National Technical University of Athens

Interdisciplinary Master's Programme in

Data Science and Machine Learning

ALGORITHMIC DATA SCIENCE

Exercise Sheet 1

[redacted]

May 15, 2022

Exercise 1

Exercise 6.3.1 - MMDS

Here is a collection of twelve baskets. Each contains three of the six items 1 through 6.

$$\begin{array}{llll} \{1,2,3\} & \{2,3,4\} & \{3,4,5\} & \{4,5,6\} \\ \{1,3,5\} & \{2,4,6\} & \{1,3,4\} & \{2,4,5\} \\ \{3,5,6\} & \{1,2,4\} & \{2,3,5\} & \{3,4,6\}. \end{array}$$

Suppose the support threshold is 4. On the first pass of the PCY Algorithm we use a hash table with 11 buckets, and the set $\{i, j\}$ is hashed to bucket $i \times j \mod 11$.

(a) By any method, compute the support for each item and each pair of items.

(b) Which pairs hash to which buckets?

(c) Which buckets are frequent?

(d) Which pairs are counted on the second pass of the PCY Algorithm?

Solution (a) To find the support of singletons we count for every item (e.g 1,2,3,4,5,6) how many times it appears in the twelve baskets. By a simple count we find that

$$1 \longrightarrow 4,\ 2 \longrightarrow 6,\ 3 \longrightarrow 8$$
$$4 \longrightarrow 8,\ 5 \longrightarrow 6,\ 6 \longrightarrow 4,$$

where by $i \longrightarrow j$ we mean that the item $1 \leq i \leq 6$ appears in j out of twelve baskets. To find the support of each pair of items we first form all subsets $\{i, j\}$ with $i \neq j$ of $\{1, 2, 3, 4, 5, 6\}$ and then for each such subset we count how many times appears in the twelve baskets. Therefore, we have the following

$$\{1,2\} \longrightarrow 2,\ \{1,3\} \longrightarrow 3,\ \{1,4\} \longrightarrow 2,\ \{1,5\} \longrightarrow 1$$
$$\{1,6\} \longrightarrow 0,\ \{2,3\} \longrightarrow 3,\ \{2,4\} \longrightarrow 4,\ \{2,5\} \longrightarrow 2$$
$$\{2,6\} \longrightarrow 1,\ \{3,4\} \longrightarrow 4,\ \{3,5\} \longrightarrow 4,\ \{3,6\} \longrightarrow 2$$
$$\{4,5\} \longrightarrow 3,\ \{4,6\} \longrightarrow 3,\ \{5,6\} \longrightarrow 2,$$

where again by $\{i, j\} \longrightarrow k$ we mean that the pair $\{i, j\}$ appears in k of the twelve baskets.

(b) Now, by evaluating the hash function $h(i, j) = i \times j \mod 11$ in each of the pairs above we find that

$$h(1,2) = 2, h(1,3) = 3,\ h(1,4) = 4,\ h(1,5) = 5,\ h(1,6) = 6$$
$$h(2,3) = 6, h(2,4) = 8,\ h(2,5) = 10,\ h(2,6) = 1,\ h(3,4) = 1$$
$$h(3,5) = 4,\ h(3,6) = 7,\ h(4,5) = 9,\ h(4,6) = 2,\ h(5,6) = 8.$$

(c) To find the frequent buckets we first count the number of pairs hashed in each of the eleven buckets in the previous step (b) and we sum the supports of these pairs. The buckets having sum greater or equal to 4 are frequent.

Table 1: Number of hashed pairs per bucket.

Bucket	**Number of hashed pairs**
0	0
1	5
2	5
3	3
4	6
5	1
6	3
7	2
8	6
9	3
10	2

By the above we observe that the frequent buckets are *bucket 1, bucket 2, bucket 4* and *bucket 8*.

(d) According to the PCY algorithm a pair $\{i, j\}$ is counted on the second pass if and only if i and j are both frequent, and the pair hashed to a frequent bucket on the first pass. We start by checking each pair of the 1st bucket (which is frequent). The pair $\boxed{(2,6)}$ belongs to bucket 1 and both items 2 and 6 have counts greater or equal than 4. Therefore, the pair $(2,6)$ is counted on the second pass. $\boxed{(3,4)}$ belongs also to the 1st bucket and both 3 and 4 are frequent items. The next frequent bucket is *bucket 2*. $(1,2)$ belongs to *bucket 2* and items $1, 2$ are frequent, therefore $\boxed{(1,2)}$ is also counted on the second pass. $\boxed{(4,6)}$ belongs to *bucket 2* and both $4, 6$ are frequent, hence the pair $(4,6)$ is also counted on the second pass. Moving onto the *bucket 4* we observe that both $\boxed{(3,5)}$ and $\boxed{(1,4)}$ satisfy the condition for being counted on the second pass. Finally, for *bucket 8* we observe that both the pairs $\boxed{(2,4)}$, $\boxed{(5,6)}$ are counted on the second pass. Summarizing all the above we obtain that the pairs

$$(2,6), (3,4), (1,2), (4,6)$$
$$(3,5), (1,4), (2,4), (5,6)$$

are the ones to be counted on the second pass of PCY algorithm.

Exercise 6.3.2 - MMDS

Suppose we run the Multistage Algorithm on the data of Exercise 6.3.1, with the same support threshold of 4. The first pass is the same as in that exercise, and for the second pass, we hash pairs to nine buckets, using the hash function that hashes $\{i, j\}$ to bucket $i + j \mod 9$. Determine the counts of the buckets on the second pass. Does the second pass reduce the set of candidate pairs? Note that all items are frequent, so the only reason a pair would not be hashed on the second pass is if it hashed to an infrequent bucket on the first pass.

Solution From exercise 6.3.1 we have that the pairs

$$(2,6), (3,4),\ (1,2),\ (4,6) \qquad (1.2.1)$$
$$(3,5), (1,4),\ (2,4),\ (5,6)$$

satisfy the following condition:

Both i,j are frequent and the pair (i,j) is hashed to a frequent bucket in the first pass. (1.2.2)

Now, the second pass of the Multistage algorithm uses another hash function to these pairs. In this case we use $h(i, j) = i + j \mod 9$. By simple calculations we find

$$h(2,6) = 8,\ h(3,4) = 7,\ h(1,2) = 3,\ h(4,6) = 1$$
$$h(3,5) = 8,\ h(1,4) = 5,\ h(2,4) = 6,\ h(5,6) = 2.$$

Therefore, we get the following table

Table 2: Number of hashed pairs per bucket on the second pass.

Bucket	**Number of hashed pairs**
0	0
1	3
2	2
3	2
4	0
5	2
6	4
7	4
8	5

By the previous table we observe that the frequent buckets on the second pass are *bucket 6, bucket 7* and *bucket 8*. Now, the conditions that a pair (i, j) has to satisfy in order to be considered as frequent are the following:

(i) Both i, j have to be frequent.

(ii) The pair (i, j) must belong to a frequent bucket on the first pass.

(iii) The pair (i, j) must belong to a frequent bucket on the second pass.

The pairs in (1.2.1) that belong to one of the *buckets* 6,7 or 8 are $(2, 6)$, $(3, 4)$ and $(2, 4)$. Since the above conditions include the condition described in (1.2.2) of the PCY algorithm then the second pass will reduce the set of candidate pairs.

Exercise 6.4.1 - MMDS

Suppose there are eight items, $\{A, B, C, D, E, F, G, H\}$ and the following are the maximal frequent itemsets: $\{A, B\}, \{B, C\}, \{A, C\}, \{A, D\}, \{E\}$, and $\{F\}$. Find the negative border.

Solution Since the itemset $\{A, B\}$ is a maximal frequent set then each superset of $\{A, B\}$ will not be frequent. Similarly for the sets $\{B, C\}, \{B, C\}, \{A, C\}, \{A, D\}, \{E\}$ and $\{F\}$. In particular, the set $\{A, B, C\}$ is not frequent. By observing that all of its subsets are frequent we conclude that the set $\{A, B, C\}$ is in the negative border.

Also, the singletons $\{G\}, \{H\}$ are in the negative border since all of their subsets (only the emptyset $\emptyset$) are frequent. Now, since the singletons $\{E\}$, $\{F\}$ are frequent if we append any of the items A, B, C, D that are frequent we will obtain an itemset which will be in the negative border. Hence, the itemsets $\{E, A\}$, $\{E, B\}$, $\{E, C\}$, $\{E, D\}$ and $\{F, A\}$, $\{F, B\}$, $\{F, C\}$, $\{F, D\}$ will lie in the negative border as well. Also, the fact that both $\{E\}, \{F\}$ are frequent the itemset $\{E, F\}$ will be in the negative border as well.

Now, the itemset $\{B, D\}$ is also in the negative border. Indeed, it is easily seen that all of its subsets are frequent. To show that $\{B, D\}$ is not frequent we argue as follows: If $\{B, D\}$ was a frequent itemset, then it would contained in some maximal frequent itemset I. By looking all of the maximal frequent itemsets we observe that none of them contains the itemsets B, D. Therefore, $\{B, D\}$ is in the negative border. By substituting B with C the same argument shows that $\{D, C\}$ is in the negative border as well.

Summarizing all the above we conclude that the itemsets that belong to the negative border are

$$\{A, B, C\}, \{E, A\}, \{E, B\}, \{E, C\}, \{E, D\}, \{B, D\}, \{E, F\},$$
$$\{F, A\}, \{F, B\}, \{F, C\}, \{F, D\}, \{G\}, \{H\}, \{D, C\}.$$

Exercise 2

(a) Comment on A-priori's algorithm correctness for the problem of frequent itemset mining for market basket data. Which is the number of passes to the database?

Answer: The correctness and the effectiveness of the A-priori algorithm relies on the following observation which is known as *monotonicity for itemsets*.

$$\text{If a set } I \text{of items is frequent, then so is every subset of } I. \tag{2.1.1}$$

To prove the above claim we let $J \subseteq I$. Then every basket that contains all the items in I surely contains all the items in J. Thus, the count for J must be at least as great as the count for I, and if the count for I is at least s, then the count for J is at least s. Since J may be contained in some baskets that are missing one or more elements of $I \setminus J$, it is entirely possible that the count for J is strictly greater than the count for I.

In addition to making the A-Priori Algorithm work, monotonicity offers us a way to compact the information about frequent itemsets. If we are given a support threshold s, then we say an itemset is maximal if no superset is frequent. If we list only the maximal itemsets, then we know that all subsets of a maximal itemset are frequent, and no set that is not a subset of some maximal itemset can be frequent. In the A-Priori Algorithm, one pass is taken for each set-size k. If no frequent itemsets of a certain size are found, then monotonicity tells us there can be no larger frequent itemsets, so we can stop. Therefore, if there are no frequent itemsets of size k then the algorithm does k passes to the database.

