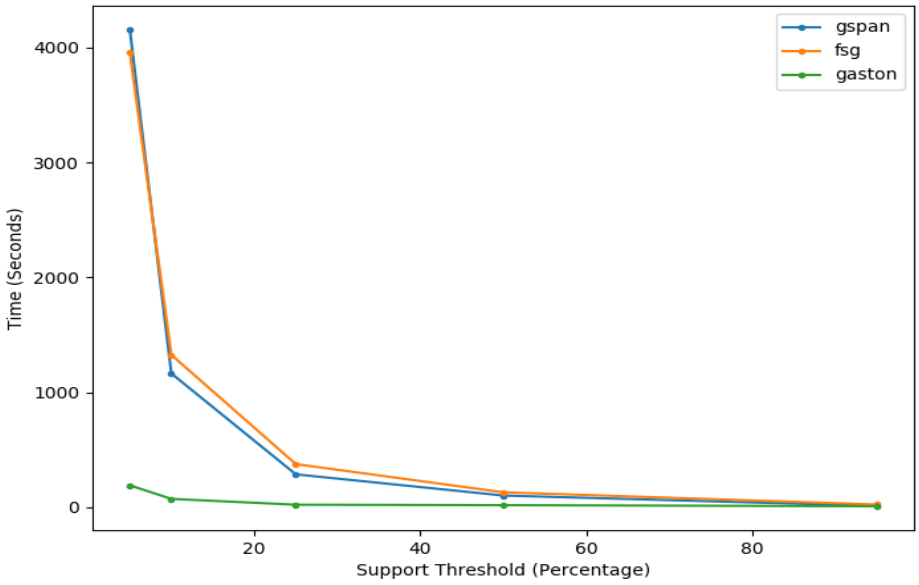

Figure 1: Timings of Different Libraries on Yeast

Trends Observed:

1. It can be seen from the above plot that Gaston performs much better than the other two methods.

2. The runtime increases for all the three methods as we decrease the threshold. This is because of exponential increase in the number of frequent subgraph patterns with decreasing threshold.

3. For the higher values of threshold, all the three methods takes similar time since the number of frequent patterns become very low for large thresholds.

4. The gap in the runtimes between Gaston and other methods increases exponentially on decreasing the threshold value.

5. gSpan performs marginally better than FSG on some values of threshold for the given dataset. Generally, gSpan performs better than FSG in terms of runtime. One possible reason for their comparable times on this dataset can be the size of frequent patterns observed. The given dataset is based on chemical compounds. Thus, the average size of of frequent patterns is not very large (less than 10 for threshold=25%) and the candidate generation step in FSG runs for small number of times. Candidate generation step is one of the distinguishing steps between FSG and gSpan that create

large gap between their time. Since this step runs for smaller number of times on this dataset, there is not much difference observed in the time taken by the two methods.

1. **Gaston**:

   - We see that Gaston always does better and does a lot better for smaller thresholds like 5 and 10. This must be because of they way Gaston splits frequent subgraph mining into phases for frequent paths, trees and graphs to optimize runtime for molecular databases. They discovered that the largest number of frequent substructures in molecular databases are actually free trees and so they handle them separately.
   - They consider only necessary legs - which are those such that, among all descendants of the current graph code C_0 , there is at least one canonical graph code which can only be obtained by applying the refinement defined by that leg. They do this check of necessary legs for free trees efficiently to avoid the canonical label check for generated descendent.
   - They consider the cycle closing refinements in the very last phase.
   - They make use of Embedding Lists that improve time (but increase memory usage).

2. **gSpan**:

   - For the given dataset, gSpan performs only slightly better than FSG.
   - gSpan is based on pattern growth (depth first) approach and builds a lexicographic ordering among the subgraphs.
   - It maps each pattern to a unique DFS code and gives it a canonical label.
   - gSpan performs better than FSG in general because it doesn't create large number of candidates like FSG during each iteration and also reduces the time by pruning repetitive patterns by using the canonical labels.

3. **FSG (PAFI)**:

   - FSG uses join-based (breadth first) approach. It involves candidate generation, pruning and frequency counting.
   - FSG takes longest time among these three methods. It gives performace comparable to gSpan on the given dataset due to the reasons mentioned earlier.
   - It takes large amount of time in candidate generation by join-based approach. Determining isomorphic cores by subgraph isomorphism takes large time.
   - Pruning uses graph isomorphism and frequency counting of candidates uses subgraph isomorphism which are expensive processes. Thus, FSG takes longest time among the three methods we tested.

