CS-E4650 Methods of Data Mining

Assignment 3

Rosales Rodríguez, Pablo

Student ID: 914769

Contents

Exercise 1

Identification of most central and influential nodes.

Initially, the most central and influential nodes are identified using different metrics:

Node degree	Weighted degree	Closeness centrality	Betweenness centrality
1443	1437	1443	1443
1477	1563	1477	1477
1457	1457	1457	1502
1502	1458	1502	1457
1428	1452	1428	1563
1452	1477	1452	1480
1563	1498	1563	1522
1426	1480	1426	1585

As it is depicted in the previous table, the top central nodes are the same considering the node degree and considering the closeness centrality. Most of them equal to the ones obtained when classifying them using betweenness centrality. Finally, the most different ones are those top central based on weighted degree method.

Identification of communities

When applying the modularity method for identifying potential communities, four groups are obtained. Taking them in groups of two, each of the pairs represents one of the classes, 5A and 5B. Considering the four of them, in the pair of groups belonging to class 5B, one of them represents the females and the other one the males. Considering the pair of groups belonging to 5A, the division based on gender is not that clear as in the previous case. The result is represented in figure 1.

When applying Girvan-Newman clusterization, 24 classes are obtained. One of them is clearly representing the students in group 5B, although there is one student belonging to 5A, such a node can be considered to have a minimum influence. However, the other classes identified do not attend to any of the possible criteria presented in the background information. The result is represented in figure 2.

Visual representation of the communities

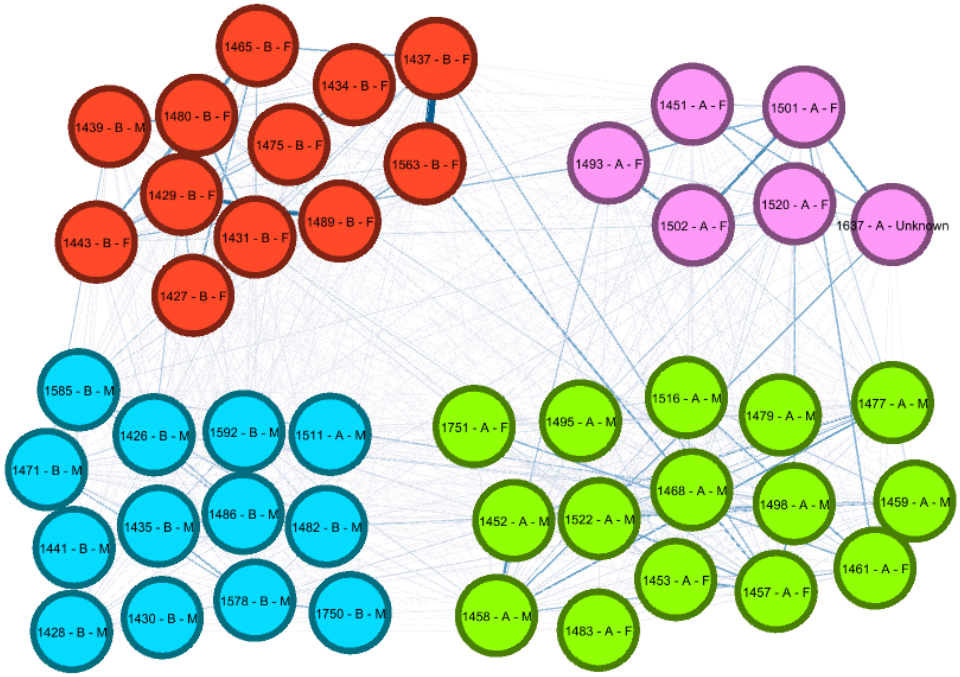


Figure 1. Representation of the classes identified with the Modularity method.

