
UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Data Mining and Machine Learning

# ABuddy

Workgroup project documentation

Pietro Tempesti, Benedetta Tessa

Contents

You can find the application code at this link:

https://github.com/PieTempesti98/ABuddy_App

# Design

## The application

Worsening mental health has become an increasingly concerning issue during the pandemic, but recent studies have shown that peer-to-peer support has been nothing but beneficial to young students.

This kind of therapy is based on sharing similar experiences, providing emotional support and advice and is frequently used in psychology to deal with stress, depression and anxiety or even preventing them.

Despite it not being able to fully replace a therapy session with an expert, it can still be extremely helpful.

What we aim to do is connect people so that they build a relationship based on empathy which may lead to a mutual mental health improvement.

We want to cluster individuals based on the answer of a quick survey so that their similar experiences may help build trust and feeling more comfortable with each other.

ABuddy is as a matter of fact an abbreviation for the slang "accountabilibuddy", a term which indicates a friend you love so much, you hold yourself accountable for their well-being.

When a student opens ABuddy for the first time, the application requires the submission of a form with questions about sleeping and studying habits before and after the pandemic.

Afterwards, the new user must wait the formation of groups, periodically started by the application admin.

The groups will be established by a clustering algorithm, which leverages on the form's answers to collect students with similar issues according to their submissions.

Clustered users can communicate with their peers using a bulletin-board system, in which they can share simple messages, like their contact information or meeting proposals.

If a user is not satisfied of the assigned group, the application offers the possibility to be re-clustered, allowing the unsatisfied student to find new buddies.

## Requirements

### Main actors

The application is meant to be used by four different types of actors:

- *Unregistered users, who have only the possibility to sign up.*
- *Non-clustered users, who can submit or modify their form while they wait to be assigned to a group.*

- *Clustered users, who can modify their form and communicate with their group buddies using the bulletin-board. Also, they can express their satisfaction about the group.*
- *Administrator, who can perform the grouping process.*

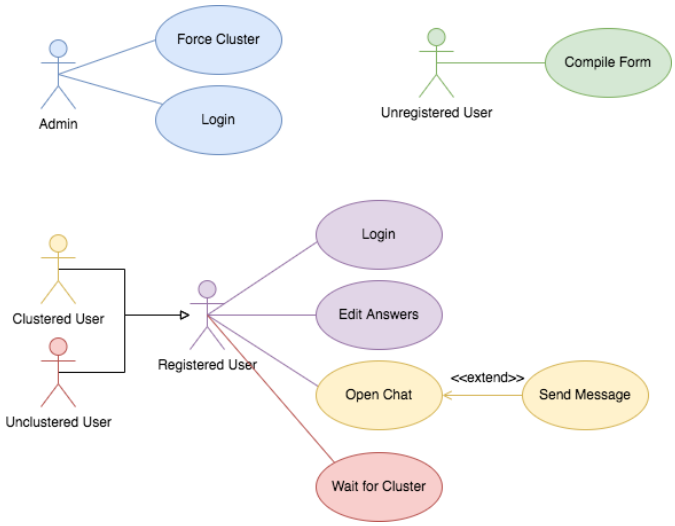
## Functional requirements

- *Registration through a form*
- *Possibility to change the answers given*
- *Possibility to chat with the members of your group*
- *Possibility to leave the group if the user is not satisfied with the current one and be re-clustered again*

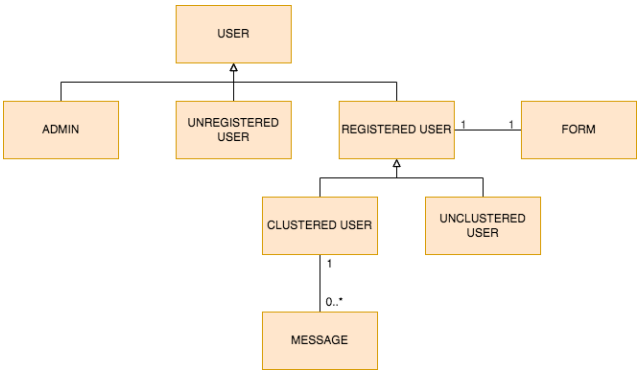
## Non-functional requirements

- *The application must implement at least one machine learning model*
- *Usability: the application must be user friendly, with a simple and intuitive user interface*
- *Robustness: the application must be robust and be able to handle unexpected behaviours*

## Use cases




## Analysis classes



A ***User*** can either be *Admin, Registered* and *Unregistered.*

***Admin***: is characterized by a mail and a password. These two are stored in a xml file therefore the admin does not require any registration.

