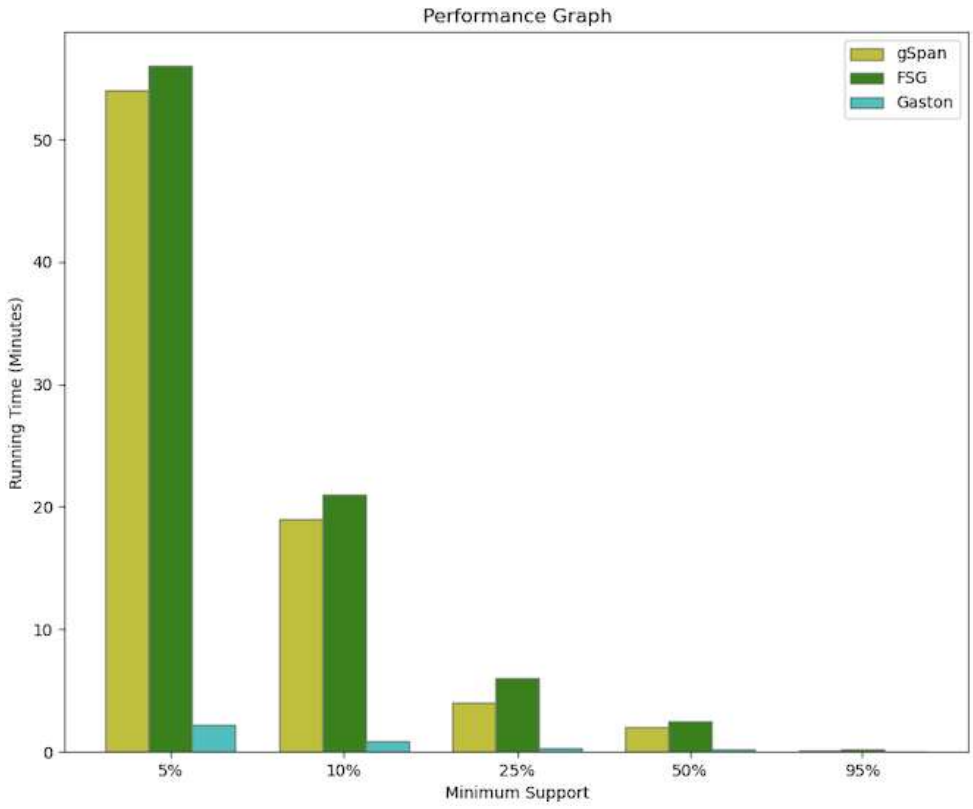
COL761 | DATA MINING
ASSIGNMENT 2

Aniket Shetty (2018EE10443)
Dipen Kumar (2018CS50098)
Umesh Parmar (2018CS50424)

Q1

We used three subgraph mining tools, gSpan, FSG, and Gaston. We tested them on yeast dataset for varying supports and plotted their runtime. Our observations are—

1. The running time increases exponentially with decrease in support threshold. Below plot shows decrease in running time with increase in support is proportional to running time at any point.
2. FSG is slower among all because of BFS approach.
3. gSpan is faster than FSG because it follows DFS nature.
4. Gaston is the fastest and optimized even at lower threshold values among all.



We explain above mentioned statements with following reasonings—

1. Low support means we allow more subgraphs being mined, and this increases exponentially as seen in most distributions.
2. FSG is slower because of the problems caused by its breadth-first Apriori level-wise approach via huge candidate set generation, multiple database scans, difficulties at mining long patterns. These problems are further aggravated by the graph nature of the data. This leads to highest growth rate in runtime with decreasing support threshold for FSG.
3. gSpan uses the subgraph isomorphism test and frequent subgraph growth together into a single procedure which drastically increases the mining process. The hierarchical nature of DFS-code; the pruning of non-minimal to avoid useless subgraph isomorphism tests; and the added restrictions to the addition of new nodes to DFS-code-tree optimize is compared to FSM.
4. Gaston breaks the entire task into three subtasks: searching first for frequent paths, then frequent free trees, and cyclic graphs. there are efficient ways to enumerate paths and trees. It considers fragments that are paths or trees first, and only proceeding to general graphs with cycles at the end.

Q2.