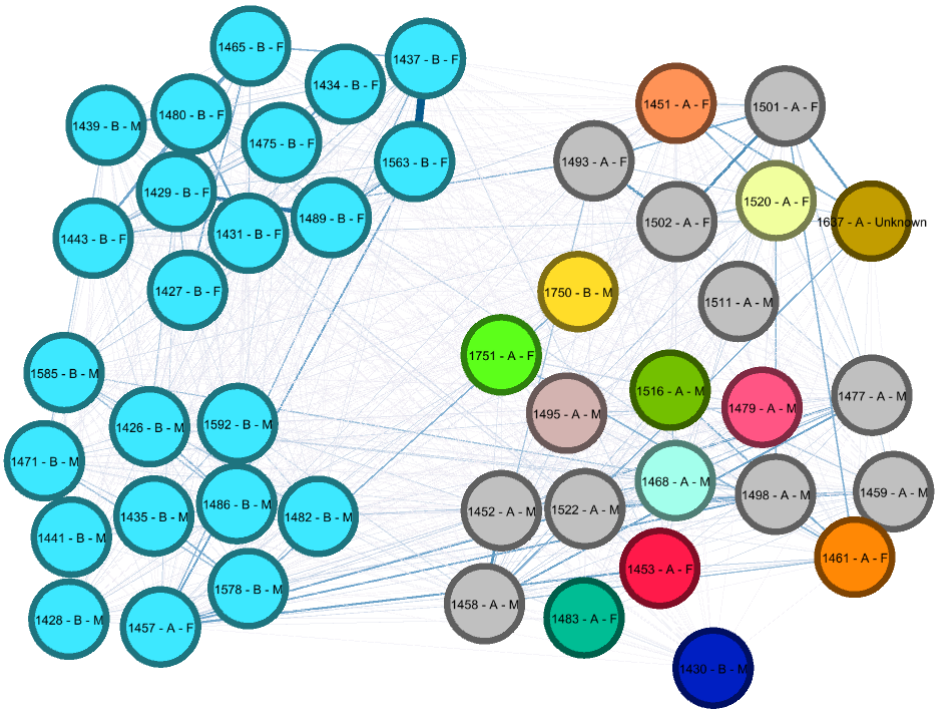


Figure 2. Representation of the classes identified with the Girvan-Newman clusterization method.

Role of the central nodes and analysis of the connections

When modifying the threshold and analyzing the links between communities, it is clear that there are some nodes strongly connected with others in their community, such as the connection between the nodes 1563 and 1437, in the community defined as 5B and females. Actually, when lowering the threshold, it can be seen that the mentioned

community, as well as part of the one composed by students in group 5A have the strongest connections. However, the community in which males in group 5B can be found has a larger number of connections.

In figure 3, the central nodes were slightly separated from their classes. There is a slightly greater concentration of edges around them, hence they act as a bridge between some of the communities, such as the case between the red one and the green one, connected through the central nodes 1457 and 1563.

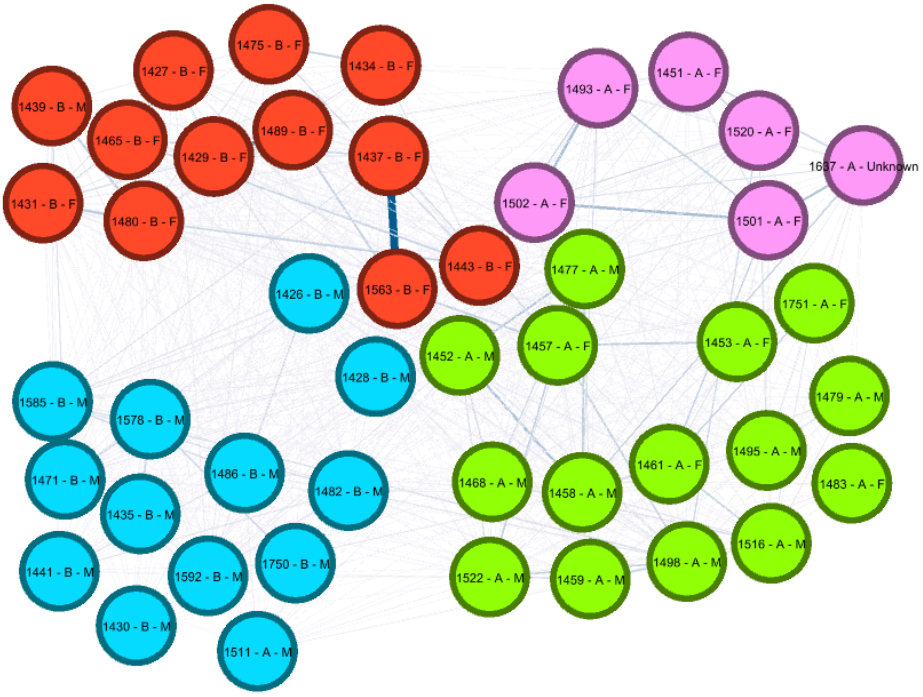


Figure 3. Central nodes are slightly separated from their communities.

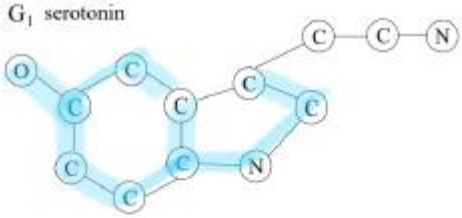
Even though some central nodes act as bridges between communities, it seems to be more frequent that their role is acting as central nodes inside their community, meaning that the number of connections between these nodes and those of their same group is high, hence they are the most influential nodes inside their own community.

Exercise 3

Identification of maximum common subgraphs and calculation of distances

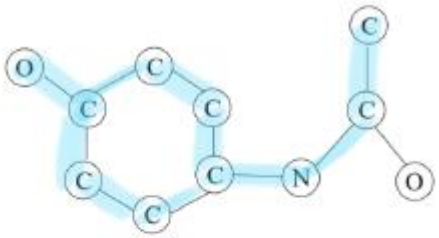
In the following pages, the maximum common subgraphs for the pairs composed by a molecule of class M and one of those belonging to class ¬M are represented.