***Registered***: is characterized by their full name, a mail, a password, a username and the cluster they've been assigned to (Clustered User) or -1 if they still haven't been clustered (Unclustered User). It goes without saying that an unclustered user can become a clustered user, but it can also happen the opposite if a person is not satisfied with their group.

***Unregistered***: there are no fields related to them. They can become registered user only after having registered.

**Form**: contains the answers a user has given. It does not need to be complied right after the registration to the application and can be changed multiple times if an user changes their mind during therapy.

***Message***: messages can be exchanged only among clustered users and are characterized by the sender, the text and the time it was sent.

## Data Model

To store all the information, we use a relational database, since we deal with data characterized by a fixed and predictable structure.

We make use of 3 tables: user, form and message.

### Tables

*Table User:*
It represents both clustered and unclustered users and we differentiate than through the *clusterId* attribute, which is -1 by default and it's later changed after the admin starts the cluster.

- *username (varchar)* ***primary key***
- *fullname (varchar)*
- *email (varchar)* ***unique***
- *password (varchar)*
- *clusterId (int)*

*Table Form:*
Contains one entry for each user that submitted the form. Has a foreign key on the "username" attribute of the table "user"

- *username (string)* ***primary key***
- *gender (int)*
- *university (string)*
- *age (int)*

- *level (int)*
- *year (int)*
- *question11 (int)*
- *question12 (int)*
- *question13 (int)*
- *question14 (int)*
- *question15 (int)*
- *question16 (int)*
- *question17 (int)*
- *question18 (int)*
- *question19 (int)*
- *question20 (int)*
- *question21 (int)*
- *question22 (int)*
- *question23 (int)*
- *question24 (int)*
- *question25 (int)*
- *question26 (int)*
- *question27 (int)*
- *question28 (int)*
- *question29 (int)*
- *question30 (int)*
- *question31 (int)*
- *question32 (int)*
- *question33 (int)*
- *question34 (int)*
- *question35 (int)*
- *question36 (int)*
- *question37 (int)*

*Table Message:*
Contains the messages exchanged in a group. Has a foreign key on the "username" attribute of the table "user"

- *id (int)* ***primary key***
- *username (string)*
- *clusterId (int)*
- *timeSent (lomg)*

## Architecture

The application is client-server.

On the client side run both the presentation and logic layer.

The first is the user interface that allows the user to interact with the application and the latter collects all the user requests and sends them to the server side and viceversa.

The server on other hand is represented by the database that needs to be queried in order to get data

It is fully implemented in Java and also makes use of:

- *JavaFX to implement an easy graphic user interface*
- *Maven to manage the building of the application*
- *JDBC Drivers to query the database*
- *A .jar file found at [https://github.com/Theldus/KValid.git](https://github.com/Theldus/KValid.git) to find the optimal k for the SimpleKMeans algorithm*

# Machine Learning

## Dataset

https://data.mendeley.com/datasets/thnzm3yk23/1

This dataset is a result of a survey given to the students of the University of Jordan to further investigate the impact of online learning to student's mental health

It is made of two main sections; the first contains demographic information and the second reports on the psychosomatic impact of COVID-19's e-learning digital tools on university students' well-being.

The same survey was proposed to our friends and colleagues as well.

The final dataset has 37 features, both ordinal and categorical.

## Preprocessing

### Data Integration

The first thing that had to be done in the preprocessing phase was the Data Integration, since we had to add the answer collected from family and friends to the original dataset.

