University of Pisa

Personality Clustering

Data Mining and Machine Learning

Cecchetti Jacopo

Terranova Franco

Academic Year 2021/2022

Indice

1. Project idea

Many contemporary personality psychologists believe that there are five basic dimensions of personality, often referred to as the "Big 5" personality traits. The five broad personality traits described by the theory are extraversion, agreeableness, openness, conscientiousness and neuroticism. It is important to note that each of the five personality factors represents a range between two extremes. In the real world, in fact, most people lie somewhere in between the two polar ends of each dimension.

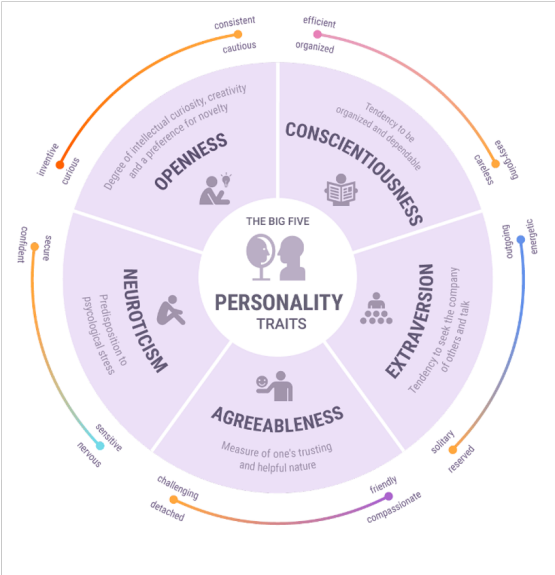


Figure 1. Personality Traits

- Conscientiousness. Standard features of this dimension include high levels of thoughtfulness, good impulse control, and goal-directed behaviors.
- Extraversion. This dimension is characterized by excitability, sociability, talkativeness, assertiveness, and high amounts of emotional expressiveness.
- Agreeableness. This personality dimension includes attributes such as trust, altruism, kindness, and affection.
- Neuroticism. This trait is characterized by sadness, moodiness, and emotional instability.

Our idea is to determine clusters of people among candidates who answered to the Big 5 Personality Test and to create an application that will allow people with similar personality to know each other.

2. Analysis

The entire analysis of the dataset has been realized using Python, and in particular using Pandas library for data preprocessing and scikit-learn's library for machine learning algorithms.

2.1. Dataset

The dataset we used comes from Kaggle (https://www.kaggle.com/tunguz/big-five-personality-test) and was collected through an interactive on-line personality test.
The personality test was constructed with the "Big-Five Factor Markers" from the IPIP (https://ipip.ori.org/newBigFive5broadKey.htm).
The items are rated on a five point scale, labeled 1=Disagree, 3=Neutral, 5=Agree.
In the dataset, 10 questions are available for each personality trait:

EXT1 I am the life of the party.
EXT2 I don't talk a lot.
EXT3 I feel comfortable around people.
EXT4 I keep in the background.
EXT5 I start conversations.
EXT6 I have little to say.
EXT7 I talk to a lot of different people at parties.
EXT8 I don't like to draw attention to myself.
EXT9 I don't mind being the center of attention.
EXT10 I am quiet around strangers.
EST1 I get stressed out easily.
EST2 I am relaxed most of the time.
EST3 I worry about things.
EST4 I seldom feel blue.
EST5 I am easily disturbed.
EST6 I get upset easily.
EST7 I change my mood a lot.
EST8 I have frequent mood swings.
EST9 I get irritated easily.
EST10 I often feel blue.
AGR1 I feel little concern for others.
AGR2 I am interested in people.
AGR3 I insult people.
AGR4 I sympathize with others' feelings.
AGR5 I am not interested in other people's problems.
AGR6 I have a soft heart.
AGR7 I am not really interested in others.
AGR8 I take time out for others.
AGR9 I feel others' emotions.
AGR10 I make people feel at ease.
CSN1 I am always prepared.

CSN2 I leave my belongings around.
CSN3 I pay attention to details.
CSN4 I make a mess of things.
CSN5 I get chores done right away.
CSN6 I often forget to put things back in their proper place.
CSN7 I like order.
CSN8 I shirk my duties.
CSN9 I follow a schedule.
CSN10 I am exacting in my work.
OPN1 I have a rich vocabulary.
OPN2 I have difficulty understanding abstract ideas.
OPN3 I have a vivid imagination.
OPN4 I am not interested in abstract ideas.
OPN5 I have excellent ideas.
OPN6 I do not have a good imagination.
OPN7 I am quick to understand things.
OPN8 I use difficult words.
OPN9 I spend time reflecting on things.
OPN10 I am full of ideas.