(b) Examine and discuss whether the time complexity of A-priori's algorithm with respect to the input and output is polynomial or not.

Answer: For the time complexity with respect to the input, suppose our database consists of n unique items, say $\{1, \ldots, n\}$. Then, the powerset of $\{1, \ldots, n\}$ consists of 2^n elements. Suppose now, that in the worst case every subset of $\{1, \ldots, n\}$ is frequent. Then in this case the algorithm has time complexity $O(2^n)$ with respect to the input and hence the complexity is non polynomial. On the other hand, when it comes to the output of the algorithm, the complexity is $O(m \cdot n^2)$ where m is equal to the number of frequent itemsets. As stated above, in the worst case m can be exponentially large. But in many practical occasions the number of the frequent itemsets falls into the category of the polynomial order. Therefore, we conclude that with respect to the output, the complexity is not polynomial in general but it can be in many practical occasions.

(c) Discuss about the input and output complexity of the FP-Growth algorithm.

Answer: The previous discussion for the case of the A-priori algorithm holds for the case of FP-Growth algorithm. In both cases, if the items is the set $\{1, \ldots, n\}$ then the number of every possible subset of $\{1, \ldots, n\}$ is still 2^n. Hence, if there are 2^n frequent itemsets (repeated identical transactions) then, the time complexity of FP-Growth's algorithm will still be $O(2^n)$ with respect to the input. Same reasoning holds also in the case of the algorithm's output.

(d) Suppose you are given the following transaction database

$$\begin{array}{rl} \{a,c,d\} & \{a,c\} \\ \{a,d\} & \{a,d\} \\ \{b,c,d\} & \{b,d\} \\ \{b,d\} & \{a,b,d\} \\ \{a,c\} & \{c,d\} \\ \{a,b,c,d\} & \{b,c\} \end{array} \tag{2.1.2}$$

.

Apply the A-priori and FP-Growth algorithms in the above example with support threshold $s = 4$.

Solution: *(A-Priori)* We first begin with the *A-priori* algorithm. In the first step we evaluate the support of the items $\{a, b, c, d\}$. By simple calculations we derive the following table.

Table 3: The support of $1-$itemsets

Item	**Support**
a	7
b	6
c	7
d	9

From Table 3 we observe that all $1-$itemsets are frequent since all of their supports exceed the threshold value 4. On the second pass of the A-Priori algorithm we form all of the $2-$itemsets with both elements being frequent and we count their support. In the table below we see the result.

Table 4: The support of $2-$itemsets

Itemset	**Support**
{a,b}	2
{a,c}	4
{a,d}	4
{b,c}	3
{b,d}	5
{c,d}	4

From Table 4 we observe that the non frequent $2-$itemsets are $\{a, b\}$, $\{b, c\}$. On the third pass we form all the $3-$itemsets with the property that all of their subsets are frequent. In the table below we see these $3-$ itemsets and their respective supports.

Table 5: The support of $3-$itemsets

Itemset	**Support**
{a,c,d}	2
{b,c,d}	2

Table 5 shows that there are no 3-frequent itemsets and hence, the algorithm terminates. Summarizing all the above we conclude that

$$C_1 = \big\{\{a\},\{b\},\{c\},\{d\}\big\},\ F_1 = C_1 \qquad \text{(1st pass)}$$

$$C_2 = \big\{\{a,b\},\ \{a,c\},\ \{a,d\},\ \{b,c\},\ \{b,d\},\ \{c,d\}\big\}$$
$$F_2 = \big\{\{a,c\},\ \{a,d\},\ \{b,d\},\ \{c,d\}\big\} \qquad \text{(2nd pass)}$$

$$C_3 = \big\{\{a,c,d\},\ \{b,c,d\}\big\},\ F_3 = \emptyset. \qquad \text{(3rd pass)}$$

(FP-Growth) Now, we run the *FP-Growth* algorithm onto the same transaction database. The algorithm proceeds as follows: We first find the support of all $1-$itemsets in our transactions (same as in *A-Priori*) and then we sort the table in descending order. Sorting Table 3 in descending order we obtain the following table.

Table 6: The support of $1-$itemsets in descending order

Item	**Support**
d	9
a	7
c	7
b	6

Now, the second step is to construct the *FP-tree*. For this, create the root of the tree. The root is represented by null. The next step is to scan the database again and examine the transactions. Examine the first transaction and find out the itemset in it. The itemset with the max count is taken at the top, the next itemset with lower count and so on. It means that the branch of the tree is constructed with transaction itemsets in descending order of count. Also, the count of the itemset is incremented as it occurs in the transactions. Both the common node and new node count is increased by 1 as they are created and linked according to transactions. In Figure 1 we see the resulting *FP-tree* by taking into account only the first transaction $\{a,c,d\}$.

Figure 1: FP-tree in transaction 1

By examining the second transaction in order, i.e. $\{a, d\}$, the tree becomes

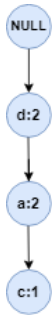

Figure 2: FP-tree in transaction 2

With the third transaction in order, i.e. $\{b, c, d\}$ the tree becomes

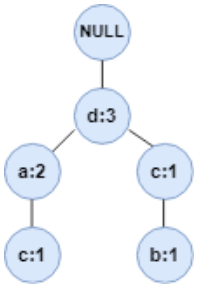

Figure 3: FP-tree in transaction 3

Taking into account the fourth transaction $\{b, d\}$ we obtain

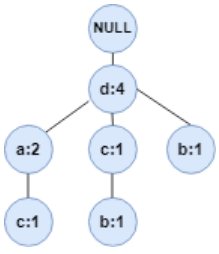

Figure 4: FP-tree in transaction 4

The next transaction is $\{a, c\}$, therefore we obtain

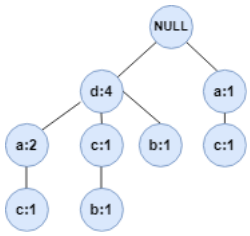

Figure 5: FP-tree in transaction 5

Continuing this way and adding the last transaction of the first column and the transactions of the second column in (2.1.2) we conclude that the complete *FP-tree* has the form

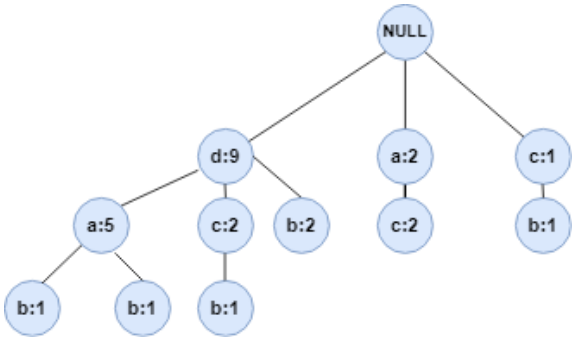

Figure 6: The complete FP-tree

Having constructed the *FP-tree* the next step is to mine it. For this, the lowest node is examined first along with the links of the lowest nodes. The lowest node represents the frequency pattern length 1. From this, traverse the path in the *FP-tree*. This path or paths are called a *conditional pattern base*. Conditional pattern base is a sub-database consisting of prefix paths in the *FP-tree* occurring with the lowest node (suffix). In more detail, to mine the *FP-tree* we proceed as follows: The lowest node item is b. For this node the *conditional pattern base* will be

$$\Big\{\{d, a : 1\},\ \{d, a, c : 1\},\ \{d, c : 1\},\ \{d : 2\},\ \{c : 1\}\Big\}.$$

From the above *conditional pattern base* we construct the conditional *FP-tree* of node b. By discarding the items we count less than the minimum support threshold we obtain the following *conditional FP-tree* of node b.

$$\langle d : 5, c : 2\rangle,\ \langle c : 1\rangle.$$

Now, making joins with item b and excluding the joints that does not meet the requirement condition we obtain the following frequent itemsets containing b

$$\Big\{\{d, b : 6\}\Big\}.$$

In similar manner, by starting with the item c as the leaf node we find that the *conditional pattern base* is given by

$$\Big\{\{d, a : 2\}, \{d : 2\}, \{a : 2\}\Big\}.$$

The conditional *FP-tree* is given by

$$\langle d : 4, a : 2\rangle, \langle a : 2\rangle.$$

By making joins with c and eliminating the items that do not meet the threshold condition we obtain the following *generated frequent patterns*.

$$\Big\{\{d, c : 4\},\ \{a, c : 4\}\Big\}.$$

Now, the next item is a which generates the *conditional pattern base* $\{d : 5\}$, therefore the only frequent pair generated is $\{d, a : 5\}$. Therefore, the frequent itemsets are the following

$$\{a\},\ \{b\},\ \{c\},\ \{d\},\ \{a, c\},\ \{a, d\},\ \{b, d\},\ \{c, d\}, \tag{2.1.3}$$

which is the same result as in the case of the *A-priori* algorithm.

(e) Execute *Toivonen*'s algorithm for the same example for the first three entries of the database. Perform two runs of the algorithm using as threshold $s' = 1$ and $s'' = 2$. What do you observe?

Answer: The first three entries of the transaction database are $\{a, c, d\},\ \{a, d\},\ \{b, c, d\}$.

Case 1: $s' = 1$ Using *A-Priori* algorithm and the minimum support threshold $s' = 1$ we find the frequent itemsets of the sub-database $\{a, c, d\},\ \{a, d\},\ \{b, c, d\}$. In the table below we see the singletons and their respective supports.

Table 7: The support of $1-$itemsets in the sample

Item	**Support**
a	2
b	1
c	2
d	3

Since the support threshold is $s' = 1$ we observe that all $1-$itemsets are frequent in the sample. Below we form all $2-$itemsets and we count the support for these pairs.

Table 8: The support of $2-$itemsets

Itemset	**Support**
{a,b}	0
{a,c}	1
{a,d}	2
{b,c}	1
{b,d}	1
{c,d}	2

By Table 8 we observe that all frequent 2-itemsets are $\{a,c\}$, $\{a,d\}$, $\{b,c\}$, $\{b,d\}$, $\{c,d\}$. Below we see the supports of the $3-$itemsets.