This was achieved by creating a Google form identical to the one designed for the Jordanian students and only had to add the corresponding answer to the corresponding column.

We couldn't get any value for the "Completion Time" attribute, but this didn't represent a problem since it's not relevant for our application.

For the Ids, we started the assignment by incrementing by one the last Id of the original dataset.

In the application every user has to login or sign in with an email and a username and since for privacy reason we could not make use of real emails, they were both randomly generated thanks to https://randomuser.me/ .

### Feature Selection

For this step we didn't use any attribute selection method (the learning is unsupervised) nor the Principal Component Analysis (our dataset contains mixed type attributes)

Therefore, we personally identified the attributes that shouldn't be taken into account to perform the clustering step.

These attributes are:

- *Id*

- *Start Time*
- *Completion Time*
- *Name*
- *Email*

### Handling of Missing Values

The dataset contained missing values and the strategy we went for was to replace them with their mean if the attribute is numeric or mode if it's categorical.

### Handling of Ordinal Values and Normalization

The majority of the attributes have possible values based on the Likert's five points scale (Strongly Disagree, Disagree, Uncertain, Agree, Strongly Agree) and therefore are ordinal.

As a matter of fact, the distance between Strongly Agree and Strongly Disagree is much higher compared to the one between Strongly Agree and Agree.

Even the attributes Level/Year, Age, cumulative average and "Before/After COVID19: How many hours do you spend using the digital tools you usually use per day?" are ordinal values, and we have a specific scale for each of these.

We handled them in two steps:

1. Assign a rank to each value
    - Strongly Disagree : 1
    - Disagree : 2
    - Uncertain : 3
    - Agree : 4
    - Strongly Agree : 5

2. Normalize onto a [0,1] interval
    - Strongly Disagree : 0
    - Disagree : 0.25
    - Uncertain : 0.5
    - Agree : 0.75
    - Strongly Agree : 1

## Clustering Evaluation

To achieve our goal, we tested our dataset against many clustering algorithms.

What we didn't take into account is the time taken to build the model since for our purposes it's irrelevant.

The admin will, as a matter of fact, cluster one every two weeks so we don't need a fast clustering algorithm.

## KMeans

We used weka with the extention Kvalid to find the optimal k to give in input to the algorithm.

We tried different initialization methods and here's the results:

| *INITALIZATION METHOD* | MIN SQUARED ERROR | NUM CLUSTERS |
|---|---|---|
| *Random* | 1340.700320618931 | 26 |
| *Kmeans ++* | 1183.1068721997253 | 48 |
| *Canopy* | 1224.757585595374 | 30 |
| *Farther First* | 1580.5897155672574 | 13 |

For each initialization method we can see that the min squared error is high, but we decide to choose SimpleKMeans with kmeans++ since it's the one with the lowest value.

With this consideration the best number of clusters is k = 48.

## DBscan

We tried DBscan both with default parameters (minPts=6, epsilon=0.9) and minPts=2 and epsilon=0.9.

*Default Parameters:*

```
0        34 ( 26%)
1        54 ( 42%)
2        16 ( 12%)
3         5 (  4%)
4         9 (  7%)
5         6 (  5%)
6         6 (  5%)

Unclustered instances: 734
```

*minPts = 2 and epsilon = 0.9*

```
 0         3 (  1%)        10         3 (  1%)        20         2 (  1%)
 1         2 (  1%)        11         2 (  1%)        21         5 (  2%)
 2         2 (  1%)        12         2 (  1%)        22         2 (  1%)
 3        13 (  6%)        13        10 (  5%)        23         2 (  1%)
 4        38 ( 18%)        14         3 (  1%)        24         2 (  1%)
 5        61 ( 29%)        15         2 (  1%)        25         2 (  1%)
 6         6 (  3%)        16         2 (  1%)        26         3 (  1%)
 7        16 (  8%)        17         3 (  1%)        27         2 (  1%)
 8         2 (  1%)        18         2 (  1%)        28         4 (  2%)
 9         2 (  1%)        19         9 (  4%)        29         6 (  3%)

Unclustered instances: 651
```

As we can see, most of the instances are not being clustered and treated as noise, which makes DBscan not suitable for our application.