The time spent on each question is also recorded in milliseconds. These are the variables ending in _E. This was calculated by taking the time when the button for the question was clicked minus the time of the most recent other button click.
Overall, the dataset we used was composed by 1015341 instances and 110 columns.

2.2. Data Preprocessing

Starting from the dataset, we selected all the answers to the test and the time spent on each question, reaching 100 attributes.

We started our data preprocessing with the data cleaning phase, removing null values and zero values, since zero is not a possible answer to the test or a number of milliseconds spent to answer a question equal to 0 means a probable error regarding the measuration of the time.
After removing zero values and null values the amount of remaining instances were 868933.
This stage also included a remapping of questions negatively correlated with the personality, considering the values in a reverse order.

Then, we started the data reduction phase with a numerosity reduction step first, selecting 50000 instances for computational and storage reasons.
After that, we applied dimensionality reduction using Variance Threshold, selecting for each possible personality the 5 questions with the highest variance.

The following questions were selected:

['EXT1', 'EXT2', 'EXT7', 'EXT9', 'EXT10']
['EST1', 'EST6', 'EST8', 'EST9', 'EST10']
['AGR1', 'AGR3', 'AGR5', 'AGR6', 'AGR9']
['CSN2', 'CSN4', 'CSN5', 'CSN6', 'CSN9']
['OPN1', 'OPN2', 'OPN4', 'OPN6', 'OPN8']

We continued our analysis using only the attributes selected and the time associated with these questions.
We then applied a feature generation phase, considering the average values for each personality and the total amount of time spent to answer all questions, expressed in minutes.
In this way, we can use the time spent to answer the test as a feature to divide users into clusters according to their personality.
We also applied outlier detection removing instances with a lot of time spent to answer the test, which were probably corresponding to error in the measuration of time.

2.3. PCA

Considering the presence of 6 different features, we decided to apply PCA for the visualization of clusterings in the comparison of our algorithms.
Selecting the first three principal components we can reach a cumulative explained variance of 0.76.
Plotting our data on the first three principal components we arrive to the following plots.

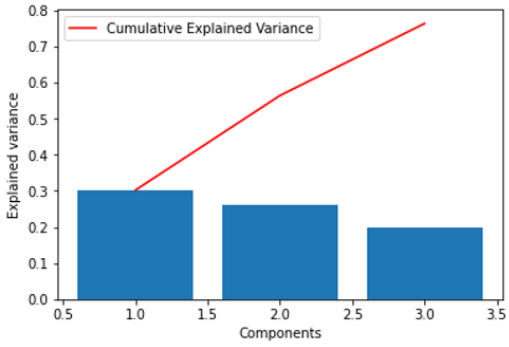


Figure 2. Cumulative Explained Variance

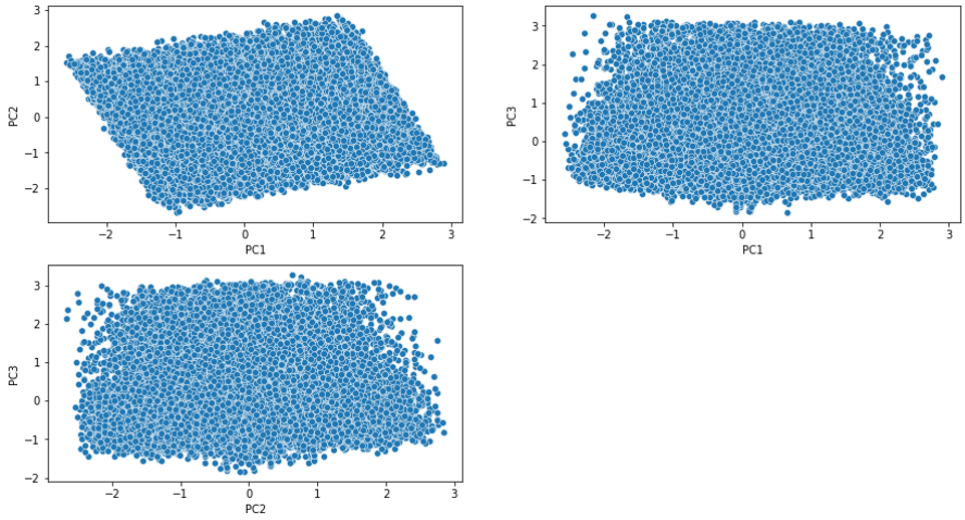


Figure 3. Principal Components Plot

We also tried to apply a varimax rotation, trying to push the values on the loading matrix towards extreme values. Using this approach we encountered a better loading matrix but a lower value of cumulative explained variance (0.57). The following plot shows the data on the principal components after varimax rotation.