G_1 serotonin



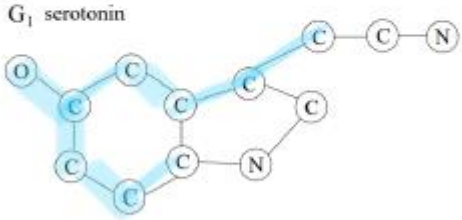
$|MCS(G_1, G_2)| = 10$

G_2 acetaminophen

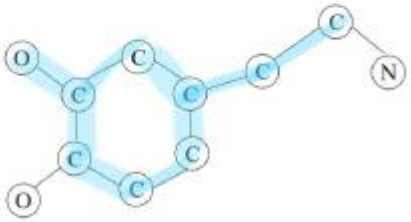


G_3 dopamine

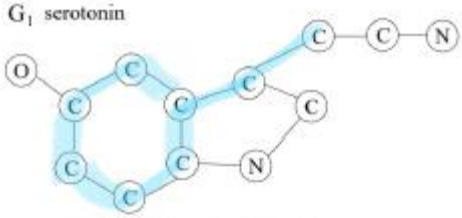
G_1 serotonin



$|MCS(G_1, G_3)| = 9$

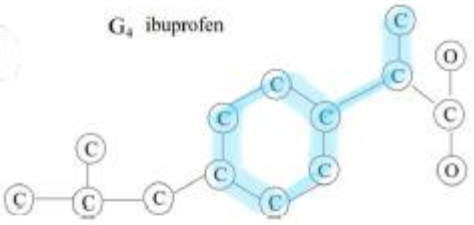


G_1 serotonin

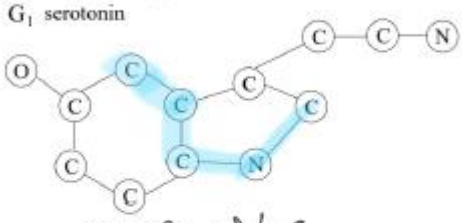


$|MCS(G_1, G_4)| = 8$

G_4 ibuprofen

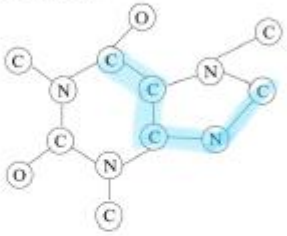


G_1 serotonin



$|MCS(G_1, G_5)| = 5$

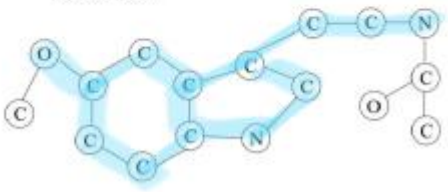
G_5 caffeine



G_1 serotonin

$|MCS(G_1, G_6)| = 13$

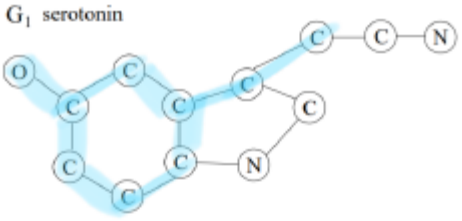
G_6 melatonin



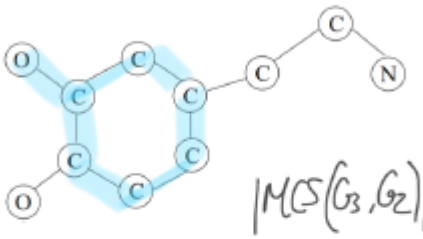
G_3 dopamine

$|MCS(G_3, G_4)| = 9$

G_1 serotonin

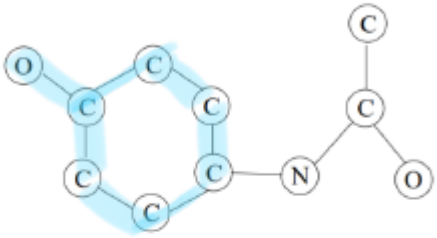


G_3 dopamine

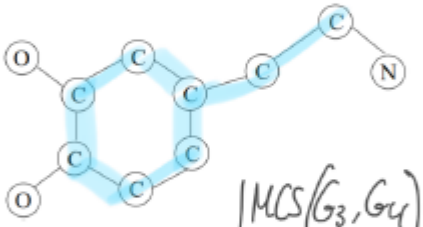


$|MCS(G_3, G_2)| = 7$

G_2 acetaminophen

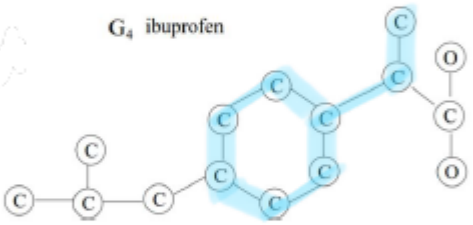


G_3 dopamine



$|MCS(G_3, G_4)| = 8$

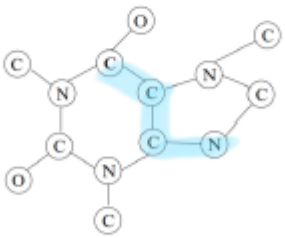