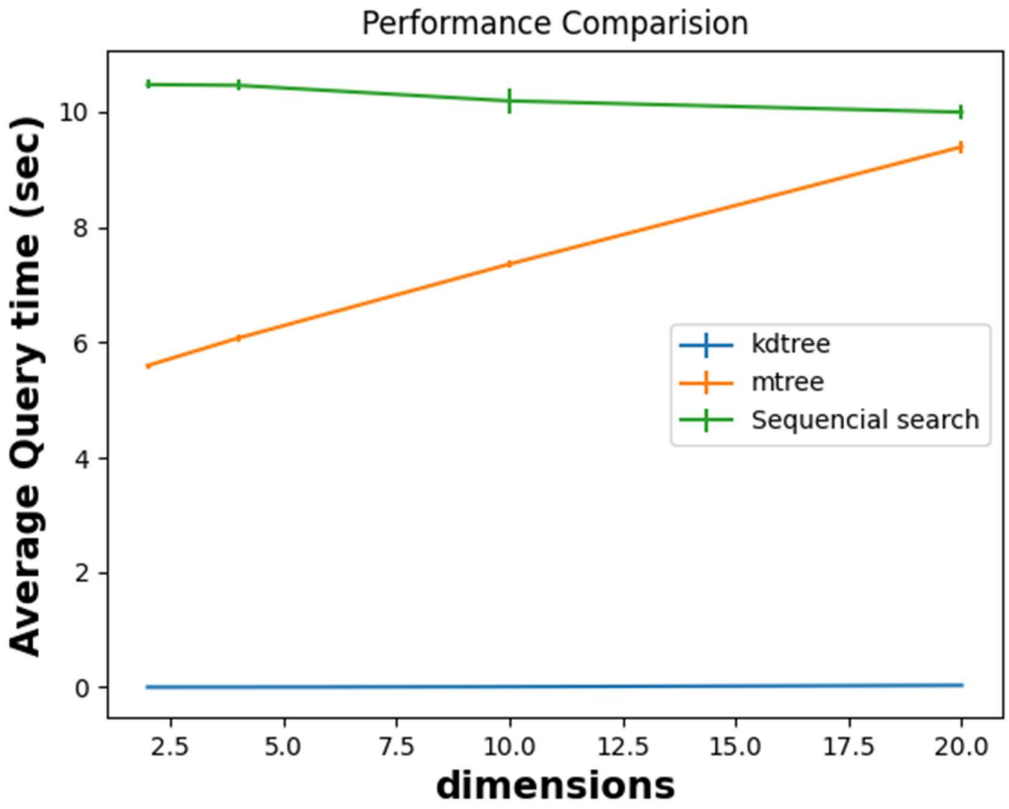
In this we have find KNN (k=5) using kdtree, mtree and sequential search and compare there average query time.

1. Library sklearn.decomposition.PCA is used to reduce the dimension of given data using PCA.
2. We used the library sklearn.neighbors.kdtree for Kdtree.
3. We use the link (https://github.com/tburette/mtree/blob/master/mtree.py) for mtree.
4. The Pseudocode for KNN Query for kdtree and mtree is written is algorithm.txt (present inside Q2 folder).

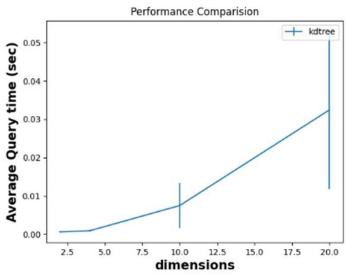
Query time mean and error on image_data.dat (time is in seconds)

```
Dimension = [     2       ,      4        ,     10        ,     20      ]
kd_mean =   [  0.000613,  0.000911,  0.007481,  0.032441]
kd_std =    [  0.000151,  0.000230,  0.005818,  0.020680]
seq_mean= [10.472132, 10.458899, 10.188722,  9.993480]
seq_std =   [  0.077511,  0.088353,  0.211762,  0.124159]
m_mean =    [  5.596427,  6.073996,  7.357527,  9.391846]
m_std =     [  0.048564,  0.057250,  0.068165,  0.112944]
```

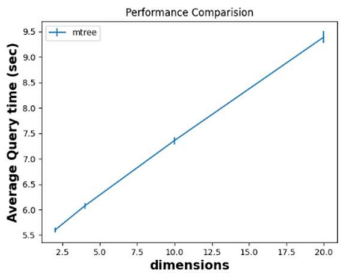
Graphs:



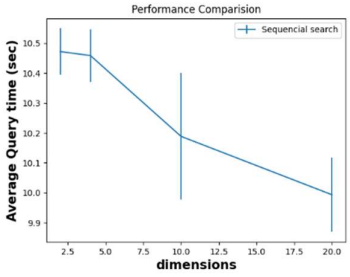
In the above graph the error bar are quite small because error is in the range of millisecond whereas average query time is in the range of seconds.
Below is the are individual plots to Cleary see the error bars.



KD-tree



M-tree



Sequential search

Obseravtions:

1. Kdtree performs better than mtree and both are much Faster than Sequential search
2. At higher dimensions, the mtree can potentially perform worsethan sequential search.