2 Q2

We make use of vf2 library for subgraph isomorphism checks.

1. We experimentally determine the value of support threshold at which frequent subgraphs should be mined from the database. We use the sample query file given as the data for this. The timing data is shown below.

   Varying Support Threshold (choosing all obtained graphs)

threshold	no. of graphs	m	type	time in index creation	time on sample
0.1	8555	8555	all	29m 15s	94s
0.2	1773	1773	all	8m 33s	19s
0.3	730	730	all	7m 23s	10s
0.35	514	514	all	8m 30s	9s
0.4	376	739	all	7m 39s	9s
0.45	282	282	all	4m 58s	9s
0.5	212	212	all	6m 4s	15s
0.6	108	108	all	7m 7s	75s
0.7	61	61	all	4m 51s	217s
0.8	18	18	all	4m 36s	381s
0.9	5	5	all	4m 94s	701s

2. Theoretically, there is a valley in the runtime v/s support threshold plot because there is a trade-off between number of subgraph isomorphism checks between query graph and database graphs and the feature vector generation for the query graph and mining on increasing feature size. On increasing feature size, there will be more pruning and lesser isomorphism checks required, while on decreasing it, we can save time in feature vector generation and comparison.

3. This trade-off actually exists because the subgraph isomorphism checks are quite cheap (because of the query graph being small) and hence comparable to feature vector matching time.

4. After choosing a value of the support threshold, we found it best to choose all the graphs available at that threshold. This is because higher m for a fixed threshold leads to more pruning and hence lesser isomorphism checks. This can be seen in the table below, which is query time vs m for a fixed threshold.

   varying m (keeping support threshold fixed at 0.4)

m	threshold	type	time in index creation	time on sample
50	0.4	middle sized	9m 21s	167s
100	0.4	middle sized	9m 9s	160s
150	0.4	middle sized	9m 25s	61s
200	0.4	middle sized	9m 56s	146s

250	0.4	middle sized	9m 50s	43s
300	0.4	middle sized	9m 27s	9s
350	0.4	middle sized	10m 15s	9s

5. In the table above, while choosing m plots, we choose the m middle sized plots. This is because smaller size graphs and larger sized patterns will be more likely and less likely to occur respectively and hence will lead to lesser pruning when comparing feature vectors for query and database graph.

3 Q3

After analyzing the elbow plots below, we can say that the optimal values for k for dimension 2, 3 and 4 datasets are 3, 5 and 9 respectively.
We use the logic that the optimal value of k is the transition point the plot of intra-cluster distance becomes parallel to x-axis, which indicates that there is no further (considerable) improvement in intra-cluster distances on further increasing the number of cluster heads.