G_4 ibuprofen



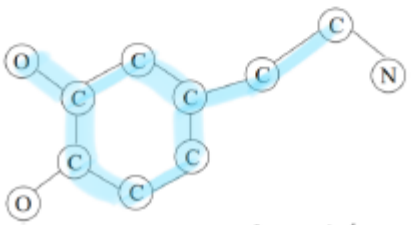
G_3 dopamine

$|MCS(G_3, G_5)| = 4$

G_5 caffeine



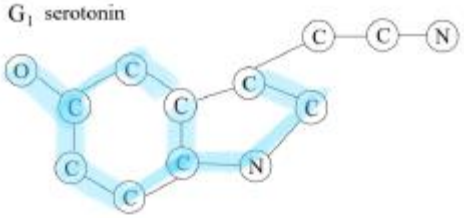
G_3 dopamine



$|MCS(G_3, G_6)| = 9$

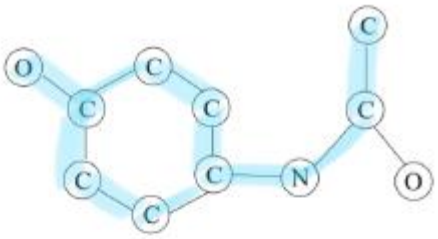
G_6 melatonin

G_1 serotonin

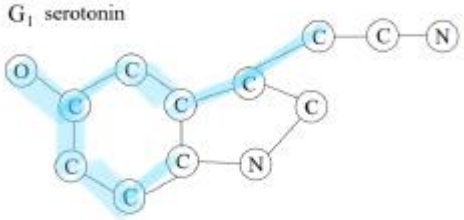


$|MCS(G_1, G_2)| = 10$

G_2 acetaminophen

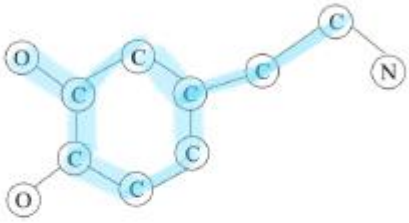


G_1 serotonin

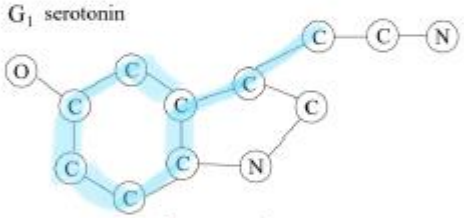


$|MCS(G_1, G_3)| = 9$

G_3 dopamine

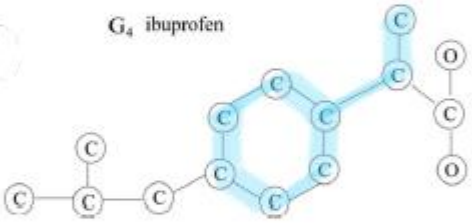


G_1 serotonin

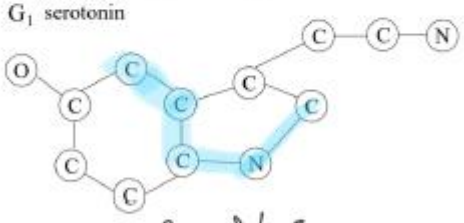


$|MCS(G_1, G_4)| = 8$

G_4 ibuprofen

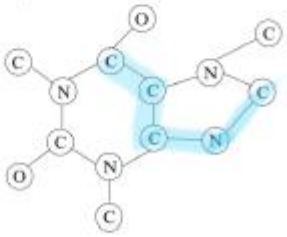


G_1 serotonin



$|MCS(G_1, G_6)| = 5$

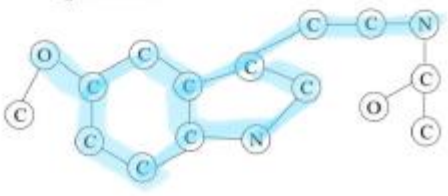
G_5 caffeine



G_1 serotonin

$|MCS(G_1, G_6)| = 13$

G_6 melatonin



Distances between the M-class molecules

Considering the general equation for the two distances that are computed:

$$Udist(G_1, G_2) = 1 - \frac{|MCS(G_1, G_2)|}{|G_1| + |G2| - |MCS(G_1, G_2)|}$$

$$Mdist(G_1, G_2) = 1 - \frac{|MCS(G_1, G_2)|}{Max\{|G_1|, |G2|\}}$$

G1-G3

$$Udist(G1, G3) = 1 - \frac{9}{13 + 11 - 9} = 0.4000$$

$$Mdist(G_1, G_3) = 1 - \frac{9}{13} = 0.3077$$

G1-G6

$$Udist(G_1, G_6) = 1 - \frac{13}{13 + 17 - 13} = 0.2353$$

$$Mdist(G_1, G_6) = 1 - \frac{13}{17} = 0.2353$$

G3-G6

$$Udist(G_3, G_6) = 1 - \frac{9}{11 + 17 - 9} = 0.5263$$

$$Mdist(G_3, G_6) = 1 - \frac{9}{17} = 0.4706$$

Distances between M-class molecules and those belonging to ¬M class

G1-G2

$$Udist(G_1, G_2) = 1 - \frac{10}{13 + 11 - 10} = 0.2857$$

$$Mdist(G_1, G_2) = 1 - \frac{10}{13} = 0.2308$$

G1-G4

$$Udist(G_1, G_4) = 1 - \frac{8}{13 + 15 - 8} = 0.6000$$

$$Mdist(G_1, G_4) = 1 - \frac{8}{15} = 0.4667$$

G1-G5

$$Udist(G_1, G_5) = 1 - \frac{5}{13 + 14 - 5} = 0.7727$$

$$Mdist(G_1, G_5) = 1 - \frac{5}{14} = 0.6429$$

G3-G2

$$Udist(G_3, G_2) = 1 - \frac{7}{11 + 11 - 7} = 0.5333$$

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

G3-G4

$$Udist(G_3, G_4) = 1 - \frac{8}{11 + 15 - 8} = 0.5556$$

$$Mdist(G_3, G_4) = 1 - \frac{8}{15} = 0.4667$$

G3-G5

$$Udist(G_3, G_5) = 1 - \frac{4}{11 + 14 - 4} = 0.8095$$

$$Mdist(G_3, G_5) = 1 - \frac{4}{14} = 0.7143$$

G6-G2

$$Udist(G_6, G_2) = 1 - \frac{10}{17 + 11 - 10} = 0.4444$$

$$Mdist(G_6, G_2) = 1 - \frac{10}{17} = 0.4118$$

G6-G4

$$Udist(G_6, G_4) = 1 - \frac{8}{17 + 15 - 8} = 0.6667$$

$$Mdist(G_6, G_4) = 1 - \frac{8}{17} = 0.5294$$

G6-G5

$$Udist(G_6, G_5) = 1 - \frac{5}{17 + 14 - 5} = 0.8077$$

$$Mdist(G_6, G_5) = 1 - \frac{5}{17} = 0.7059$$

Considering the previous distances, in both metrics, when having perfectly matching graphs, the distance between them is 0, meaning that the closer to 1 is the distance, the "further" those graphs are; the more different. Considering this reasoning, the union-normalized distance separates better those nodes belonging to different molecules, since it returns higher values, closer to 1.

Identification of maximum common subgraph for M-class molecules

In the figure 4, the maximum common subgraph between the M-class molecules is represented.