## EM

We tried this algorithm both without giving any parameter in input and with forcing the number of clusters at 48.

Here's the results:

| *NUM CLUSTERS* | LOG LIKELIHOOD |
|---|---|
| *Default (3)* | 4.65438 |
| *48* | 9.7165 |

The latter seems to be the best approach.

## Hierarchical Clustering

We perform the hierarchical clustering with the numClusters = 48 using four different linkages: single, complete and ward and average.

*Single linkage:*

| | | | | | |
|---|---|---|---|---|---|
| 0 | 814 ( 94%) | 16 | 1 ( 0%) | 32 | 1 ( 0%) |
| 1 | 2 ( 0%) | 17 | 1 ( 0%) | 33 | 1 ( 0%) |
| 2 | 1 ( 0%) | 18 | 1 ( 0%) | 34 | 1 ( 0%) |
| 3 | 1 ( 0%) | 19 | 1 ( 0%) | 35 | 1 ( 0%) |
| 4 | 1 ( 0%) | 20 | 1 ( 0%) | 36 | 1 ( 0%) |
| 5 | 1 ( 0%) | 21 | 1 ( 0%) | 37 | 1 ( 0%) |
| 6 | 1 ( 0%) | 22 | 1 ( 0%) | 38 | 1 ( 0%) |
| 7 | 1 ( 0%) | 23 | 1 ( 0%) | 39 | 1 ( 0%) |
| 8 | 1 ( 0%) | 24 | 1 ( 0%) | 40 | 1 ( 0%) |
| 9 | 1 ( 0%) | 25 | 1 ( 0%) | 41 | 1 ( 0%) |
| 10 | 1 ( 0%) | 26 | 1 ( 0%) | 42 | 1 ( 0%) |
| 11 | 1 ( 0%) | 27 | 1 ( 0%) | 43 | 1 ( 0%) |
| 12 | 1 ( 0%) | 28 | 1 ( 0%) | 44 | 1 ( 0%) |
| 13 | 2 ( 0%) | 29 | 1 ( 0%) | 45 | 2 ( 0%) |
| 14 | 1 ( 0%) | 30 | 1 ( 0%) | 46 | 1 ( 0%) |
| 15 | 1 ( 0%) | 31 | 1 ( 0%) | 47 | 1 ( 0%) |

*Complete linkage:*

| | | | | | |
|---|---|---|---|---|---|
| 0 | 97 ( 11%) | 16 | 6 ( 1%) | 32 | 1 ( 0%) |
| 1 | 5 ( 1%) | 17 | 4 ( 0%) | 33 | 24 ( 3%) |
| 2 | 45 ( 5%) | 18 | 52 ( 6%) | 34 | 3 ( 0%) |
| 3 | 55 ( 6%) | 19 | 18 ( 2%) | 35 | 2 ( 0%) |
| 4 | 9 ( 1%) | 20 | 3 ( 0%) | 36 | 1 ( 0%) |
| 5 | 10 ( 1%) | 21 | 18 ( 2%) | 38 | 3 ( 0%) |
| 6 | 44 ( 5%) | 22 | 8 ( 1%) | 39 | 4 ( 0%) |
| 7 | 1 ( 0%) | 23 | 24 ( 3%) | 40 | 12 ( 1%) |
| 8 | 29 ( 3%) | 24 | 18 ( 2%) | 41 | 18 ( 2%) |
| 9 | 17 ( 2%) | 25 | 7 ( 1%) | 42 | 6 ( 1%) |
| 10 | 19 ( 2%) | 26 | 2 ( 0%) | 43 | 1 ( 0%) |
| 11 | 71 ( 8%) | 27 | 8 ( 1%) | 44 | 1 ( 0%) |
| 12 | 10 ( 1%) | 28 | 2 ( 0%) | 45 | 2 ( 0%) |
| 13 | 99 ( 11%) | 29 | 5 ( 1%) | 46 | 2 ( 0%) |
| 14 | 55 ( 6%) | 30 | 10 ( 1%) | 47 | 4 ( 0%) |
| 15 | 23 ( 3%) | 31 | 6 ( 1%) | | |