Table 9: The support of $3-$itemsets in the sample

Itemset	**Support**
{a,c,d}	1
{b,c,d}	1

By Table 9 we observe that both $\{a,c,d\}$ and $\{b,c,d\}$ are frequent. Now we find the negative border of the sample. Since both $\{a,c,d\}$ and $\{b,c,d\}$ are frequent and there are no $4-$itemsets in the sample we conclude that the negative border cannot be larger than these sets. Also, since all of the subsets of these two $3-$itemsets are frequent we conclude that the only possible $2-$itemset for being frequent is $\{a,b\}$. By observing that all of the subsets of $\{a,b\}$ we conclude that $\{a,b\}$ is in the negative border, in fact is the only set in the negative border. By looking at the whole transaction database and by table 4 we conclude that $\{a,b\}$ has support 2 and thefore since it does not exceed the value of 4 we conclude that $\{a,b\}$ is non frequent on the whole sample. Therefore, the itemsets

$$\{a,c\},\{a,d\},\ \{b,c\},\ \{b,d\},\ \{c,d\}$$
$$\{a\},\{b\},\ \{c\},\ \{d\},\ \{a,c,d\},\ \{b,c,d\},$$

are the possible candidates for being frequent in the whole sample. To terminate the algorithm we find the support of all these itemsets and determine whether these itemsets are frequent or not. By simple count or from Table 4 we find that the

$$\{a\},\ \{b\},\ \{c\},\ \{d\},\ \{a,c\},\ \{a,d\},\ \{b,d\},\ \{c,d\}, \tag{2.1.4}$$

are the frequent itemsets of the whole sample which are in accordance with the result in (2.1.3).

Case 2: $s'' = 2$ Now, in this case by Table 8 and 9 we observe that the frequent itemsets in the sample are $\{a,d\}$, $\{c,d\}$, $\{a\}$, $\{c\}$, $\{d\}$. In this case the itemsets that belong to the negative

border are $\{b\}$ and $\{a, c\}$. In this case we observe that $\{b\}$ is frequent in the whole dataset. Thus, we can give no answer at this time and must repeat the algorithm with a new random sample or with a smaller support threshold.

Exercise 3

Examine at least two algorithms with at least three different datasets from the Frequent Itemset Mining Implementation Repository http://fimi.uantwerpen.be. Give a short description of your implementations and present thoroughly your results.

In this exercise we study two algorithms for mining frequent itemsets; the *PatriciaMine* algorithm presented in paper [2] by Andrea Pietracaprina and Dario Zandolin, and the *LCM* (*Linear time Closed item set Miner*) algorithm presented in a paper by Takeaki Uno, Masashi Kiyomi and Hiroki Arimura [3]. Moreover, we make use of the source code of these algorithms which is available in http://fimi.uantwerpen.be repository and we execute the algorithms in each of the following three datasets: 1) T10I4D100K, 2) chess and 3) mushroom dataset. The *LCM* algorithm as being subsequent of the *PatriciaMine* algorithm has better performance in all datasets. Before we present the performance of the algorithms we give a short description of the key features of the aforementioned algorithms.

Several algorithms proposed in the literature to discover all frequent itemsets follow a depth-first approach by considering one item at a time and generating (recursively) all frequent itemsets which contain that item, before proceeding to the next item. A prominent member of this class of algorithms is FP-Growth. It represents the dataset D through a standard trie (FP-tree) and, for each frequent itemset X, it materializes a projection D_X of the dataset on the transactions containing X, which is used to recursively discover all frequent supersets $Y \supset X$. This approach is very effective for dense datasets, where the trie achieves high compression, but becomes space inefficient when the dataset is sparse, and incurs high costs due to the frequent projections. *PatriciaMine* is an algorithm that also follows a depth-first approach. The authors made improvements upon previous algorithms stemmed from FP-Growth in two aspects: a) In the space required for executing the algorithm and b) in the running time of the algorithm through several optimizations regarding the implementation of the algorithm. For the former aspect, the authors used a compressed (Patricia) trie to store the dataset which provides a space-efficient representation for both sparse and dense datasets, without resorting to two alternatives structures, namely array-based and trie-based. As for the latter, they utilized a heuristic technique to limit the number of physical projections of the dataset during the course of execution, with the intent to avoid the time and space overhead incurred by projection, when not beneficial.

On the other hand, LCM algorithms are based on backtracking algorithms, and use an efficient techniques for the frequency counting. LCM algorithms compute the frequency efficiently without keeping previously obtained itemsets in memory compared to the classical scheme of the a-priori algorithms. The main difference of the LCM algorithm compared to the algorithms that utilize tree-based techniques is that the LCM uses array structures instead of trees. This is because the LCM does not have to search transactions in the database. Another benefit of using array-based structures instead of trees is the computational cost of the initialization of an FP-tree. For example, if we denote by $\mathcal{I} = \{1, \ldots, n\}$ the set of items and by $\mathcal{T} = \{t_1, \ldots, t_m\}$ the transaction database; then the construction of the tree takes $O(\|\mathcal{T}\| + |\mathcal{T}| \log |\mathcal{T}|)$ time, where $\|\mathcal{T}\|$ is the sum of sizes of all transactions. Compared to this, LCM detects the identical

transactions and stores the database in memory within linear time of the database size. This is because LCM uses radix sort for this task, which sorts the transactions in a lexicographic order in linear time. In general, the datasets of data mining problems have many transactions, and each transaction has few items. Thus, $\|\mathcal{T}\|$ is usually smaller than $|\mathcal{T}| \log |\mathcal{T}|$, and LCM has an advantage. The constant factors of the computation time of binary tree operations are relatively larger than that of array operations. LCM also has an advantage at this point. Again, we recall that LCM never search the transactions, so each operation required by LCM can be done in constant time.

The Mushroom Dataset

The mushroom dataset is one of the datasets prepared by Roberto Bayardo from the UCI datasets and PUMSB. It contains a total of 119 unique items and consists of 8124 transactions. In Table 10 we see the performance of the two algorithms with respect to the support threshold. The time is measured in seconds.

Table 10: The performance of the Algorithms on Mushroom dataset

Support	**PatriciaMine**	**LCM**
200	23.913	2.622
300	6.484	0.676
400	4.878	0.46
500	1.827	0.213
600	1.151	0.161
700	0.795	0.118
800	0.741	0.098
900	0.259	0.049
1000	0.233	0.043

From Table 10 we observe that the LCM algorithm has better performance than the PatriciaMine by being in almost cases one order of magnitude faster. Of course, the execution time is disproportional to the number of threshold since for lower values of threshold there are more frequent itemsets that have to be taken into account. In Figure 9 we see the graph of the execution time of the algorithms with respect to the support threshold. As we can see PatriciaMine is much slower than the LCM for low values of the support threshold and for higher values both algorithms exhibit almost the same behavior.

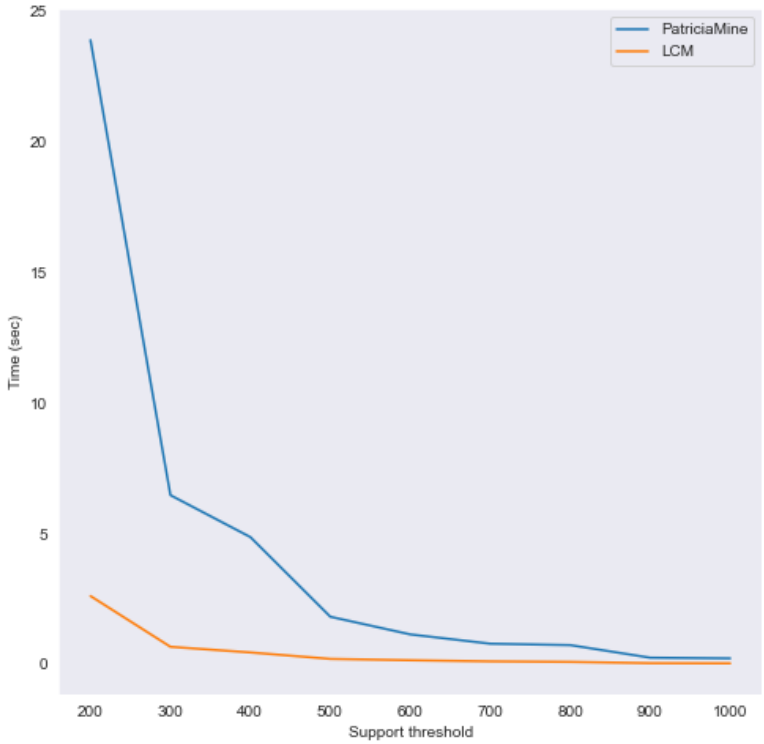

Figure 7: Performance of the algorithms on Mushroom dataset

The T10I4D100K Dataset

The T10I4D100K dataset contains 870 unique items and 100000 transactions. T10I4D100K is an artificial dataset which was created from a generator by the IBM Almaden Quest research group. In Table 12 we summarize the results for the two algorithms.

Table 11: The performance of the Algorithms on T10I4D100K dataset

Support	**PatriciaMine**	**LCM**
5	3.278	1.534
10	1.649	0.564
15	1.504	0.433
20	1.393	0.369
25	1.31	0.335
30	1.264	0.315
35	1.108	0.316
40	1.078	0.3
45	1.147	0.287
50	1.119	0.283
55	1.098	0.276
60	1.071	0.252

In the following Figure you can see the graph of the experimental results shown in Table 12.

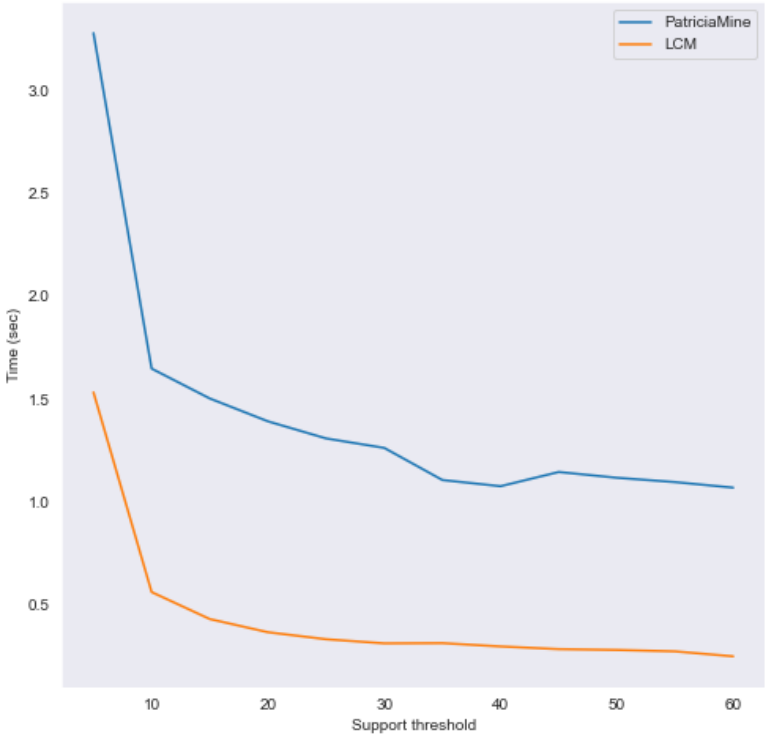

Figure 8: Performance of the algorithms on T10I4D100K dataset

The Chess dataset

Finally, the last dataset that we tested the two algorithms on is the Chess dataset which consists of 3196 transactions and 75 items. In the table below you can see the execution time of the algorithms with respect to the threshold.

