MDM Ex3

November 7, 2020

1 Methods of Data Mining: Exercise 3

1.1 Ahmed Bin Shafaat 795933

2 Exercise 1

2.1 a)

Let us first define all these measures so that we can interpret results later in a more precise manner.

Node Degree: Number of direct neighbours is referred to as degree.

Label	Degree
1477	43
1443	43
1457	42
1502	42
1563	41
1452	41
1428	41
1458	40

Weighed Degree: It is simply the node degree with a weight assigned to the edge between two nodes determining how strong the connection is.

Label	Degree	Weighted Degree
1437	39	221.000003
1563	41	216.800002
1457	42	186.800001
1458	40	183.799999
1452	41	171.800002
1477	43	165.2
1498	40	164.599999
1480	40	161.8

Closeness Centrality: Closeness Centrality referes to the mean distance from one node to all other node using **Geodesic path i.e. shortest distance** between two nodes.

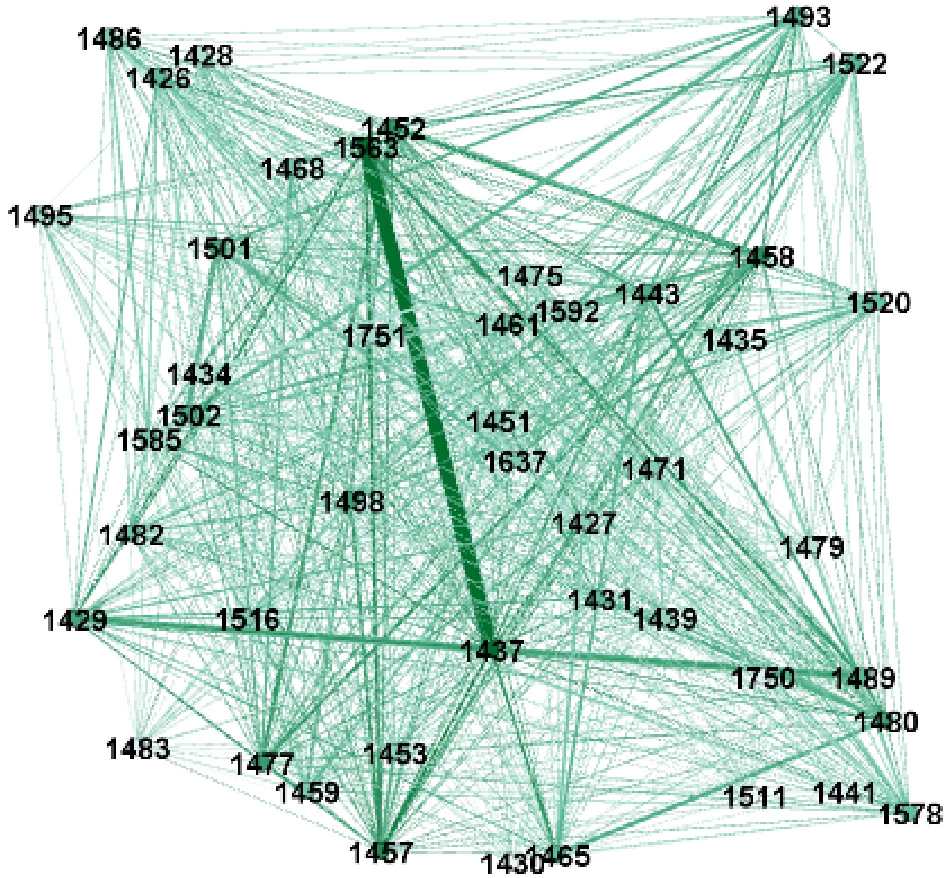
Label	Closeness Centrality
1477	0.957447
1443	0.957447
1502	0.9375
1457	0.9375
1452	0.918367
1428	0.918367
1563	0.918367
1459	0.9

Betweenness Centrality: Betweenness Centrality is a measure of how often a node is a **bridge** between other nodes.

Label	Betweenness Centrality
1443	10.267852
1477	9.288583
1502	8.774473
1457	8.27553
1563	8.193991
1480	7.875956
1522	7.730772
1585	7.57164

We chose to select 8 nodes having highest values for given metrices and reported them. We can see that nodes **1477,1443,1457,1502,1563, 1452,1428,1458** have highest degree meaning they have highest number of neighbours. So they are most influential in terms of degree.

For weighted degree, we can see that nodes **1437,1563,1457,1458,1452,1477,1498,1480** have the highest values meaning they have the strongest interaction among them. I have visualized the nodes in terms of weighted degree below and we can see that **1437** and **1563** have spent most time with eachother than other nodes.