*Average linkage:*

| | | | | | |
|---|---|---|---|---|---|
| 0 | 663 ( 77%) | 8 | 2 ( 0%) | 16 | 2 ( 0%) |
| 1 | 15 ( 2%) | 9 | 7 ( 1%) | 17 | 6 ( 1%) |
| 2 | 1 ( 0%) | 10 | 3 ( 0%) | 18 | 2 ( 0%) |
| 3 | 2 ( 0%) | 11 | 5 ( 1%) | 19 | 3 ( 0%) |
| 4 | 13 ( 2%) | 12 | 3 ( 0%) | 20 | 1 ( 0%) |
| 5 | 7 ( 1%) | 13 | 4 ( 0%) | 21 | 1 ( 0%) |
| 6 | 6 ( 1%) | 14 | 2 ( 0%) | 22 | 5 ( 1%) |
| 7 | 16 ( 2%) | 15 | 4 ( 0%) | 23 | 3 ( 0%) |

| | | | | | |
|---|---|---|---|---|---|
| 24 | 4 ( 0%) | 33 | 1 ( 0%) | 41 | 1 ( 0%) |
| 25 | 2 ( 0%) | 34 | 1 ( 0%) | 42 | 1 ( 0%) |
| 26 | 1 ( 0%) | 35 | 1 ( 0%) | 43 | 2 ( 0%) |
| 27 | 1 ( 0%) | 36 | 45 ( 5%) | 44 | 1 ( 0%) |
| 28 | 1 ( 0%) | 37 | 14 ( 2%) | 45 | 1 ( 0%) |
| 29 | 1 ( 0%) | 38 | 3 ( 0%) | 46 | 1 ( 0%) |
| 31 | 1 ( 0%) | 39 | 2 ( 0%) | 47 | 1 ( 0%) |
| 32 | 1 ( 0%) | 40 | 1 ( 0%) | | |

*Ward linkage:*

| | | | | | |
|---|---|---|---|---|---|
| 0 | 29 ( 3%) | 16 | 8 ( 1%) | 32 | 15 ( 2%) |
| 1 | 4 ( 0%) | 17 | 9 ( 1%) | 33 | 9 ( 1%) |
| 2 | 26 ( 3%) | 18 | 22 ( 3%) | 34 | 25 ( 3%) |
| 3 | 36 ( 4%) | 19 | 11 ( 1%) | 35 | 19 ( 2%) |
| 4 | 24 ( 3%) | 20 | 17 ( 2%) | 36 | 5 ( 1%) |
| 5 | 29 ( 3%) | 21 | 50 ( 6%) | 37 | 3 ( 0%) |
| 6 | 27 ( 3%) | 22 | 23 ( 3%) | 38 | 10 ( 1%) |
| 7 | 9 ( 1%) | 23 | 36 ( 4%) | 39 | 2 ( 0%) |
| 8 | 46 ( 5%) | 24 | 9 ( 1%) | 40 | 7 ( 1%) |
| 9 | 33 ( 4%) | 25 | 5 ( 1%) | 41 | 14 ( 2%) |
| 10 | 38 ( 4%) | 26 | 11 ( 1%) | 42 | 2 ( 0%) |
| 11 | 7 ( 1%) | 27 | 18 ( 2%) | 43 | 6 ( 1%) |
| 12 | 25 ( 3%) | 28 | 4 ( 0%) | 44 | 5 ( 1%) |
| 13 | 24 ( 3%) | 29 | 6 ( 1%) | 45 | 27 ( 3%) |
| 14 | 23 ( 3%) | 30 | 50 ( 6%) | 46 | 6 ( 1%) |
| 15 | 29 ( 3%) | 31 | 7 ( 1%) | 47 | 14 ( 2%) |

As we can see from the cluster assignments, single linkage and average linkage tend to assign almost every user in only one cluster, leaving few people in the others. Complete linkage and Ward linkage distribute the assignments in a more uniform way across all the clusters.