Table 12: The performance of the Algorithms on Chess dataset

Support	**PatriciaMine**	**LCM**
1500	2.92	0.484
1600	1.608	0.247
1700	0.94	0.182
1800	0.584	0.111
1900	0.369	0.084
2000	0.255	0.059

Again in this case, the LCM algorithm is one order of magnitude faster than the PatriciaMine algorithm for independently of the support threshold value. In the Figure below you can see the execution times as a function of the support threshold.

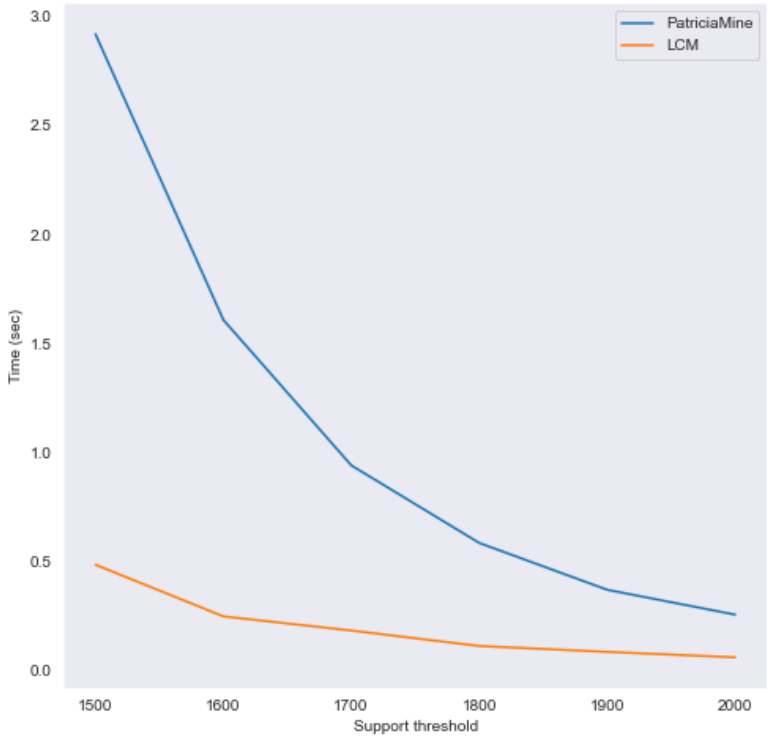

Figure 9: Performance of the algorithms on Chess dataset

Exercise 4

Exercise 5.2.2 - MMDS

Represent the transition matrices of the following graphs:

Figure 10: Figures 5.4 and 5.7 in MMDS

(i) We begin by constructing the transition of the *Web* described by the left graph given in Figure 10. In general, This matrix M has n rows and columns, if there are n pages. The element m_{ij} in row i and column j has value $1/k$ if page j has k arcs out, and one of them is to page i. Otherwise, $m_{ij} = 0$. Below we see the resulting matrix for the left graph in Figure 10.

$$\begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1/2 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 1/2 & 0 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array}$$

Below we see the compact representation of the above matrix.

Source	Degree	Destinations
A	3	B,C,D
B	2	A,D
C	1	E
D	2	B,C

(ii) Repeating the above procedure for graph on the right hand side in Figure 10 we obtain the following transition matrix

$$\begin{matrix} A & B & C \end{matrix}$$
$$\begin{pmatrix} 1/3 & 1/2 & 0 \\ 1/3 & 0 & 1/2 \\ 1/3 & 1/2 & 1/2 \end{pmatrix} \begin{matrix} A \\ B \\ C \end{matrix}$$

and the corresponding representation.

Source	Degree	Destinations
A	3	A,B,C
B	2	A,C
C	2	B,C

Exercise 5.2.3 - MMDS

Using the method of Section 5.2.4 (MMDS), represent the transition matrices of the graph given in Figure 11, assuming blocks have side 2.

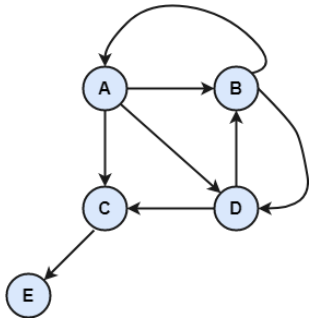

Figure 11: Graph of Figure 5.3 in MMDS

Solution: We first begin by constructing the transition matrix for the above graph. Below we see the resulting matrix.

$$\begin{matrix} A & B & C & D \end{matrix}$$
$$\begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

Dividing the above matrix into blocks of size 2 the matrix becomes

$$M = \left(\begin{array}{c|c} M_{11} & M_{12} \\ \hline M_{21} & M_{22} \end{array}\right) = \left(\begin{array}{cc|cc} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ \hline 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{array}\right),$$

where $M_{11} = \begin{pmatrix} 0 & 1/2 \\ 1/3 & 0 \end{pmatrix}$, $M_{12} = \begin{pmatrix} 0 & 0 \\ 0 & 1/2 \end{pmatrix}$, $M_{21} = \begin{pmatrix} 1/3 & 0 \\ 1/3 & 1/3 \end{pmatrix}$ and $M_{22} = \begin{pmatrix} 0 & 1/2 \\ 0 & 0 \end{pmatrix}$. Now, the upper-left quadrant represents links from A or B to A or B, the upper-right quadrant represents links from C or D to A or B, and so on. The representation of the blocks M_{ij}, $1 \leq i, j \leq 2$ is the following.

Source	Degree	Destinations
A	3	B
B	2	A

(a) Representation of M_{11} connecting A and B to A and B.

Source	Degree	Destinations
D	2	B

(b) Representation of M_{12} connecting C and D to A and B.

Source	Degree	Destinations
A	3	C,D
B	2	D

(c) Representation of M_{21} connecting A and B to C and D.

Source	Degree	Destinations
D	2	C

(d) Representation of M_{22} connecting C and D to C and D.

Exercise 5.3.1 - MMDS

Compute the *topic-sensitive PageRank* of the Web page graph given below,

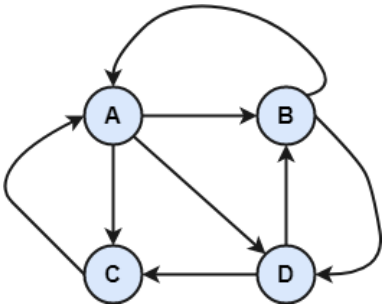

Figure 12: Graph of Figure 5.15 in MMDS

assuming the teleport set is:

(a) A only.

(b) A and C.

Solution: The idea behind the *topic-sensitive PageRank* is to take advantage of the preferences of the users for certain topics. Assuming that the user is interested in a certain topic, then it makes sense to bias the PageRank in favor of pages on that topic. To compute this form of PageRank, we identify a set of pages known to be on that topic, and we use it as a "teleport set." The PageRank calculation is modified so that only the pages in the teleport set are given a share of the tax, rather than distributing the tax among all pages on the Web.

The mathematical formulation for the iteration that yields topic-sensitive PageRank is similar to the equation we used for general PageRank. The only difference is how we add the new surfers. Suppose S is a set of integers consisting of the row/column numbers for the pages we have identified as belonging to a certain topic (the teleport set). Let e_S be a vector that has 1 in the components in S and 0 in other components. Then the *topic-sensitive Page- Rank* for S is the limit of the iteration

$$v' = \beta M v + (1 - \beta)\frac{e_S}{|S|}, \tag{4.3.1}$$

where M is the transition matrix of the Web, β is a chosen constant, usually in the range 0.8 to 0.9 and $|S|$ is the size of set S.

(a) In this case we have $S = \{A\}$, $\beta = 0.8$ and the transition matrix

$$M = \begin{array}{c} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

The vector e_S is $(1, 0, 0, 0)$. If we assume that a surfer has equal probability to land to any of the four pages we begin by estimating the limit vector by using (4.3.1) repeatedly. Therefore, starting with $v_0 = (1/4, 1/4, 1/4, 1/4)$, and using (4.3.1) for 5 iterations we get the following vectors

$$v_1 = \underbrace{\begin{pmatrix} 0.5 \\ 0.166 \\ 0.166 \\ 0.166 \end{pmatrix}}_{\text{Iter 1}}, v_2 = \underbrace{\begin{pmatrix} 0.4 \\ 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}}_{\text{Iter 2}}, v_3 = \underbrace{\begin{pmatrix} 0.44 \\ 0.186 \\ 0.186 \\ 0.186 \end{pmatrix}}_{\text{Iter 3}}, v_4 = \underbrace{\begin{pmatrix} 0.424 \\ 0.192 \\ 0.192 \\ 0.192 \end{pmatrix}}_{\text{Iter 4}}, v_5 = \underbrace{\begin{pmatrix} 0.43 \\ 0.189 \\ 0.189 \\ 0.189 \end{pmatrix}}_{\text{Iter 5}}.$$

In the table below we see the result for the *topic-sensitive PageRank* in the 20th iteration.

Table 13: Topic-sensitive PageRank in 20th iteration, $S = \{A\}$

Web Page	**Rank Value**
A	0.428
B	0.190
C	0.190
D	0.190

(b) Now since the teleport set S consists of the Web pages A and C we have that $e_S = (1, 0, 1, 0)$ and $|S| = 2$. Using the same procedure as before, we obtain the following vectors for the first 5 five iterations.

$$v_1 = \underbrace{\begin{pmatrix} 0.4 \\ 0.166 \\ 0.266 \\ 0.166 \end{pmatrix}}_{\text{Iter 1}}, v_2 = \underbrace{\begin{pmatrix} 0.38 \\ 0.173 \\ 0.273 \\ 0.173 \end{pmatrix}}_{\text{Iter 2}}, v_3 = \underbrace{\begin{pmatrix} 0.388 \\ 0.170 \\ 0.270 \\ 0.170 \end{pmatrix}}_{\text{Iter 3}}, v_4 = \underbrace{\begin{pmatrix} 0.3848 \\ 0.171 \\ 0.271 \\ 0.171 \end{pmatrix}}_{\text{Iter 4}}, v_5 = \underbrace{\begin{pmatrix} 0.368 \\ 0.171 \\ 0.271 \\ 0.171 \end{pmatrix}}_{\text{Iter 5}}.$$

Below we see the resulting vector in the 20th iteration.

Table 14: Topic-sensitive PageRank in 20th iteration, $S = \{A, C\}$

Web Page	**Rank Value**
A	0.385
B	0.171
C	0.271
D	0.171

Exercise 5.4.2 - MMDS

For the Web graph

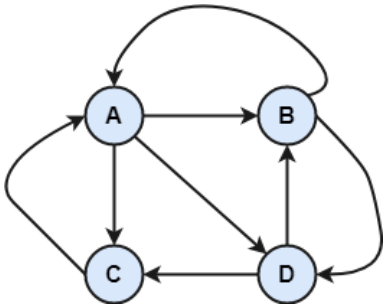

Figure 13: Graph of Figure 5.15 in MMDS

assuming only B is a trusted page:

(a) Compute the TrustRank of each page.