Speaking in terms of closeness centrality, **1477,1443,1502,1457,1452,1428,1563,1459** have the highest closeness centrality measure values.

For betweenness centrality, **1443,1477,1502,1457,1563,1480,1522,1585** are most likely to behave as a connection/bridge between other nodes.

2.2 b)

After runnign modularity and Girvan-Newman clustering, we can identify three clusters/communities in identified by modularity technique and Gilvan-Newman yields 24 communities.

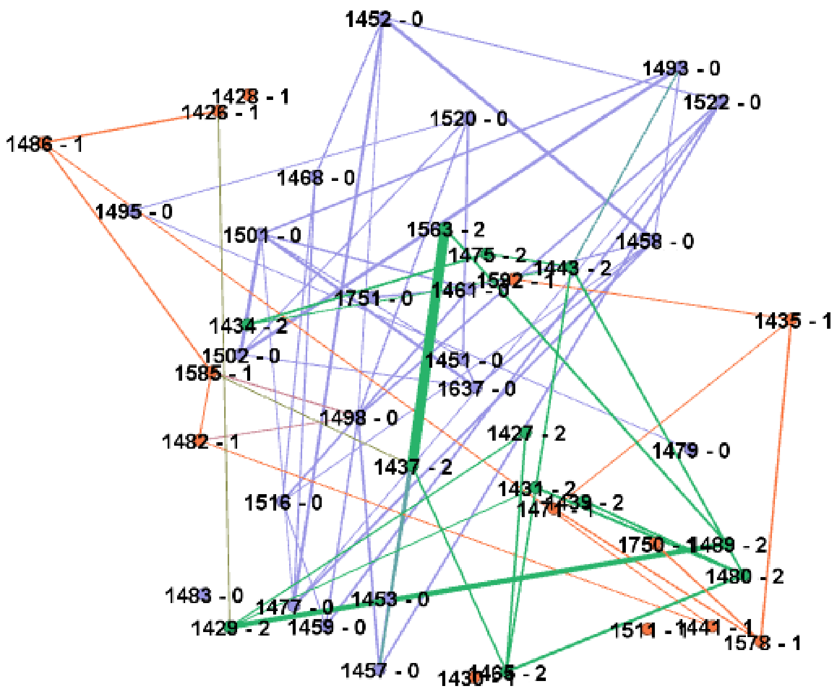
Modularity produces three communities, one of them is bigger having 45.65% of the total nodes while other two communites have 26% and 28% population.

Gilvan-Newman clustering identifies 24 communities having Maximum found modularity of 0.027242184. One of the communities is biggest of them all having 50% of the total nodes while other 23 communites have 2.17% of the total nodes in each.

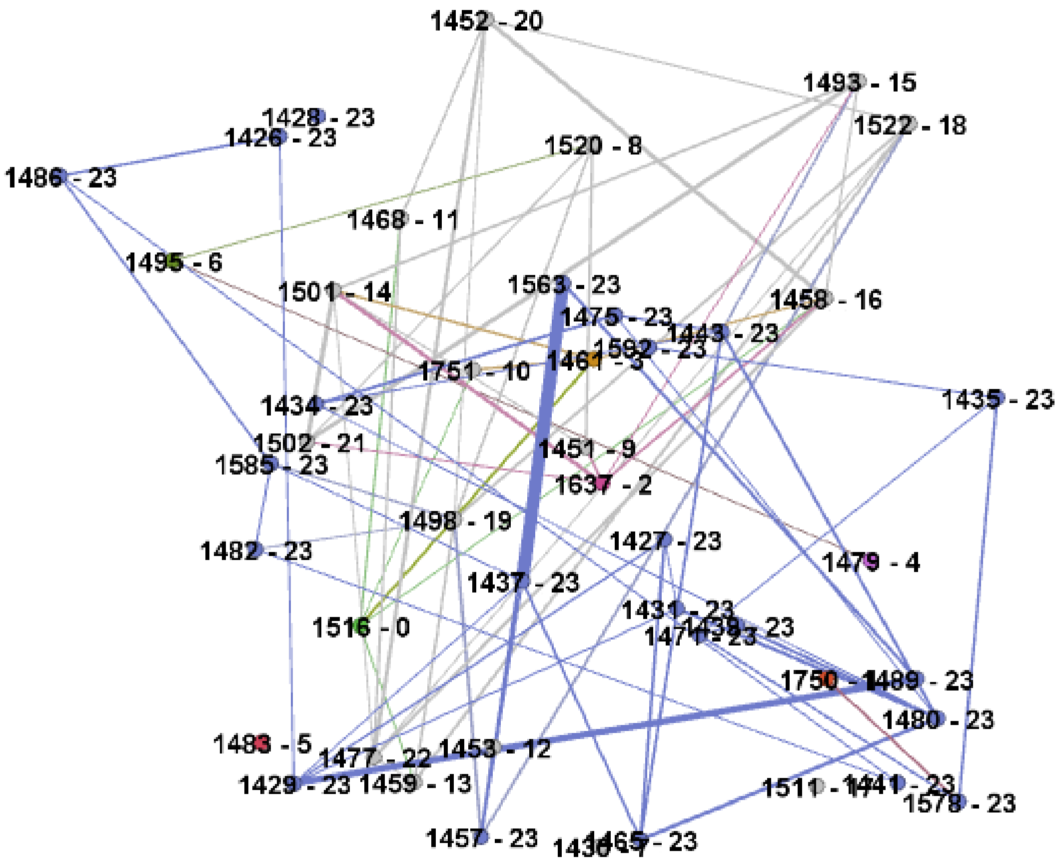
Speaking of Modularity clustering, after going through the metadata file, I infer that nodes belonging to bigger cluster are in section 5A and nodes in other two clusters are in 5B so most probably clustering is done on the basis of section data.