## Conclusions

Among all the criteria that need to be considered, we thought that the number of items in the same cluster has a certain relevance.

This is because the studies the application are based on state that a smaller group of people is easier to handle and will favour the creation of empathy and trustworthy relationship among the members, unlike bigger groups.

This is the reason why we decided not to deal with the single and average linkage in the hierarchical approach since they will generate clusters containing up to 800 members

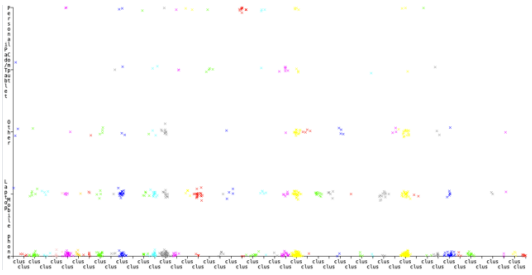
We've decided not to consider DBScan because of the high noise in its results, and finally we focus on the comparison between the best results for each model.

Among the different approaches with KMeans we decided to use the kmeans++ initializer, which has the minimum minSquared error.

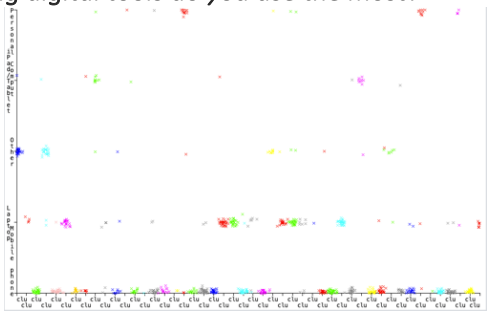
For the EM approach we chose the model with numClusters = 48 in input because of the higher log likelihood.

In the light of these considerations, the 4 approaches we decided to compare are SimpleKMeans with k = 48, EM with numClusters = 48, Hierarchical Clustering with complete and ward linkage type cutting the dendogram at numClusters = 48.

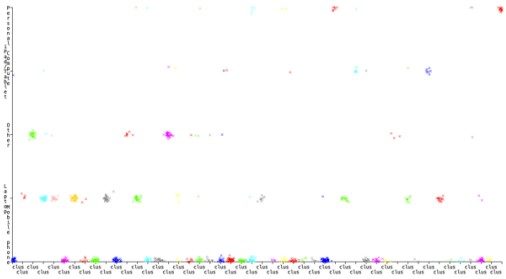
*Before COVID-19: which of the following digital tools do you use the most?*