(b) Compute the spam mass of each page.

(a) The *TrustRank* is simply the *topic-sensitive PageRank* where in this case the *teleport* set consists of the trusted pages. Since B is the only trusted page we have that $S = \{A\}$, $e_S = (0, 1, 0, 0)$ and the transition matrix M of the Web page graph is equal to

$$M = \begin{array}{c} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

Since B is the only trusted page our set S consists only of the page B, i.e. $S = \{B\}$ and hence $e_S = (0, 1, 0, 0)$. If we assume that a surfer has equal probability to land to any of the four pages we get that $v_0 = \big(1/4,\, 1/4,\, 1/4,\, 1/4\big)$. Using (4.3.1) with $v' = v_0$ we have that the estimated PageRank vector in the first iteration is

$$\begin{aligned} v_1 &= \frac{4}{5} \begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} + \frac{1}{5} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\ &\approx (0.3,\, 0.366,\, 0.166,\, 0.166)^T. \end{aligned} \tag{4.4.1}$$

Repeating again the same procedure by plugging v_1 to equation (4.3.1) we obtain

$$v_2 = \frac{4}{5}\begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0.3 \\ 0.366 \\ 0.166 \\ 0.166 \end{pmatrix} + \frac{1}{5} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$
$$\approx (0.28,\, 0.346,\, 0.146,\, 0.226)^T. \tag{4.4.2}$$

Continuing this way, and summarizing the results for the first 5 iterations we obtain

$$v_1 = \begin{pmatrix} 0.3 \\ 0.366 \\ 0.166 \\ 0.166 \end{pmatrix},\ v_2 = \begin{pmatrix} 0.28 \\ 0.346 \\ 0.146 \\ 0.226 \end{pmatrix},\ v_3 = \begin{pmatrix} 0.256 \\ 0.365 \\ 0.165 \\ 0.213 \end{pmatrix},\ v_4 = \begin{pmatrix} 0.278 \\ 0.353 \\ 0.153 \\ 0.214 \end{pmatrix},\ v_5 = \begin{pmatrix} 0.264 \\ 0.36 \\ 0.16 \\ 0.215 \end{pmatrix}.$$

In the following table we summarize the results of the TrustRank for each Web page in the 20th iteration.

Table 15: TrustRank per Web Page in the 20th iteration

Web Page	**TrustRank**
A	0.269
B	0.357
C	0.157
D	0.214

(b) *Spam mass* is a combination of *PageRank* and *TrustRank*. In detail, *spam mass* is a calculation that identifies the pages that are likely to be spam and allows the search engine to eliminate those pages or to lower their PageRank strongly. In precise mathematics, if we denote by r the PageRank and by t the TrustRank of a Web page p then the spam mass of the page is given by the ratio $(r-t)/r$. A negative or small positive spam mass means that p is probably not a spam page, while a spam mass close to 1 suggests that the page probably is spam.

To calculate the spam mass for each Web page we need both the PageRank and the TrustRank values. In the previous step we calculated the TrustRank values; the results shown in table 12. Now, since the graph is strongly connected without dead ends we can calculate the PageRank values by simply solving the linear system of equations described by $M \cdot v = v$, where $v = (v_1, v_2, v_3, v_4)$ and $v_1 + v_2 + v_3 + v_4 = 1$. Substituting the transition matrix M we obtain

$$\begin{pmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$$

which is equivalent to

$$\begin{cases} \frac{v_2}{2} + v_3 = v_1 \\ \frac{v_1}{3} + \frac{v_4}{2} = v_2 \\ \frac{v_1}{3} + \frac{v_4}{2} = v_3 \\ \frac{v_1}{3} + \frac{v_2}{2} = v_4 \\ v_1 + v_2 + v_3 + v_4 = 1 \end{cases} . \tag{4.4.3}$$

By the second and third equation we observe that $v_2 = v_3$. Therefore, $v_1 = (3/2)v_2$ and

$$\begin{aligned} v_1 + v_2 + v_3 + v_4 = 1 &\iff \frac{3}{2}v_2 + 2v_2 + v_4 = 1 \\ &\iff \frac{7}{2}v_2 + v_4 = 1. \end{aligned}$$

By the third equation we obtain $v_2 = v_4$. Hence, $v_2 = 2/9$. Therefore, the solution to (4.4.3) which corresponds to the PageRank of the Web is $v = (1/3,\ 2/9,\ 2/9,\ 2/9)$. Now, since we have calculated both the PageRank and the TrustRank of the pages, using the ratio $(r - t)/r$ we obtain the following table.

Table 16: PageRank and its variants per Web page

Web Page	**PageRank**	**TrustRank**	**Spam mass**
A	1/3	0.269	0.193
B	2/9	0.357	-0.6065
C	2/9	0.157	0.2935
D	2/9	0.214	0.037

Exercise 5

Exercise 7.2.2 - MMDS

Use hierarchical clustering to the points in Figure 14 using as distance between two clusters:

(i) The minimum of the distances (*minimum intercluster distance*) between any two points, one from each cluster.

(ii) The average of the distances between pairs of points, one from each of the two clusters.

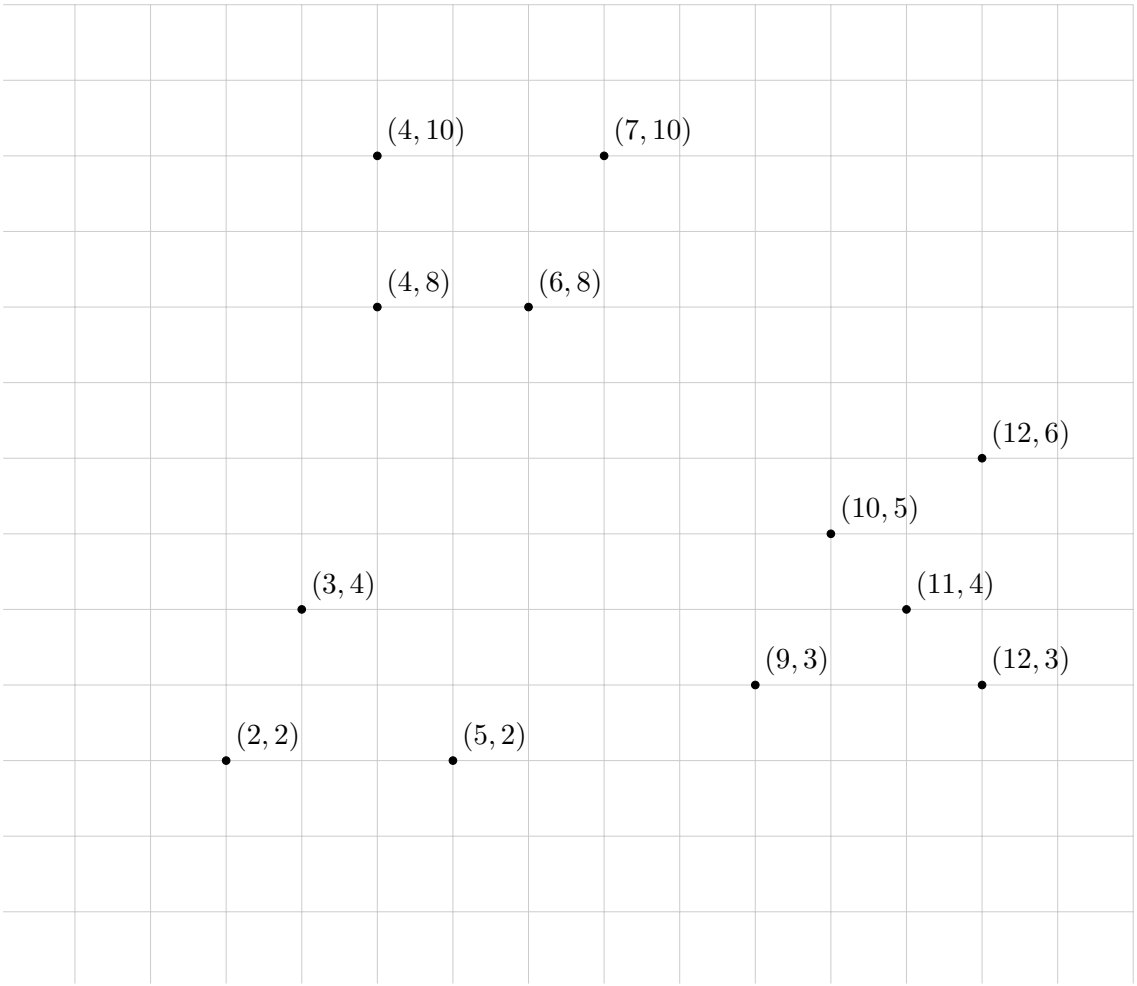

Figure 14: Twelve points to be clustered

Solution: (i) In this case the condition for merging two clusters is with respect to the minimum distances between any two points, one from each cluster. Formally, suppose in some step k of the hierarchical algorithm there are $C_1, \ldots, C_m$ distinct clusters. Then, the minimum distance between any two points, one each from the cluster C_i, C_j is given by the quantity

$$d(C_i, C_j) = \min\{\|x - y\|_2 : x \in C_i, y \in C_j\}, \tag{5.1.1}$$

where by $\|x-y\|_2$ we denote the Euclidean distance between the vectors $x, y \in \mathbb{R}^n$ ($n = 2$ in our case). Then, in step $k+1$ of the algorithm the clusters C_l, C_k are merged into one cluster if and only if

$$d(C_l, C_k) = \min\{d(C_i, C_j) : 1 \leq i, j \leq m,\ l \neq k\}. \tag{5.1.2}$$

Now, in our example in Figure 14, in step 1 the clusters are just the twelve points. Therefore, to merge the clusters in the first iteration we just measure the distances between each pair of points and the pair with the minimum distance forms a new cluster. By looking at Figure 1 we observe that among all pairs of points, there are two pairs that are closest: (10,5) and (11,4) or (11,4) and (12,3). In this case, we could proceed by choosing either of the two pairs. We choose the pair $(10, 5)$ and $(11, 4)$ and we denote the cluster consisting of these two points by $C_1^{(1)}$. In the second iteration we have 11 distinct clusters, 10 consisting only of singletons and the cluster $C_1^{(1)}$. Now, the minimum of the distances between any two points, one from each cluster for any two clusters is again $\sqrt{2}$. By a simple calculation we see that the clusters $C_1^{(1)} = \{(10, 5), (11, 4)\}, \{(12, 3)\}$ have minimum distance $\sqrt{2}$. Indeed, for the points $(11, 4), (12, 3)$ we find that

$$\|(11, 4) - (12, 3)\|_2 = \sqrt{(11-12)^2 + (4-3)^2} = \sqrt{2}.$$

Therefore, in step 2 the clusters $C_1^{(1)}$ and $\{(12, 3)\}$ are merged into one cluster which we denote by $C_1^{(2)}$. In the step 3, the family of clusters contains 10 clusters; 9 of them consisting of singletons and the cluster $C_1^{(2)} = \{(10, 5), (11, 4), (12, 3)\}$. The minimum distance in this case is 2 and it is achieved by the pairs $(4, 8), (6, 8)$ and $(4, 10), (4, 8)$. We choose the pair $(4, 8), (6, 8)$ to form a cluster which we denote by $C_2^{(3)}$. In the step 4 the point $(4, 10)$ appends to $C_2^{(3)}$ and form a new cluster $C_2^{(4)}$. Continuing this way and by ignoring ties the algorithm terminates we all of the twelve points are merged into one cluster. In Figure 15 we see the formation of the clusters with respect to each iteration.