2.3 c)

Communities identified by Modularity

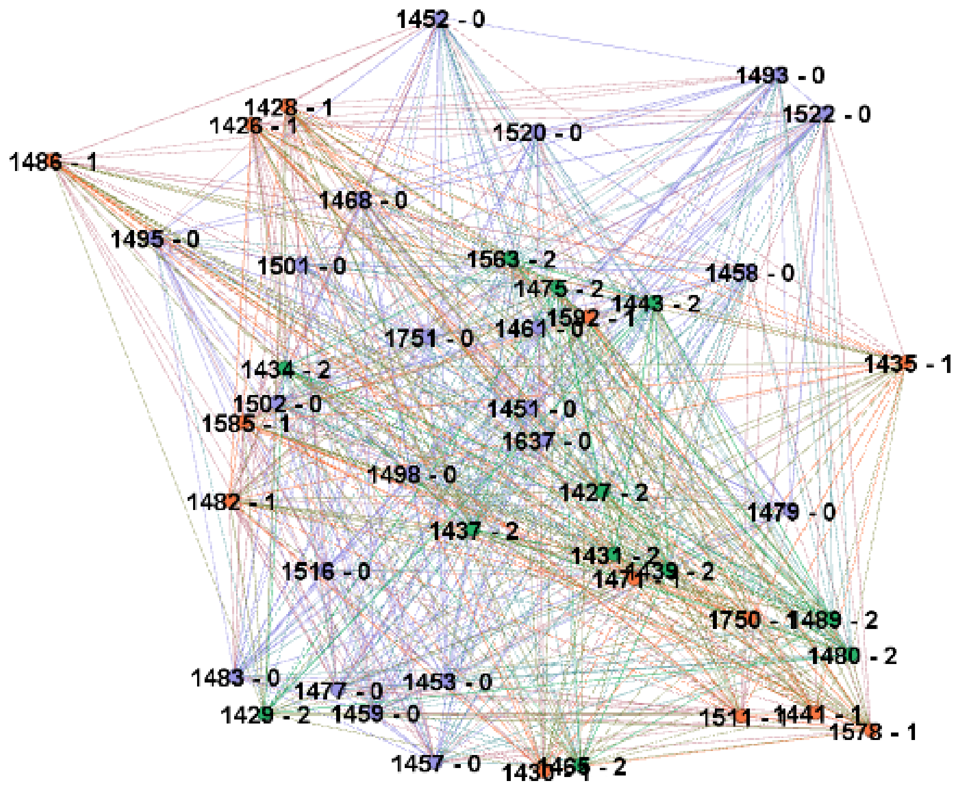


Communities identified by Gilvan-Newman Clustering

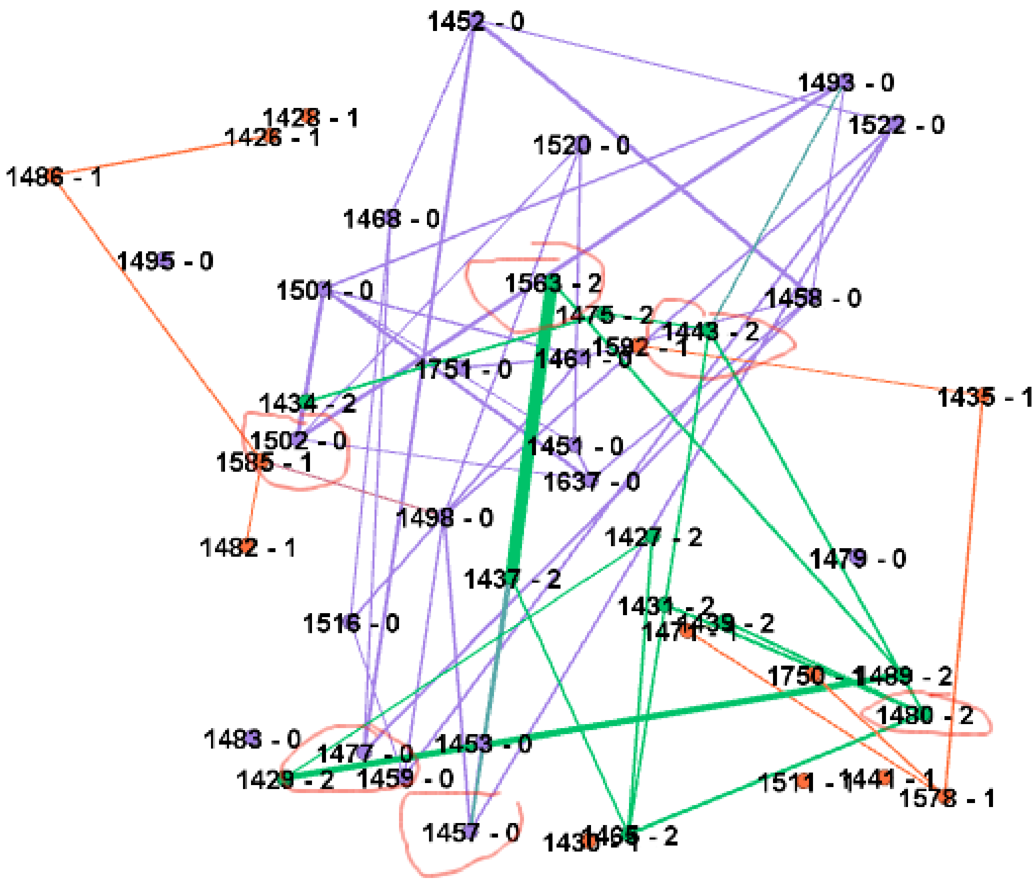


2.4 d)

First I visualized for the lowest values of the edge weights. This is to see that all class students are supposed to be connected with eachother being the classfellows. We know that this interaction is very weak but majority of edges carry low weights in this type of interaction.



And when I go from weak links to medium and high links, the number of links starting to decrease depicting that there are comparatively less interactions