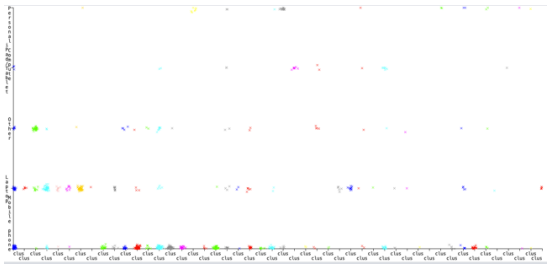
EM



KMEANS

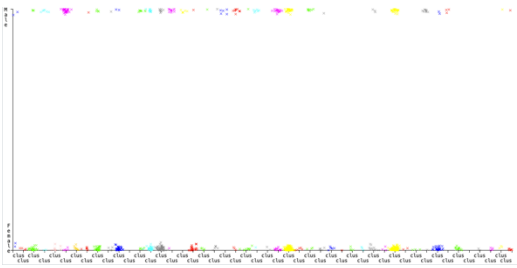


HIERARCHICAL – WARD

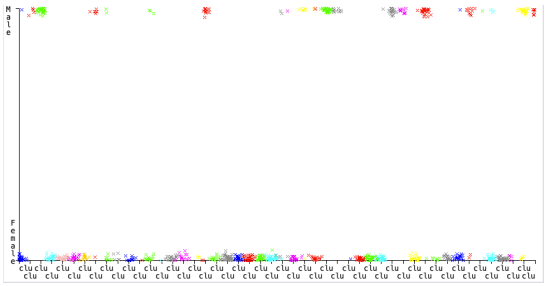


HIERARCHICAL - COMPLETE

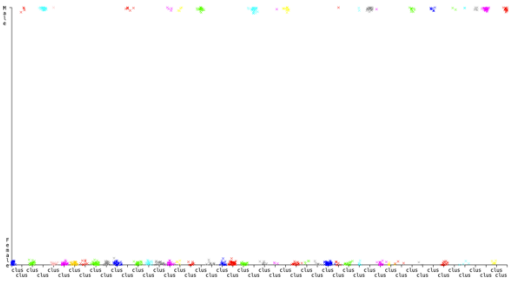
*Gender*



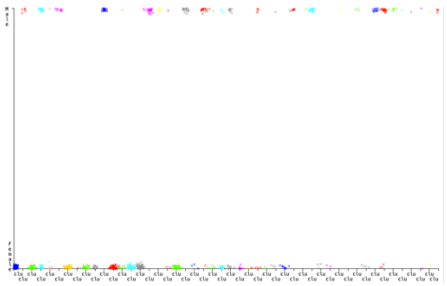
EM



KMEANS

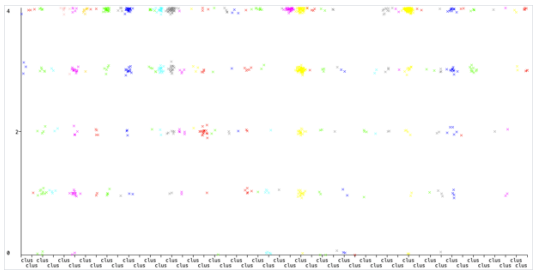


HIERARCHICAL – WARD

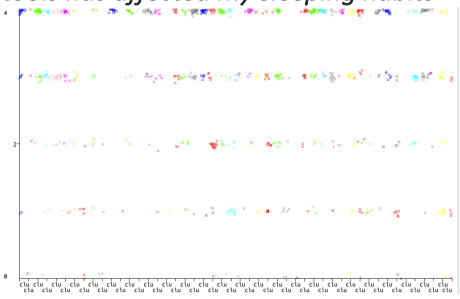


HIERARCHICAL - COMPLETE

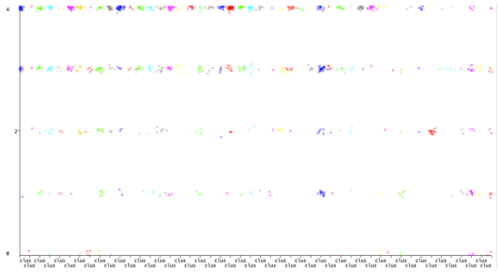
*After COVID-19: prolonged use of digital tools has affected my sleeping habits*