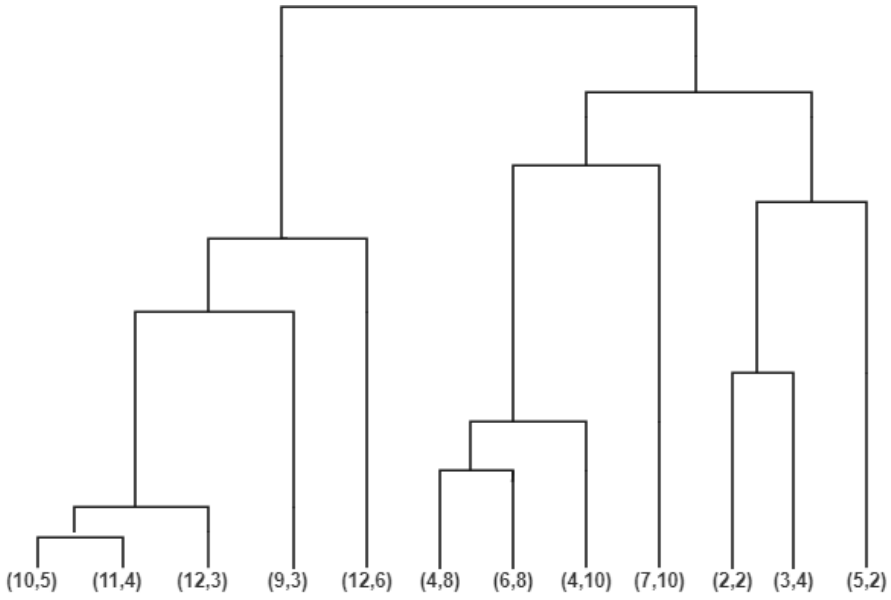

Figure 15: Hierarchical dendrogram with minimum intercluster distance

In Table 17 we see how the clusters merge in each iteration along with their corresponding *minimum intercluster distance*.

Table 17: Formation of clusters with minimum intercluster distance

Iteration	**Clusters**	**Minimum Distance**
1	$C_1^{(1)} = \{(10,5),(11,4)\}$	$\sqrt{2}$
2	$C_1^{(2)} = \{(10,5),(11,4),(12,3)\}$	$\sqrt{2}$
3	$C_1^{(3)} = C_1^{(2)}, C_2^{(3)} = \{(4,8),(6,8)\}$	2
4	$C_1^{(4)} = C_1^{(3)}, C_2^{(4)} = \{(4,8),(6,8),(4,10)\}$	2
5	$C_1^{(5)} = C_1^{(4)}, C_2^{(5)} = C_2^{(4)}, C_3^{(5)} = \{(2,2),(3,4)\}$	$\sqrt{5}$
6	$C_1^{(6)} = C_1^{(5)} \cup \{(9,3)\}, C_2^{(6)} = C_2^{(5)}, C_3^{(6)} = C_3^{(5)}$	$\sqrt{5}$
7	$C_1^{(7)} = C_1^{(6)} \cup \{(12,6)\}, C_2^{(6)} = C_2^{(5)}, C_3^{(6)} = C_3^{(5)}$	$\sqrt{5}$
8	$C_1^{(8)} = C_1^{(7)}, C_2^{(8)} = C_2^{(8)}, C_3^{(8)} = C_3^{(7)} \cup \{(5,2)\}$	3
9	$C_1^{(9)} = C_1^{(8)}, C_2^{(9)} = C_2^{(8)} \cup \{(7,10)\}, C_3^{(9)} = C_3^{(8)}$	3
10	$C_1^{(10)} = C_1^{(9)}, C_2^{(10)} = C_2^{(9)} \cup C_3^{(9)}$	$\sqrt{17}$
11	$C_1^{(11)} = C_1^{(10)} \cup C_2^{(10)}$	$\sqrt{37}$

(ii) Given two clusters C, D the average of the distances between pairs of points, one from each of the two clusters is given by the quantity

$$d_{\text{Av}}(C,D) = \frac{1}{|C||D|}\sum_{x\in C}\sum_{y\in D}\|x-y\|_2. \tag{5.1.3}$$

Then if during the k-th step of the algorithm there are $C_1,\ldots,C_m$ clusters, the condition for merging C_i and C_j is the following

$$d_{\text{Av}}(C_i,C_j) = \min\{d_{\text{Av}}(C_k,C_l) : 1\le k,l\le m,\ k\ne m\}. \tag{5.1.4}$$

Since every cluster is a singleton in the first iteration, (5.1.3) is just the distance between a pair of points. Therefore, the minimum distance in this case is again $\sqrt{2}$ and it is achieved by the pair $(10,5)$ and $(11,4)$ or $(11,4)$ and $(12,3)$. We proceed again with the pair $(10,5)$ and $(11,4)$ and we keep the same notation $C_1^{(1)}$ for the resulting cluster. Now, in the second iteration the point which is closest to the cluster $C_1^{(1)}$ according to (5.1.3) is $(12,3)$ with average distance

$$\begin{aligned} d_{\text{AV}}\big(C_1^{(1)},\{(12,3)\}\big) &= \frac{1}{2}\left(\sqrt{(12-10)^2+(5-3)^2}+\sqrt{(12-11)^2+(4-3)^2}\right)\\ &= \frac{1}{2}\left(\sqrt{4+4}+\sqrt{1+1}\right)\\ &= \frac{1}{2}(\sqrt{8}+\sqrt{2}) = \frac{3}{2}\sqrt{2}\approx 2.12. \end{aligned}$$

By looking at Figure 14 we observe that there are pairs of points with Euclidean distance 2; $(4,8)$ and $(6,8)$ or $(4,8)$ and $(4,10)$. Therefore, in this step either of the two pairs is chosen to form a new cluster. We choose the pair $(4,8)$ and $(6,8)$ and we denote the resulting cluster with

$C_2^{(2)}$. In the third iteration the pair $(4, 10)$ is the closest to the cluster $C_2^{(2)}$ according to (5.1.4) with distance

$$d_{\text{Av}}\big(C_2^{(2)}, \{(4,10)\}\big) = \frac{1}{2}\left(\sqrt{(4-4)^2+(10-8)^2} + \sqrt{(6-4)^2+(10-8)^2}\right)$$
$$= \frac{1}{2}(2+2\sqrt{2}) \approx 1.7.$$

Hence, in the third iteration we obtain $C_2^{(3)} = \big\{(4,8),(6,8),(4,10)\big\}$. Continuing in similar manner for the next iterations the algorithm terminates in the 11-th iteration. In the following table we summarize the results.

Table 18: Formation of clusters per iteration using average distance

Iteration	**Clusters**	**Average Distance**
1	$C_1^{(1)} = \big\{(10,5),(11,4)\big\}$	$\sqrt{2}$
2	$C_1^{(2)} = C_1^{(1)}, C_2^{(2)} = \big\{(4,8),(6,8)\big\}$	2
3	$C_1^{(3)} = C_1^{(2)}, C_2^{(3)} = C_2^{(2)} \cup \{(4,10)\}$	1.7
4	$C_1^{(4)} = C_1^{(3)} \cup \{(12,3)\}, C_2^{(4)} = C_2^{(3)}$	2.12
5	$C_1^{(5)} = C_1^{(4)}, C_2^{(5)} = C_2^{(4)}, C_3^{(5)} = \{(2,2),(3,4)\}$	$\sqrt{5}$
6	$C_1^{(6)} = C_1^{(5)} \cup \{(9,3)\}, C_2^{(6)} = C_2^{(5)}, C_3^{(6)} = C_3^{(5)}$	2.49
7	$C_1^{(7)} = C_1^{(6)}, C_2^{(7)} = C_2^{(6)}, C_3^{(7)} = C_3^{(6)} \cup \{(5,2)\}$	2.91
8	$C_1^{(8)} = C_1^{(7)} \cup \{(12,6)\}, C_2^{(8)} = C_2^{(7)}, C_3^{(8)} = C_3^{(7)}$	2.92
9	$C_1^{(9)} = C_1^{(8)}, C_2^{(9)} = C_2^{(8)} \cup \{(7,10)\}, C_3^{(9)} = C_3^{(8)}$	2.94
10	$C_1^{(10)} = C_1^{(9)}, C_2^{(10)} = C_2^{(9)} \cup C_3^{(9)}$	6.56
11	$C_1^{(11)} = C_1^{(10)} \cup C_2^{(10)}$	

In Figure 16 we see the resulting dendrogram using the average distance.

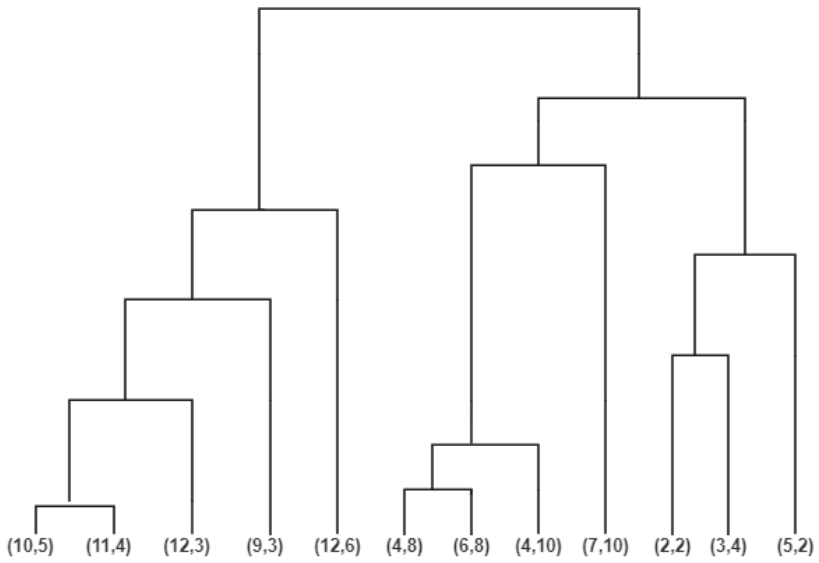

Figure 16: Hierarchical dendrogram with minimum average distance

Exercise 7.3.2 - MMDS

Repeat the clustering of Figure 14 if we choose to merge the two clusters whose resulting cluster has:

(i) The smallest radius.