which are strong between students. And we can point out nodes which behave as a bridge between different nodes.



We can observe that community 0 and community 1 have strong connections between them as compared to community 1 as it has very few nodes in it. And by visualization, we can observe those nodes who have highest values for betweenness centrality like **1477,1443,1563,1502,1480** are acting as a bridge between two communities. And these nodes were among them who were identified by betweenness and closeness centrality as most influential and central nodes.

3 Exercise 2

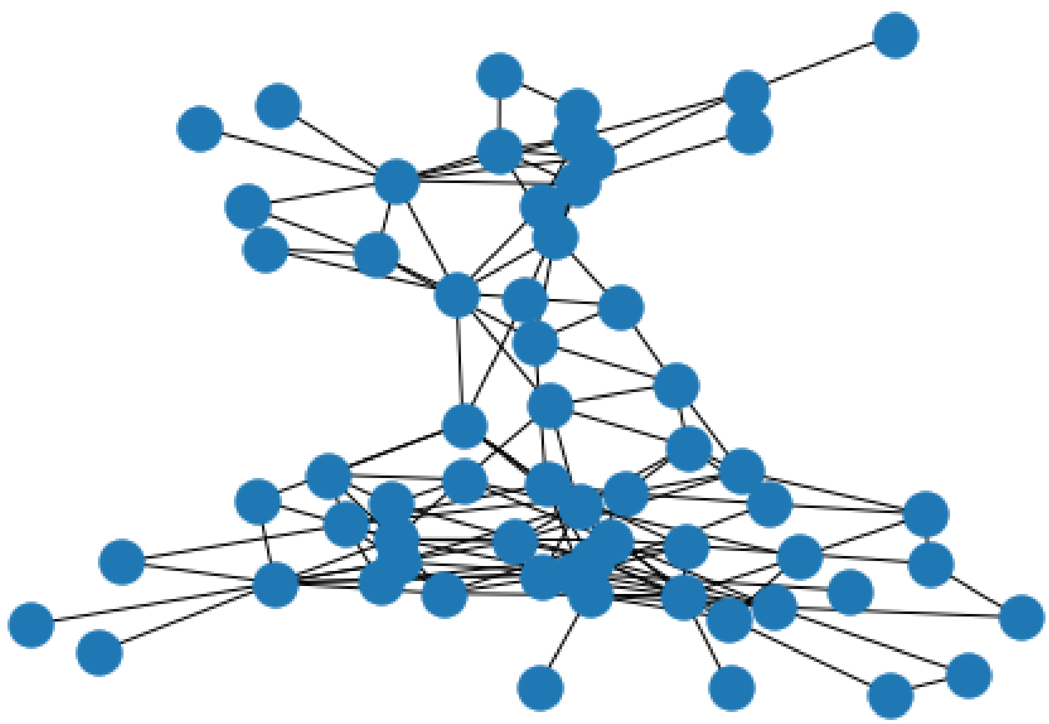
3.1 a)