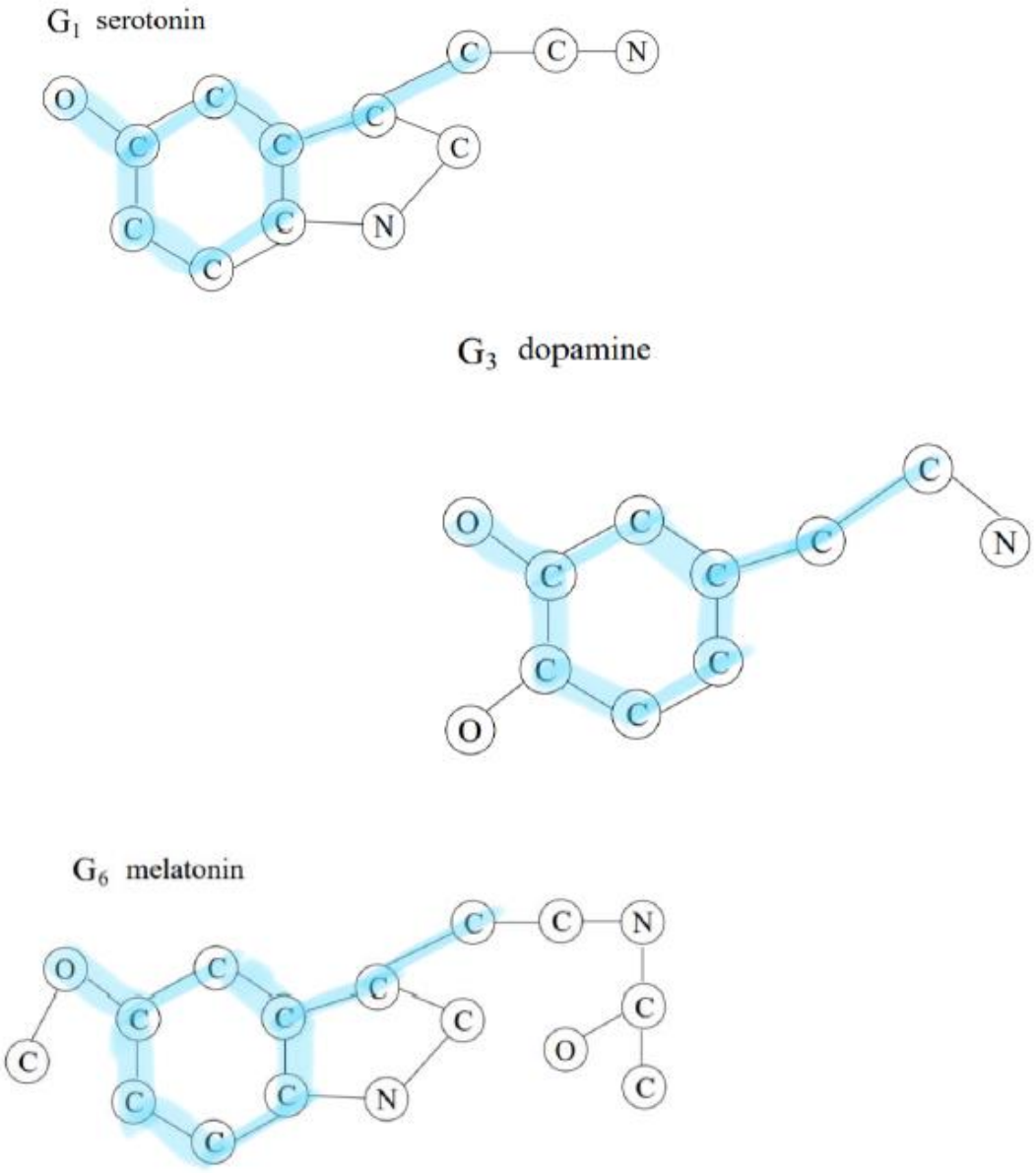


Figure 4. Representation of the MCG between the M-class molecules.

True \ predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Considering the definition of precision as:

$$Precision = \frac{TP}{TP + FP}$$

Rule $G \rightarrow M$

The number of true positives in the given case would be 3: those graphs containing the MCG and belonging to the M-class.

The number of true negatives would be also 3: those graphs that do not contain the MCG and do not belong to the M-class.

The number of false positives would be 0: those graphs that do not contain the MCG but belong to the M-class molecules.

Finally, the number of false negatives is also 0, since there are no graphs that containing the MCG and not belonging to the M-class.

$$Precision = \frac{3}{3+0} = 1$$

Hence, the precision of the rule is 100%.

The same happens for the rule $\neg G \rightarrow \neg M$. It also has precision 100%.

Statistical dependency G-A is not a monotonic property

Considering those molecules in class M as having a common attribute, such as drug and naming the MCG as G, then, the following rule is true:

$$G \rightarrow A$$

However, when considering a subgraph of G, such as the one in red in the following figure, the rule is no longer valid, since that subgraph is also part of those graphs that belong to a different class (no drug), meaning that there is independency between this subgraph and the attribute *drug*.

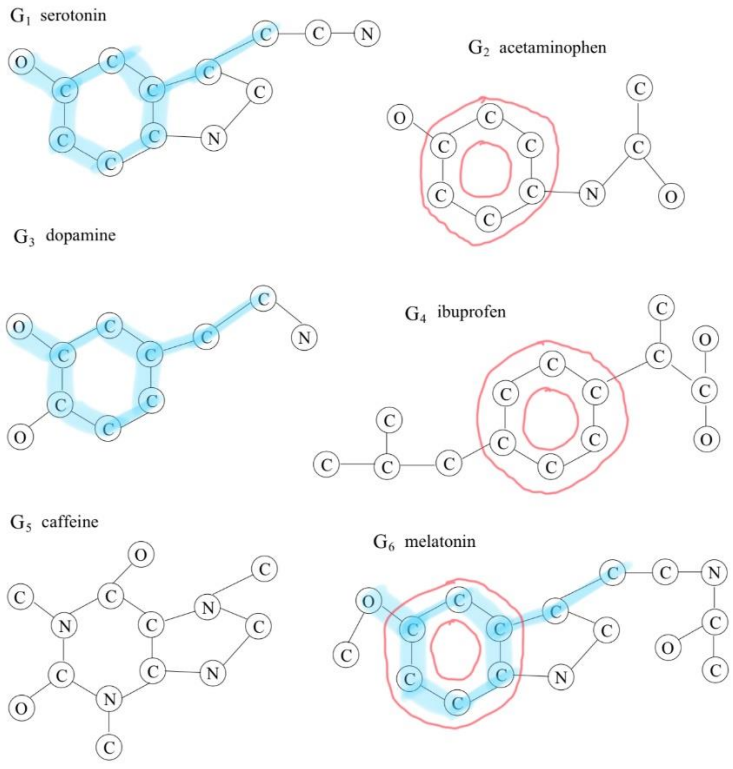


Figure 5.

Statistically significant associations describing positive dependencies

It is possible to find statistically significant associations describing positive dependencies using GraphApriori. However, it is not the best approach. GraphApriori has a low computational efficiency, since the number of patterns candidate could be very large. Moreover, in each iteration there could be several options that are valid according to the algorithm: isomorphic matchings. This leads, in some occasions, to generate the same subgraph, so it must be discarded.

Exercise 4

The recommendation system implemented is based on a SimRank algorithm. The main target of a SimRank algorithm is establishing the similarity between nodes. In the given case, the algorithm is bipartite-type, since there are users and items, in this case, jokes, related as following:

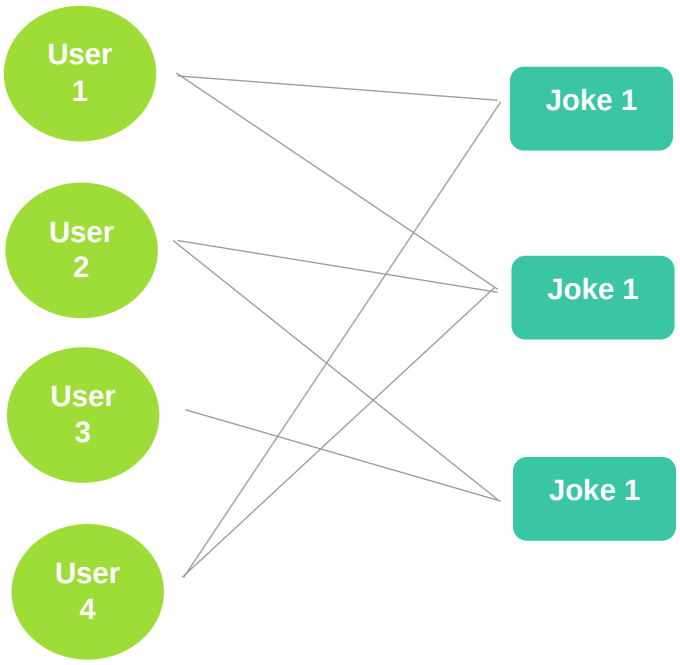


Figure 6. Simplified representation of the graph for the recommendation algorithm (It is an example).

The graph represented in figure 6 can be implemented in a matrix as following:

	$User\ ID$	$J1$	$J2$	$J3$	$Total$
$U1$	...	...	...	...	...
$U2$	...	...	...	...	...
$U3$	...	...	...	...	...
$U4$	...	...	...	...	...

The previous matrix is directly extracted from the document csv-format document provided. It was called X in the code.

Two more matrices were defined for computing the SimRank between users:

- The matrix that contains the SimRank between users. It has the following shape:

	$U1$	$U2$	$U3$	$U4$
$U1$	...	...	...	...
$U2$	...	...	...	...
$U3$	...	...	...	...
$U4$	...	...	...	...

- The matrix that contains the SimRank between jokes.

	$J1$	$J2$	$J3$
$J1$	...	...	...
$J2$	...	...	...
$J3$	...	...	...

Having the definition for the previous variables, the SimRank can be computed.

The main equation for the SymRank between two objects a and b is:

$$s(a,b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$$

Where |*I(a)*| represents the number of neighbors of the object a, meaning those nodes connected to a. In the given case, when computing the SimRank between users, |*I(user)*| represents the number of jokes that the user liked. In the opposite case, when computing the SimRank between jokes, |*I(joke)*| represents the number of users who liked the joke. Finally, $I_i(a)$ represents each of those neighbors. In the user case, each of the jokes that he or she liked, while in the joke case, it refers to the each of the users that liked the joke.

Given the previous explanation, the designed algorithm has as main target filling the matrix that contains the SimRank for the users. In order to make it converge, it performs a fixed number of iterations, represented by the variable *k*. In the first iteration the matrix is filled as:

	$U1$	$U2$	$U3$	$U4$
$U1$	1	0	0	0
$U2$	0	1	0	0
$U3$	0	0	1	0
$U4$	0	0	0	1

In the following iterations, the algorithm proceeds as following:

- It starts by going through each of the positions in the matrix (iterators *u_row* and *u_column*).
- For each of the positions, the algorithm goes through the matrix X, searching for a number greater than 0, meaning that it looks for those jokes that the user liked. The users are given by *u_row* and *u_column*, since the point is computing the similarity between those two to fill the corresponding cell in the matrix.
- Those cells with a 1 in the X matrix, represent the jokes they liked and, thus, those jokes between which the SimRank must be computed.
- The SimRank of the jokes is computed following the same strategy and the result is returned in each iteration, so that one of the users can be calculated.

In a very simple example, supposing that the SimRank between the users U1 and U2 must be computed, in the second iteration, given the following X matrix:

	$User\ ID$	$J1$	$J2$	$J3$	$Total$
$U1$	...	1	0	1	2
$U2$	...	0	1	1	2
$U3$	...	0	1	0	1
$U4$	...	1	1	1	3

The SimRank between the two first users is:

$$s(U_1, U_2) = \frac{C_1}{|2|\,|2|}[s(J_1, J_2) + s(J_1, J_3) + s(J_3, J_2) + s(J_3, J_3)]$$

In order to compute the previous expression, the SimRank between jokes must be computed as:

$$s(J_1, J_2) = \frac{C_2}{|2|\,|3|}[s(U_1, U_2) + s(U_1, U_3) + s(U_1, U_4) + s(U_4, U_2) + s(U_4, U_3) + s(U_4, U_4)]$$

Considering the first iteration, as mentioned before:

$$s(J_1, J_2) = \frac{C_2}{|2|\,|3|}[0 + 0 + 0 + 0 + 0 + 1]$$

The algorithm would compute the rest of the jokes similarities and return the matrix, so the SimRank for the users can be computed.

The previous explanation was applied to a simple example with four users and three jokes. The program implemented on Python follows the same strategy. However, the provided data base is much larger.

Once the program has computed the SimRank between users and has filled the matrix, the recommendation can be carried out. An additional extract of code with a few lines was programmed to do so. The user for who the recommendation is wanted, can be chosen. The program, automatically, checks if it is an existing user. If so, it finds the 50 most similar users to it, as those users with a higher SimRank value in the row of the given user, obviously ignoring itself. Having those 50 users, the algorithm goes through them, from the most similar to the least similar, if necessary, and fills an array called *recommendation* that contains those jokes that these similar users liked but that the chosen user did not consider.