(ii) The smallest diameter.

(i) The *radius* of a cluster is the maximum distance between all the points and the centroid. In this case we combine the two clusters whose resulting cluster has the lowest radius. Using the same reasoning as in Exercise in 7.2.2 we obtain the following table which gives a summary of the resulting clusters in each iteration.

Table 19: Formation of clusters per iteration using minimum radius

Iteration	**Clusters**	**Radius**
1	$C_1^{(1)} = \{(10,5),(11,4)\}$	$\sqrt{2}/2$
2	$C_1^{(2)} = C_1^{(1)}, C_2^{(2)} = \{(4,8),(6,8)\}$	1
3	$C_1^{(3)} = C_1^{(2)}, C_2^{(3)} = C_2^{(2)}, C_3^{(3)} = \{(2,2),(3,4)\}$	1.11
4	$C_1^{(4)} = C_1^{(3)} \cup \{(12,3)\}, C_2^{(4)} = C_2^{(3)}, C_3^{(4)} = C_3^{(3)}$	$\sqrt{2}$
5	$C_1^{(5)} = C_1^{(4)}, C_2^{(5)} = C_2^{(4)} \cup \{(4,10)\}$	1.49
6	$C_1^{(6)} = C_1^{(5)} \cup \{(12,6)\}, C_2^{(6)} = C_2^{(5)}, C_3^{(6)} = C_3^{(5)}$	1.67
7	$C_1^{(7)} = C_1^{(6)}, C_2^{(7)} = C_2^{(6)}, C_3^{(7)} = C_3^{(6)} \cup \{(5,2)\}$	1.79
8	$C_1^{(8)} = C_1^{(7)}, C_2^{(8)} = C_2^{(7)} \cup \{(7,10)\}, C_3^{(8)} = C_3^{(7)}$	2.01
9	$C_1^{(9)} = C_1^{(8)} \cup \{(9,3)\}, C_2^{(9)} = C_2^{(8)}, C_3^{(9)} = C_3^{(8)}$	2.16
10	$C_1^{(10)} = C_1^{(9)} \cup C_2^{(9)}, C_3^{(10)} = C_3^{(9)}$	5.67
11	$C_1^{(11)} = C_1^{(10)} \cup C_2^{(10)} \cup C_3^{(10)}$	

As we observe from Table 19 the cluster $C^{(1)} = \{(10,5),(11,4)\}$ has radius $\sqrt{2}/2$. Indeed, the centroid of the cluster is the point

$$\left(\frac{10+11}{2}, \frac{5+4}{2}\right) = (11.5, 4.5).$$

Hence, by calculating the distance of the centroid from both points of the cluster we obtain that the radius is equal to

$$\begin{aligned} R &= \max\{\sqrt{(1/2)^2+(1/2)^2}, \sqrt{(1/2)^2+(1/2)^2}\} \\ &= \frac{\sqrt{2}}{2}. \end{aligned}$$

In a similar manner we calculate the radius of the rest of the clusters shown in Table 19. In Figure 17 we see the resulting dendrogram using the as a merging criterion the minimum radius.

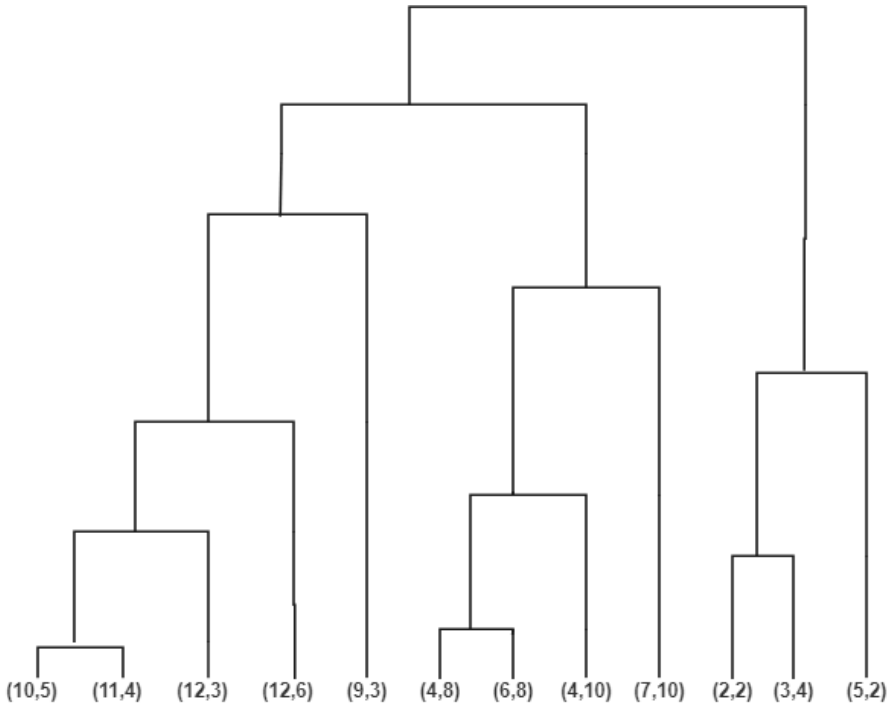

Figure 17: Hierarchical dendrogram using minimum radius

(ii) The *diameter* of a cluster is the maximum distance between any two points of the cluster. In detail, if C is a cluster during some iteration of the algorithm, then its diameter is defined by

$$\text{diam}(C) = \max\big\{\|x-y\|_2 : x, y \in C\big\}. \tag{5.2.1}$$

Then, if $C_1, \ldots, C_m$ are the clusters during the k-th iteration, C_k, C_n are merged into one cluster if and only if $\text{diam}(C_k \cup C_n) = \min\big\{\text{diam}(C_i \cup C_j) : 1 \le i, j \le m,\, i \ne j\big\}$.

Table 20: Formation of clusters per iteration using minimum diameter

Iteration	**Clusters**	**Diameter**
1	$C_1^{(1)} = \{(10,5),(11,4)\}$	$\sqrt{2}$
2	$C_1^{(2)} = C_1^{(1)}, C_2^{(2)} = \{(4,8),(6,8)\}$	2
3	$C_1^{(3)} = C_1^{(2)} \cup \{(9,3)\}, C_2^{(3)} = C_2^{(2)}$	$\sqrt{5}$
4	$C_1^{(4)} = C_1^{(3)}, C_2^{(4)} = C_2^{(3)}, C_3^{(4)} = \{(2,2),(3,4)\}$	$\sqrt{5}$
5	$C_1^{(5)} = C_1^{(4)}, C_2^{(5)} = C_2^{(4)} \cup \{(4,10)\}, C_3^{(5)} = C_3^{(5)}$	$2\sqrt{2}$
6	$C_1^{(6)} = C_1^{(5)}, C_2^{(6)} = C_2^{(5)}, C_3^{(6)} = C_3^{(5)} \cup \{(5,2)\}$	$2\sqrt{2}$
7	$C_1^{(7)} = C_1^{(6)} \cup \{(12,3)\}, C_2^{(7)} = C_2^{(6)}, C_3^{(7)} = C_3^{(6)}$	3
8	$C_1^{(8)} = C_1^{(7)}, C_2^{(8)} = C_2^{(7)} \cup \{(7,10)\}, C_3^{(8)} = C_3^{(7)}$	$\sqrt{13}$
9	$C_1^{(9)} = C_1^{(8)} \cup \{(12,6)\}, C_2^{(9)} = C_2^{(8)}, C_3^{(9)} = C_3^{(8)}$	$3\sqrt{2}$
10	$C_1^{(10)} = C_1^{(9)}, C_2^{(10)} = C_2^{(9)} \cup C_3^{(9)}$	9.43
11	$C_1^{(11)} = C_1^{(10)} \cup C_2^{(10)}$	10.77

In Figure 18 we see the resulting dendrogram using as merging criterion the minimum diameter.

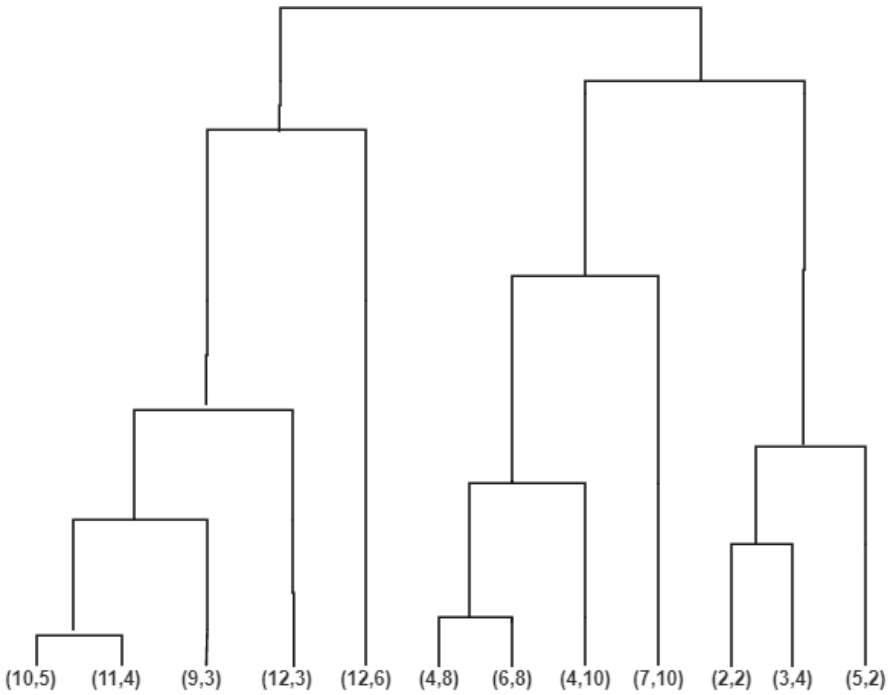

Figure 18: Hierarchical dendrogram using minimum diameter

Exercise 7.3.2 - MMDS

Prove that no matter what point we start with in Figure 14, if we select three starting points by the method of Section 7.3.2 we obtain points in each of the three clusters. *Hint*: You could solve this exhaustively by beginning with each of the twelve points in turn. However, a more generally applicable solution is to consider the diameters of the three clusters and also consider the *minimum intercluster distance*, that is, the minimum distance between two points chosen from two different clusters. Can you prove a general theorem based on these two parameters of a set of points?

Solution: For the sake of completeness, let us describe the method presented in Section 7.3.2 in MMDS. The method can be described by the following algorithm.