For undirected graphs, density is defined as D = (2 * number of edges) / number of nodes So if I remove one node or add new edges between vertices, density would increase. The basic idea behind my proposed greedy algorithm is that if we remove a node with minimum degree of the graph, density should increase as the removed node would have minimum number of edges associated to it. So I can remove nodes with minimum degree and compute the densities of newly formed subgraphs and at the end, I can find the subgraph with higest density. 1. Find a node with minimum degree of the current graph and remove it. 2. Now we have a subgraph. Calculate its density. 3. Find a new node with minimum degree of the current subgraph and remove it 4. Repeat step 2 5. Keep repeating steps 2,3,4 untill no nodes left. 6. Find the subgraph with maximum density

3.2 b)

[45]:
```
import networkx as nx
import copy
dolphin_data = open('dolphins.txt')
edges = dolphin_data.readlines()
g=nx.Graph()
for idx,edge in enumerate(edges):
    if idx>0:
        start, end = edge.rstrip().split(' ')
        g.add_edge(start, end)
# drawing graph
nx.draw(g)
print(nx.number_of_nodes(g))
print(nx.density(g))
```

```
62
0.08408249603384453
```

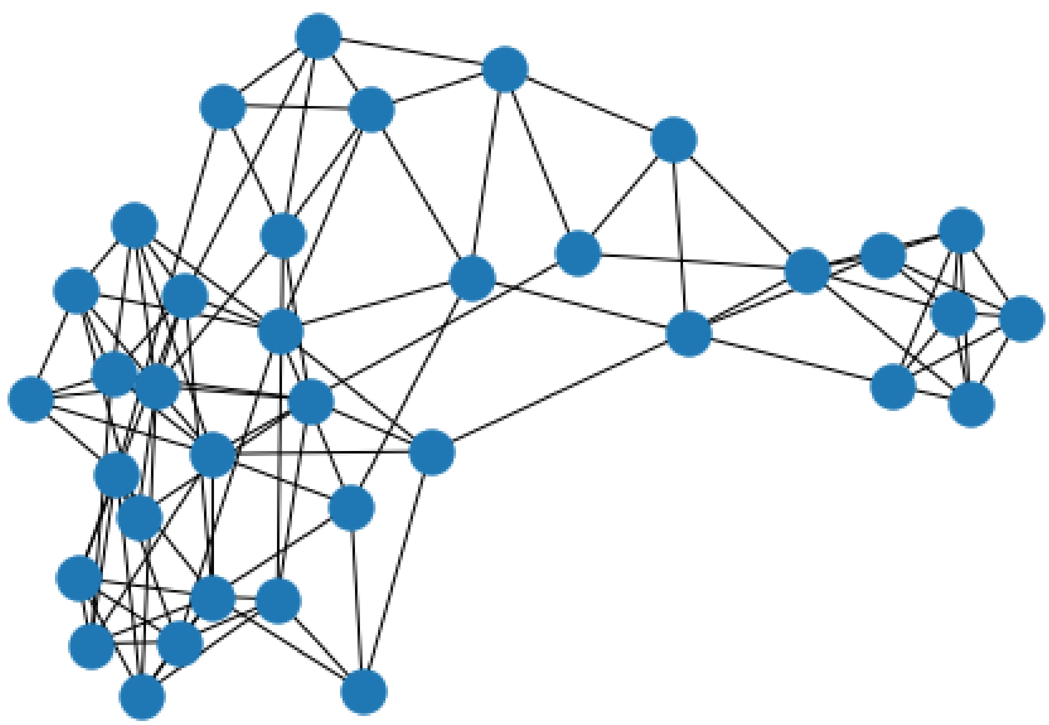


[46]:
```
%matplotlib inline
highest_density=0
while (nx.number_of_nodes(g) > 0):
    node_min_degree=sorted(g.degree, key=lambda x: x[1])[0][0]
```

```python
    g.remove_node(node_min_degree)
    nodes=nx.number_of_nodes(g)
    edges=nx.number_of_edges(g)
    if (nodes > 0):
        D = 2 * edges / nodes
        if (D > highest_density):
            highest_density= D
            sub_graph=copy.deepcopy(g)
```