EM



KMEANS

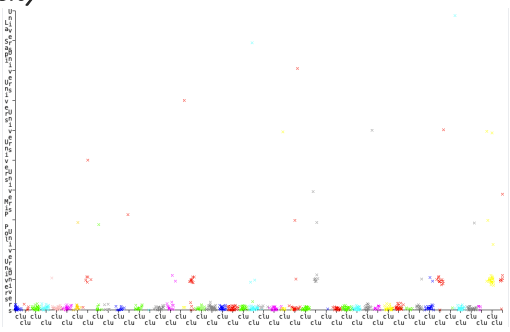


HIERARCHICAL – WARD

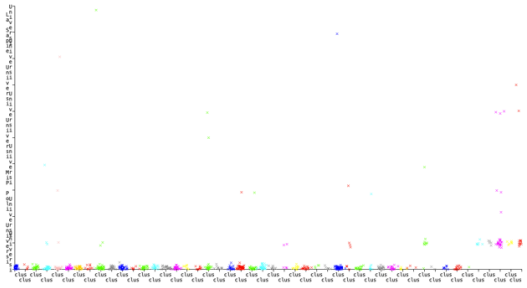
HIERARCHICAL - COMPLETE

*University*

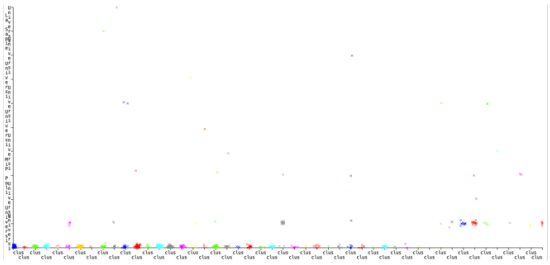
EM



KMEANS



HIERARCHICAL – WARD



HIERARCHICAL - COMPLETE

These pictures show that EM has a worse performance compared to the others.

As we can see, in many clusters there are people who either Agree or Strongly Agree on a certain matter.

Although these are two different answers, they both express a very similar opinion and clustering people who share the similar amount of agreement (even if the answer is slightly different) won't be a menace to the creation on empathy and trust.

This does not happen for all the clusters, and for many users this could be an obstacle and there is the possibility that many people won't be satisfied with the group.

There is also to say that in many cases this could be an asset just as psychotherapist Hillary Jacob Senders states.

It would be interesting to further investigate the percentage of groups that regards dissimilarity as an asset rather than an obstacle.

In conclusion, SimpleKMeans and both Hierarchical Clustering methods seem to have similar results and we couldn't identify a method significantly better than the other.

Eventually, our final decision was to opt for SimpleKMeans, since thanks to the KValid extensions, whenever there will be the need to cluster the optimal k will be found and we don't need to stick to a fixed value of k.

Having the opportunity to find a new k each time is crucial since the number of people to cluster will be always different.

# Implementation

## Main packages and classes

The application consists in one module organized in packages containing one or more classes

### it.unipi.dii.dmml.abuddy.abuddy_application

HelloApplication: the main class that starts the application

### it.unipi.dii.dmml.abuddy.abuddy_application.controller

Contains the controller handling specific .fxml files

- *AdminController: manages the admin page on which he can start a new clustering*
- *AnswerController: manages the loading and submit of the form*
- *ChatController: manages the page which allows the user to exchange messages and leave the group*
- *FormController: manages the registration to the application before compiling the form*
- *HelloController: manages the desire of the user to login/compile the form*
- *LoginController: manages the login of the user which may lead to the chat page if the user is assigned to a group or to a waiting page if it's not the case*
- *WaitingController: manages the page the user is sent to if it's not been assigned to a group yet.*

### it.unipi.dii.dmml.abuddy.abuddy_application.dto

Contains the classes used for the data model

- *User: the registered user making use of the application*
- *Answer: the answers given to the form*
- *Message: the message exchanged among users*

### it.unipi.dii.dmml.abuddy.abuddy_application.clustering

Contains the class used to perform the clustering of the users

- *KValid_Clusterer: retrieves the dataset, performs the preprocessing, the clustering and the group assignments*

### it.unipi.dii.dmml.abuddy.abuddy_application.config

Contains the classes used for initializing the configuration parameters and setting the session when a user logs in

- *ParameterConfig: reads configuration data from an xml file that can't be changed during the application. The data are than used to configure the access to the database*
- *InfoSession: stores the information of the session such as the user or whether the admin is logged or not*

### it.unipi.dii.dmml.abuddy.abuddy_application.utilities

Contains the classes which handle the most common operations and functions to query the database

- ***DatabaseUtils:*** *handles the database connections and performs CRUD operations on the MySQL database*
- ***Utils:*** *handles the change of scene after a button is clicked for all the controllers*

### it.unipi.dii.dmml.abuddy.abuddy_application.kv

Contains the package KValid mentioned in the 'Architecture' paragraph with some modifications

- ***KValid:*** *implements the KValid model used in the clustering*

### it.unipi.dii.dmml.abuddy.abuddy_application.kv.kvalid

Contains the classes used for tuning the optimal value of k

- ***SilhouetteIndex:*** *implements the Silhouette Coefficient in order to find the optimal K*
- ***ClusterEvaluator:*** *interface used for cluster evaluation*