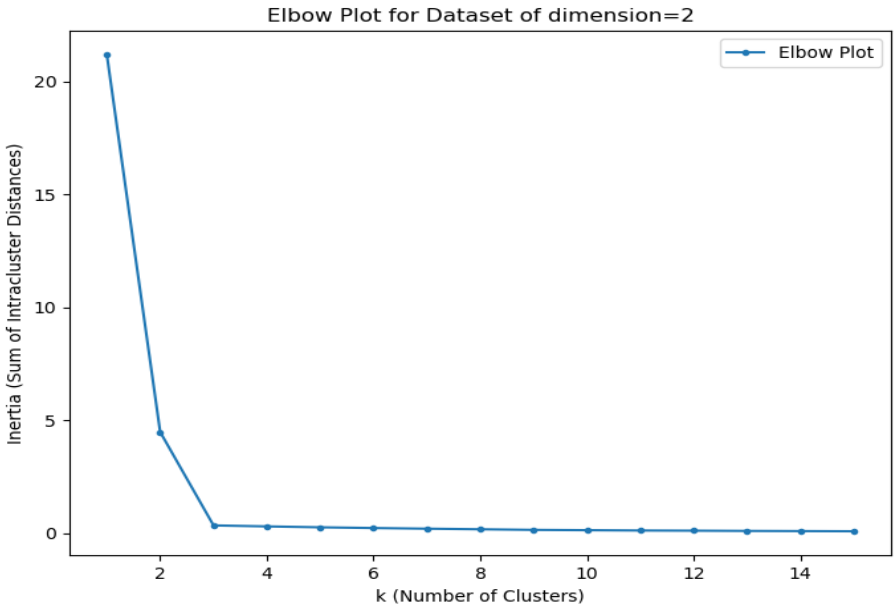

Figure 2: Elbow Plot for dim=2 Dataset

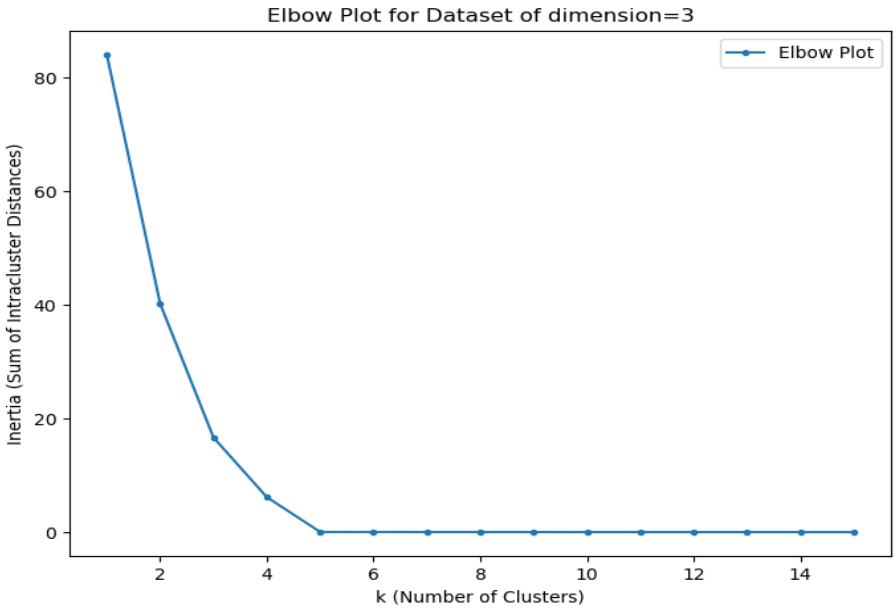

Figure 3: Elbow Plot for dim=3 Dataset

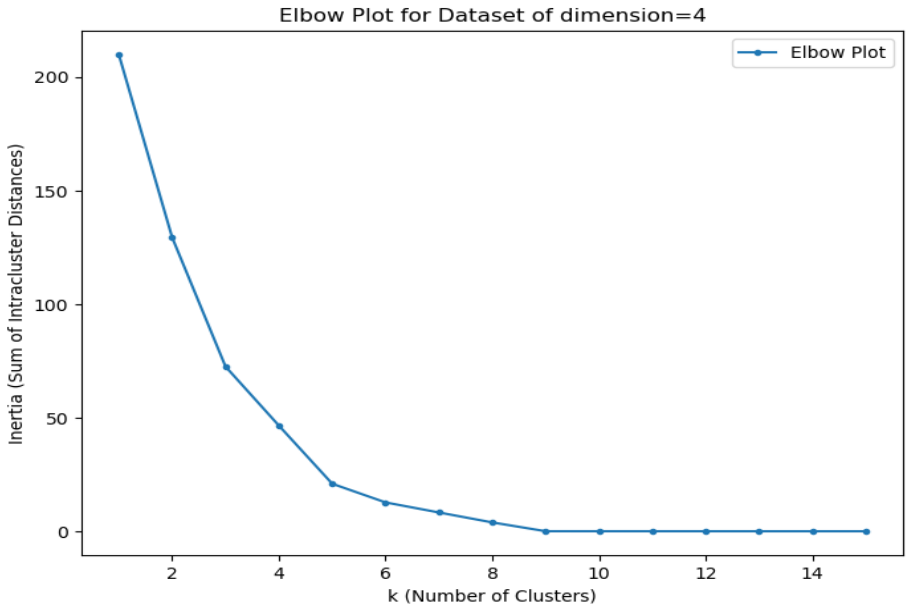

Figure 4: Elbow Plot for dim=4 Dataset