[47]:
```python
print(highest_density)
nx.draw(sub_graph)
print(nx.number_of_nodes(sub_graph))
```

```
6.055555555555555
36
```



We started with a graph having 62 nodes and a density of 0.084 and after executing our algorithm, we ended with a subgraph of 36 nodes and density of 6.055.

4 Exercise 3

4.1 a)

Udist and **Mdist** between class M moecules and from class M molecules to their nearest neighbour

Q3:-

$$Udist(G_1, G_2) = 1 - \frac{|MCS(G_1, G_2)|}{|G_1| + |G_2| - |MCS(G_1, G_2)|}$$

$$MDIST(G_1, G_2) = 1 - \frac{|MCS(G_1, G_2)|}{\max\{|G_1|, |G_2|\}}$$

a):- Calculate MCG-based distance btw class M molecules

$$Udist(G_1, G_3) = 1 - \frac{9}{13+11-9} = 1 - \frac{9}{15} = \frac{6}{15} = 0.4$$

$$Udist(G_1, G_6) = 1 - \frac{13}{\cancel{13}+17-\cancel{13}} = 1 - \frac{13}{17} = 0.235$$

$$Udist(G_3, G_6) = 1 - \frac{9}{11+17-9} = 1 - \frac{9}{19} = 0.526$$

$$Mdist(G_1, G_3) = 1 - \frac{9}{13} = 0.3076$$

$$Mdist(G_1, G_6) = 1 - \frac{13}{17} = 0.235$$

$$Mdist(G_3, G_6) = 1 - \frac{9}{17} = 0.470$$

Calculate MCG-based distance from class M molecules to their nearest neighbours.

$$Udist(G_1, G_2) = 1 - \frac{8}{13+11-8} = 1 - \frac{8}{16} = 0.5$$

$$Udist(G_1, G_4) = 1 - \frac{9}{13+15-9} = 1 - \frac{9}{19} = 0.526$$

$$Udist(G_1, G_5) = 1 - \frac{5}{13+14-5} = 1 - \frac{5}{22} = 0.7728$$

$$Udist(G_3, G_2) = 1 - \frac{7}{11+11-7} = 1 - \frac{7}{15} = 0.533$$

$$Udist(G_3, G_4) = 1 - \frac{8}{11+15-8} = 1 - \frac{8}{18} = 0.555$$

$$Udist(G_3, G_5) = 1 - \frac{4}{11+14-4} = 1 - \frac{4}{21} = 0.809$$

$$Udist(G_6, G_2) = 1 - \frac{8}{17+11-8} = 1 - \frac{8}{20} = 0.6$$

$$Udist(G_6, G_4) = 1 - \frac{9}{17+15-9} = 1 - \frac{9}{23} = \frac{14}{23} = 0.608$$

$$Udist(G_6, G_5) = 1 - \frac{5}{17+14-5} = 1 - \frac{5}{26} = 0.8076$$

Nearest Neighbours of $G_1 = G_6$ and G_3

Nearest Neighbours of $G_3 = G_1$, G_6 and G_4

Nearest Neighbours of $G_6 = G_1$, G_3

Compare two MCG-based distances: Udist and Mdist

We already computed **Udist** between class M and to all other molecules. Let us calculate **Mdist** between class M and to all other molecules.

Which distance metric separates class M molecules better from other molecules?

$$MDist(G_1, G_2) = 1 - \frac{8}{13} = 0.3846$$

$$Mdist(G_1, G_4) = 1 - \frac{9}{15} = \frac{6}{15} = 0.4$$

$$Mdist(G_1, G_5) = 1 - \frac{5}{14} = 0.6428$$

$$Mdist(G_3, G_2) = 1 - \frac{7}{11} = 0.3636$$

$$Mdist(G_3, G_4) = 1 - \frac{8}{15} = 0.4666$$

$$Mdist(G_3, G_5) = 1 - \frac{4}{14} = \frac{10}{14} = 0.714$$

$$Mdist(G_6, G_2) = 1 - \frac{8}{17} = 0.529$$

$$Mdist(G_6, G_4) = 1 - \frac{9}{17} = 0.4705$$

$$Mdist(G_6, G_5) = 1 - \frac{5}{17} = 0.705$$

It is evident that **Udist** is showing larger values between M and other molecules as compared with **Mdist** separating class M molecules from other molecules in more precise manner. So I would infer that **Udist** is a better choice in this case.