Hence, when running the algorithm for a chosen user, such as the one with ID 4519:

User ID	**Joke 5**	**Joke 7**	**Joke 8**	**Joke 13**	**Joke 15**	**Joke 16**	**Joke 17**	**Joke 18**	**Joke 19**	**Joke 20**	**Total**
4519	0	1	0	0	0	1	1	1	0	1	5

It extracts those 50 users who are the most similar to the 4159 one. They are shown in the following figure.

```
In [9]: print(indices)
[  4  90 534 297 628 566 551 406 421 183 632 698 684 604 560   3 343 693
 114  89  57 644 209 749 615  65 368 555 385 588 319 513 300 393 686 217
 770 119 607 157 387 176 439 722 508 279 255 189 786 627]
```

Figure 7. Indices of those users that are most similar to the one with ID 4519. They are ordered by similarity, from the top 1 to the top 50.

Given the figure 7, the user 4 is the most similar to the 4519, so those jokes that 4 liked but 4519 did not consider, will be recommended.

The algorithm yields a recommendation of three jokes:

- Joke 4
- Joke 0
- Joke 8

Exercise 5

Preprocessing

The code provided in exercise session 3 does already have document features based on the frequencies of words in positive and negative reviews. The given code yields an accuracy of 81% and the following result for the most informative features:

```
0.81
Most Informative Features
  contains(unimaginative) = True              neg : pos    =      8.2 : 1.0
           contains(ugh) = True              neg : pos    =      8.2 : 1.0
     contains(schumacher) = True              neg : pos    =      7.3 : 1.0
      contains(atrocious) = True              neg : pos    =      6.9 : 1.0
         contains(shoddy) = True              neg : pos    =      6.9 : 1.0
```

Figure 8. Result of the classification.

In order to perform the preprocessing steps, the following extracts of code were added to the default one:

```
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
import re
```

```
# Step 1: tokenization
tokens_list = [word_tokenize(i) for i in all_words]
print('After tokenization (first five elements):')
for i in tokens_list[:5]:
    print(i)
```

```
# Step 2: stemming
porter = PorterStemmer()
stemmed_tokens_list = []
for i in tokens_list:
    stemmed_tokens_list.append([porter.stem(j) for j in i])
print('After stemming (first five elements):')
for i in stemmed_tokens_list[:5]:
    print(i)
print()
print('The grammatic root is obtained')
```

```
# Steps 3 and 4: stop words removal and punctuation removal
stop_words = set(stopwords.words('english'))
print('Stop words from NLTK english: ')
print(stop_words)
print()
```

```
# The punctuation is added to the stop words
punctuation = set("!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~")
stop_words.update(punctuation)
stop_words.add("...")
```

```
filtered_words = []
for i in stemmed_tokens_list:
    filtered_words.append([token for token in i if token not in stop_words])

# Numbers removal
filtered_words = [' '.join(i) for i in filtered_words]
filtered_words = [ re.sub(r'\d+','', word) for word in filtered_words]

print('After preprocessing:')
for i in filtered_words[:5]:
    print(i)


# From now on, the filtered words are used for classifying
all_words = filtered_words
```

The previous steps were implemented to remove the stop words from the total words that were extracted from the library. As a consequence of this, the accuracy decreased with respect to the situation exposed above. However, the accuracy is still good, and the decrease is not significant. This is due to the greater difficulty to perform classification because of the lower number of words and because of the use of the grammatical roots.

```
0.79
Most Informative Features
              apparently = 2              neg : pos    =     13.8 : 1.0
                  boring = 2              neg : pos    =     11.9 : 1.0
                     bad = 5              neg : pos    =     11.1 : 1.0
                     ugh = 1              neg : pos    =      8.4 : 1.0
                      we = 8              pos : neg    =      8.2 : 1.0
```

Figure 9. Result of the classification.

Augmenting the document

In this second task of the exercise, the document was modified in order to include the hypernyms of the words, instead of the proper words. For those words not having a hypernym, the words themselves are used. For extracting the hypernyms, as suggested, the first synset was considered as the most useful one. In the figure 10 there is an image of how the code was modified.

```
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:
        word_synset = wn.synsets(word)
        if len(word_synset)>0:
            word_synset = word_synset[0]
            word_hyp = word_synset.hypernyms()
            if len(word_hyp)>0:
                word_hyp = word_hyp[0]
                word_hyp = word_hyp.lemma_names()
                word_hyp = word_hyp[0]
                features['contains({})'.format(word_hyp)] = word_hyp in document_words
            else:
                features['contains({})'.format(word)] = word in document_words
        else:
            features['contains({})'.format(word)] = word in document_words

    return features
```

Figure 10. Code for including the hypernyms in the features of the document.

After the explained modification, the accuracy decreases to 0.8, while with the one proposed in the document, the accuracy remains on 0.81. This could be a consequence

of the use of hypernyms, because many words can share the same hypernym, making it more difficult to identify different classes among them.

If, instead, only the hypernyms are taken into account (for those words that have one), the accuracy plummets to 0.61.