Algorithm 1 The Algorithm in Section 7.3.2

```
1: Pick the first point at random;
2: while there are fewer than k points do
3:     Add the point whose minimum distance from the selected
4:         points is as large as possible;
5: end while
```

A theorem concerning only the diameters and the minimum intercluster distances of the clusters

involved could not be formulated during the development of this series of exercises. However, a slightly different condition applicable to the situation in Figure 14 can be formulated. Before we state and prove this general theorem we recall some definitions that we will use throughout the course of this exercise.

Definition 1 (Diameter)**.** Let X be a Euclidean space and $A \subseteq X$ be a cluster[1] in X. The diameter $\text{diam}(A)$ of A is given by the quantity

$$\text{diam}(A) = \max\big\{\|x - y\|_2 : x, y \in A\big\}. \tag{5.3.1}$$

Definition 2 (Minimum Intercluster distance)**.** Let X be a Euclidean space and $A, B \subseteq X$ be two non empty clusters in X. The *minimum intercluster distance* of A, B is given by

$$d(A, B) = \min\big\{\|x - y\|_2 : x \in A, y \in B\big\}. \tag{5.3.2}$$

Now, the theorem that can be applied to the situation in Figure 14 to ensure that Algorithm 1 will generate exactly one element from every cluster is the following.

Theorem 1. *Let $C_1, \ldots, C_m$ be m distinct clusters in some Euclidean space. Suppose that for every $1 \leq i \leq m$ there exists an element $z_i \in C_i$ such that*

$$d(C_j, \{z_i\}) > \max\big\{\textit{diam}(C_j) : 1 \leq j \leq m,\, j \neq i\big\}. \tag{5.3.3}$$

Then, if we run Algorithm 1 to the clusters $C_1, \ldots, C_m$, no matter in which cluster the starting point would be; the Algorithm will always terminate by generating exactly one point from each cluster.

Before we prove Theorem 1 let us first show that the condition in (5.3.3) is satisfied in the setting of Figure 14. We define $C_1 = \big\{(9,3), (10,5), (11,4), (12,3), (12,6)\big\}$, $C_2 = \big\{(2,2), (3,4), (5,2)\big\}$ and $C_3 = \big\{(4,8), (6,8), (7,10), (4,10)\big\}$ for our three clusters. Then by simple calculations we obtain $\text{diam}(C_1) \approx 4.24$, $\text{diam}(C_2) \approx 3$ and $\text{diam}(C_3) = 3.60$. Now, for the first cluster C_1 we fix the point $z_1 = (12, 3)$. Then,

$$\begin{aligned} d(C_2, \{z_1\}) &= \min\big\{\|x - z_1\|_2 : x \in C_2\big\} \\ &= \sqrt{(12-5)^2 + (3-2)^2} \\ &= \sqrt{50} = \sqrt{51} \approx 7.07. \end{aligned}$$

Therefore, $d(C_2, \{z_1\}) > \textit{diam}(C_3) = \max\big\{\text{diam}(C_j) : 1 \leq j \leq 3,\, j \neq 1\big\}$. Similarly,

$$\begin{aligned} d(C_3, \{z_1\}) &= \min\big\{\|x - z_1\|_2 : x \in C_2\big\} \\ &= \sqrt{(12-6)^2 + (3-8)^2} \\ &= \sqrt{36 + 25} = \sqrt{51} \approx 7.14. \end{aligned}$$

[1] In general, the diameter of a subset $A \subseteq X$ is given by $\text{diam}(A) = \sup\big\{\|x - y\|_2 : x, y \in A\big\}$

And it easily seen that $d(C_3, \{z_1\}) > \max\{\text{diam}(C_j) : 1 \leq j \leq 3,\, j \neq 1\}$. Now, for the cluster C_2 we pick the point $z_2 = (2, 2)$. Then, in similar manner we calculate

$$\begin{aligned} d(C_1, \{z_2\}) &= \min\{\|x - z_2\|_2 : x \in C_1\} \\ &= \sqrt{(2-9)^2 + (2-3)^2} \\ &= \sqrt{49+1} = \sqrt{50} \approx 7.07. \end{aligned}$$

And it is readily seen that (5.3.3) is satisfied for $j = 1$ and $i = 2$. Now,

$$\begin{aligned} d(C_3, \{z_2\}) &= \min\{\|x - z_2\|_2 : x \in C_3\} \\ &= \sqrt{(2-4)^2 + (2-8)^2} \\ &= \sqrt{4+36} = \sqrt{40} \approx 6.32, \end{aligned}$$

and obviously (5.3.3) is satisfied for $j = 3$ and $i = 2$. Finally, for the cluster C_3 we choose the point $z_3 = (4, 10)$. Then,

$$\begin{aligned} d(C_1, \{z_3\}) &= \min\{\|x - z_3\|_2 : x \in C_1\} \\ &= \sqrt{(4-10)^2 + (10-5)^2} \\ &\sqrt{36+25} = \sqrt{51} \approx 7.14, \end{aligned}$$

which implies that $d(C_1, \{z_3\}) > \max\{\text{diam}(C_j) : 1 \leq j \leq 3,\, j \neq 3\}$. Finally,

$$\begin{aligned} d(C_2, \{z_3\})) &= \min\{\|x - z_3\|_2 : x \in C_x\} \\ &= \sqrt{(4-3)^2 + (10-4)^2} \\ &= \sqrt{1+36} = \sqrt{37} \approx 6.08, \end{aligned}$$

and (5.3.3) is satisfied in the case where $j = 2, i = 3$ and $z_i = z_3$. Therefore, provided that Theorem 1 holds; if we run Algorithm 1 to the clusters C_1, C_2, C_3 then the output will be a set of points x_1, x_2, x_3 such that (after possible re-indexing) $x_1 \in C_1, x_2 \in C_2$ and $x_3 \in C_3$. Now, we proceed by proving Theorem 1.

Proof of Theorem 1. Let $C_1, C_2, \ldots C_m$ be m distinct clusters in some Euclidean space satisfying (5.3.3). Suppose we run Algorithm 1 on the set $C_1 \cup C_2 \cdots \cup C_m$. Then, by its structure the algorithm will terminate after m iterations and its output will be a set of points $x_1, x_2, \ldots, x_m$ from the union of C_i's. Our claim is that

$$\big|C_i \cap \{x_1, x_2, \ldots, x_m\}\big| = 1 \text{ for every } 1 \leq i \leq m, \tag{5.3.4}$$

where by $|A|$ we denote the cardinal number of A. Equivalently, we claim that every x_i will belong to exactly one of the clusters $C_i, 1 \leq i \leq n$. Suppose for the sake of contradiction that (5.3.4) is not true. Furthermore, suppose that the points $x_1, x_2, \ldots x_m$ are generated by the algorithm in the order indicated by their indices. The fact that (5.3.4) is false implies that there exists some $1 \leq k \leq m$ such that

$$\big|C_k \cap \{x_1, x_2, \ldots, x_m\}\big| \geq 2. \tag{5.3.5}$$

Let x_i, x_j be two points belonging to the set C_k. Suppose without loss of generality, that $i < j$; i.e. x_i is generated before x_j. Now, since the number of the points generated by the algorithm is equal to the number of clusters C_i, there will exist a cluster C_n such that

$$C_n \cap \{x_1, x_2, \ldots, x_m\} = \emptyset. \tag{5.3.6}$$

Our claim is that the algorithm should have generated a point $z \neq x_j$ during its jth iteration. Indeed, let z_n be the point of C_n that satisfies (5.3.3); i.e.

$$d(C_l, \{z_n\}) > \max\big\{\text{diam}(C_l) : 1 \leq l \leq m, l \neq n\big\}. \tag{5.3.7}$$

for every $1 \leq l \leq m$, $l \neq n$. We will prove that z_n is a better candidate than x_j according to the condition 3 of Algorithm 1 during its jth iteration. The only thing we have to show is that the minimum distance of z_n from the set $\{x_1, \ldots, x_{j-1}\}$ is strictly larger than the minimum distance of the point x_j from the same set. The mathematical expression of this is described by the inequality

$$d\big(\{x_1, \ldots, x_{j-1}\}, \{z_n\}\big) > d\big(\{x_1, \ldots, x_{j-1}\}, \{x_j\}\big). \tag{5.3.8}$$

Since $x_i, x_j \in C_k$ we have that the right hand side of (5.3.8) is lower or equal than $\|x_i - x_j\|_2$; i.e.

$$d\big(\{x_1, \ldots, x_{j-1}\}, \{x_j\}\big) \leq \|x_i - x_j\|_2,$$

but the fact that $x_i, x_j \in C_k$ implies that $\|x_i - x_j\|_2 \leq \text{diam}(C_k)$ and hence

$$d\big(\{x_1, \ldots, x_{j-1}\}, \{x_j\}\big) \leq \text{diam}(C_k). \tag{5.3.9}$$

Now, fix some $1 \leq l \leq j-1$. Now, every x_l belongs to one of the clusters $C_1, \ldots, C_m$ excluding C_n. In other words, there exists $l_k \in \{1, \ldots, m\} \setminus \{n\}$ such that $x_l \in C_{l_k}$. Then, since $x_l \in C_{l_k}, z_n \in C_n$ by (5.3.7) it follows that

$$\begin{aligned} \|x_l - z_n\|_2 &\geq d(C_{l_k}, \{z_n\}) \\ &> \max\big\{\text{diam}(C_l) : 1 \leq l \leq m, l \neq n\big\}. \end{aligned}$$

Since the above inequality holds for every l and the right hand side is just a constant it follows that

$$\min_{1 \leq l \leq j-1} \|x_l - z_n\|_2 > \max\big\{\text{diam}(C_l) : 1 \leq l \leq m, l \neq n\big\}. \tag{5.3.10}$$

Combining (5.3.9) and (5.3.10) we get

$$\begin{aligned} \min_{1 \leq l \leq j-1} \|x_l - z_n\|_2 &> \max\big\{\text{diam}(C_l) : 1 \leq l \leq m, l \neq n\big\} \\ &\geq \text{diam}(C_k) \\ &\geq d\big(\{x_1, \ldots, x_{j-1}\}, \{x_j\}\big). \end{aligned}$$

Finally, by observing that $\min_{1 \leq l \leq j-1} \|x_l - z_n\|_2$ is just another expression of $d\big(\{x_1, \ldots, x_{j-1}\}, \{z_n\}\big)$ we have that (5.3.7) is true and hence we conclude that z_n is a better candidate than x_j. □

References

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2020.

[2] Andrea Pietracaprina. “Mining frequent itemsets using patricia tries”. In: (2003).

[3] Takeaki Uno, Masashi Kiyomi, Hiroki Arimura, et al. “LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets”. In: *Fimi*. Vol. 126. 2004.