4.2 b)

b)

Let M be the probability of belonging to class M molecule.

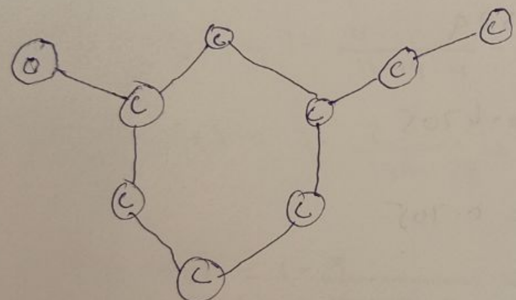
$$P(M) = \frac{3}{6} = 0.5$$

$$P(\neg M) = 1 - 0.5 = 0.5$$

Joint probability of subgraph (MCG) to belong to class M molecules:

$$P(G, M) = 0.5$$

We identified following MCG in all molecules belonging to class M. And it is not present in ¬M classes.



So, Probability of MCG in class $M = 0.5$

Probability of not finding MCG in ¬M class $= 0.5$

$$\phi(G \rightarrow M) = P(M \mid G) = \frac{P(GM)}{P(G)} = \frac{0.5}{0.5} = 1$$

$$P(\neg G \rightarrow \neg M) = P(\neg M \mid \neg G) = \frac{P(\neg G \neg M)}{P(\neg G)} = \frac{0.5}{0.5} = 1$$

Our identified MCG very strongly predict class M molecules with 100% confidence as it is present in all class M molecules and not present in molecules which do not belong to class M.

4.3 c)

I do not know

4.4 d)

I do not know

5 Exercise 5

[48]:
```
#Provided code in the excercise session
import nltk
import random
nltk.download('book')
from nltk.corpus import movie_reviews
```

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /home/binsha1/nltk_data…
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package brown to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package chat80 to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package chat80 is already up-to-date!
[nltk_data]    | Downloading package cmudict to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package conll2000 to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package conll2000 is already up-to-date!
[nltk_data]    | Downloading package conll2002 to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package conll2002 is already up-to-date!
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package dependency_treebank is already up-to-date!
[nltk_data]    | Downloading package genesis to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package ieer to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package ieer is already up-to-date!
[nltk_data]    | Downloading package inaugural to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /home/binsha1/nltk_data…
```

```
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package nps_chat to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package nps_chat is already up-to-date!
[nltk_data]    | Downloading package names to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package ppattach to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package ppattach is already up-to-date!
[nltk_data]    | Downloading package reuters to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package reuters is already up-to-date!
[nltk_data]    | Downloading package senseval to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package senseval is already up-to-date!
[nltk_data]    | Downloading package state_union to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package state_union is already up-to-date!
[nltk_data]    | Downloading package stopwords to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package swadesh to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package swadesh is already up-to-date!
[nltk_data]    | Downloading package timit to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package timit is already up-to-date!
[nltk_data]    | Downloading package treebank to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package toolbox to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package toolbox is already up-to-date!
[nltk_data]    | Downloading package udhr to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package udhr is already up-to-date!
[nltk_data]    | Downloading package udhr2 to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package udhr2 is already up-to-date!
[nltk_data]    | Downloading package unicode_samples to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package unicode_samples is already up-to-date!
[nltk_data]    | Downloading package webtext to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package webtext is already up-to-date!
[nltk_data]    | Downloading package wordnet to
[nltk_data]    |     /home/binsha1/nltk_data…
```

```
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_treebank_pos_tagger to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package maxent_treebank_pos_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package universal_tagset to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package universal_tagset is already up-to-date!
[nltk_data]    | Downloading package punkt to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package book_grammars is already up-to-date!
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package city_database is already up-to-date!
[nltk_data]    | Downloading package tagsets to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package tagsets is already up-to-date!
[nltk_data]    | Downloading package panlex_swadesh to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package panlex_swadesh is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /home/binsha1/nltk_data…
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection book
```

[49]:
```
documents = [(list(movie_reviews.words(fileid)), category)
              for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
```

[50]:
```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = set(list(all_words)[:2000]) # We only focus on the most␣
 ↪frequent 2000 words.
```

```
# Use one-hot encoding as features. 1 means the given word is contained in the␣
↪document and 0 otherwise.
def document_features(document): # [_document-classify-extractor]
    document_words = set(document) # [_document-classify-set]
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

[51]:
```
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
classifier.show_most_informative_features(5)
```

```
0.81
Most Informative Features
 contains(unimaginative) = True              neg : pos    =      8.4 : 1.0
    contains(schumacher) = True              neg : pos    =      7.4 : 1.0
     contains(atrocious) = True              neg : pos    =      6.6 : 1.0
       contains(singers) = True              pos : neg    =      6.3 : 1.0
        contains(turkey) = True              neg : pos    =      6.1 : 1.0
```

5.1 a)

[52]:
```
from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
```

Remove stopwords

[53]:
```
stopwords=stopwords.words('english')
documents = [(list(movie_reviews.words(fileid)), category)
              for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
```

[54]:
```
#Removing stopwords
word_feature = [word for word in list(all_words)[:2000] if word not in␣
↪stopwords]
def document_features(document): # [_document-classify-extractor]
    document_words = set(document) # [_document-classify-set]
    features = {}
    removed_words= [word for word in document_words if word not in stopwords]
    for word in word_feature:
        if 'count({})'.format(word) in features:
```

```
            features['count({})'.format(word)] =features['count({})'.
↪format(word)] + removed_words.count(word)
        else:
            features['count({})'.format(word)] = removed_words.count(word)
    return features
```

[55]:
```
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
classifier.show_most_informative_features(5)
```

```
0.86
Most Informative Features
    count(unimaginative) = 1                 neg : pos    =      8.3 : 1.0
        count(atrocious) = 1                 neg : pos    =      7.0 : 1.0
           count(shoddy) = 1                 neg : pos    =      7.0 : 1.0
       count(schumacher) = 1                 neg : pos    =      6.6 : 1.0
          count(singers) = 1                 pos : neg    =      6.3 : 1.0
```

Improved accuracy has proved that removal of stopwords and creating word features on the basis of frequency has resulted in the performance gain for the classifier. We can see that Most Informative Features are showing positive and negative examples now. Reviewes are being classified correctly on the basis of word used which can be seen above. Noisy data can still be seen as an example like schumacher which is a name.

5.2 b)

For part b, punctuation in the sentences is removed as it does not add any value to the word feature. Then I removed some of propernouns like name of person, country as they are not useful in determining the sentiment of the sentence. To get the max value from word features, I exploited the use of hypernym.

[56]:
```
# removed stop words, punctutaions and some random words which seem to be␣
↪spelling errors or people's names
word_features = [word for word in list(all_words)[:1500] if word.isalnum()
                 and word not in stopwords and word not in ['justin', 'turkey',␣
↪'schumacher', 'suvari', 'mena']]

def document_features(document):
    document_words = set(document)
    features = {}
    removed_words = [word for word in document_words if word.isalnum() and word␣
↪not in stopwords and
                        word not in ['justin', 'turkey', 'schumacher',␣
↪'suvari', 'mena']]
    for word in word_features:
        hyper_exists = False
```

```python
        syn_list = wn.synsets(word)
        if syn_list:
            syn = syn_list[0]
            # get hypernym of a word
            hypernyms = syn.hypernyms()
            if hypernyms:
                hypernym = hypernyms[0]
                root = hypernym.name().split('.')[0]
                hyper_exists = True
        if hyper_exists:
            if 'count({})'.format(root) in features:
                features['count({})'.format(root)] += features['count({})'.
↪format(root)] + removed_words.count(word)
            else:
                features['count({})'.format(root)] = removed_words.count(word)
        else:
            if 'count({})'.format(word) in features:
                features['count({})'.format(word)] = features['count({})'.
↪format(root)] + removed_words.count(word)
            else:
                features['count({})'.format(word)] = removed_words.count(word)
    return features
```

[57]:
```python
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))

classifier.show_most_informative_features(5)
```

```
0.84
Most Informative Features
         count(musician) = 1                 pos : neg    =      6.3 : 1.0
          count(knowing) = 2                 pos : neg    =      5.9 : 1.0
           count(people) = 80                neg : pos    =      5.7 : 1.0
          count(bronson) = 1                 neg : pos    =      5.7 : 1.0
      count(police_work) = 1                 neg : pos    =      5.7 : 1.0
```

We can see that our model is not using the names of the people or places like schumacher or turkey which add zero value in sentiment of the word features.

[ ]:
```python

```