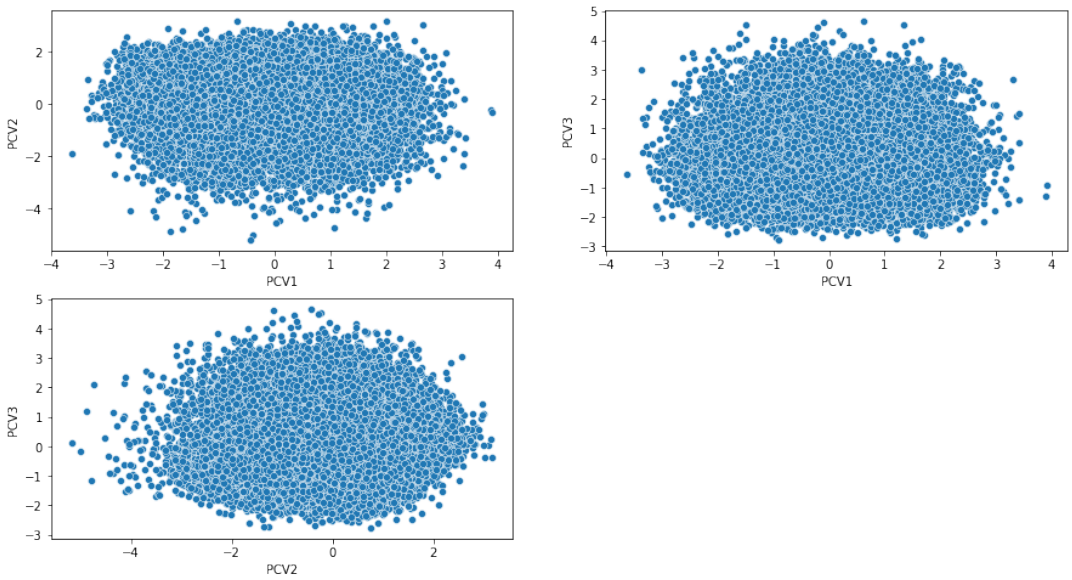


Figure 4. Principal Components Varimax Plot

Due to the loss of a huge amount of cumulative explained variance, we decided to use the original principal components found by PCA.

3. Clustering Algorithms

3.1. Hopkins Statistic

We assessed the clustering tendency on our data using the Hopkins Statistic, obtaining:

H = 0.7394046576506141

With a Hopkins value close to 0.75, we can assess the clustering tendency with a confidence of 90%.

3.2. Partitioning Algorithms

3.2.1. K-Means

3.2.1.1. Choosing the best value of K

Starting from the K-Means algorithm, we used the Elbow Method to determine the optimal value of K. The Elbow method suggested the choice of K = 5, value in which we have the most relevant turning point. After this result, we decided to compare the values of K close to K = 5, to be sure that this could be the best solution.

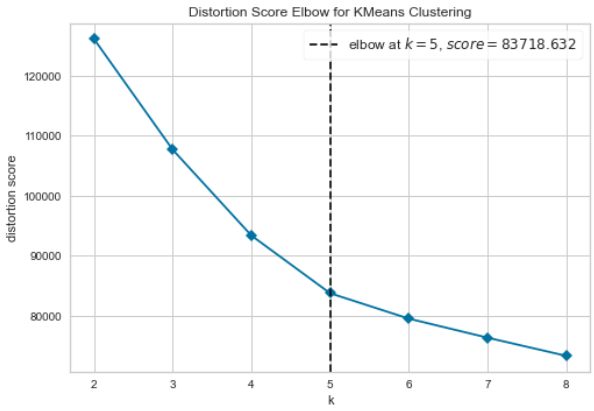


Figure 5. Elbow Method for K-Means

We first used a M-fold cross validation method, using 10 as the number of folds, that confirmed us that K = 5 is the best choice for K-Means.

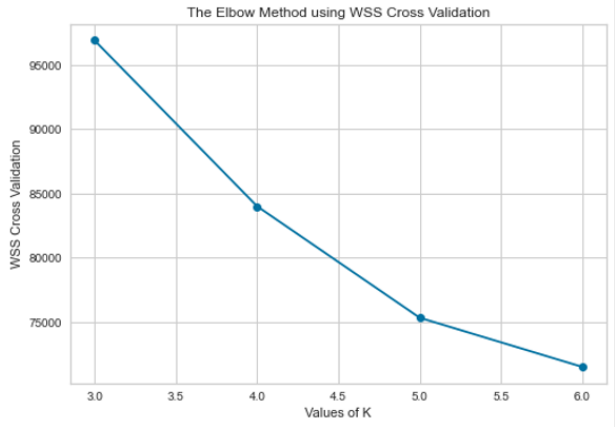


Figure 6. M-Fold Cross Validation for K-Means

We also compared the average silhouette score, the clusters' average silhouette values, the standard deviation of the clusters' silhouette values and the number of negative single silhouette values varying K between {3, 6}.

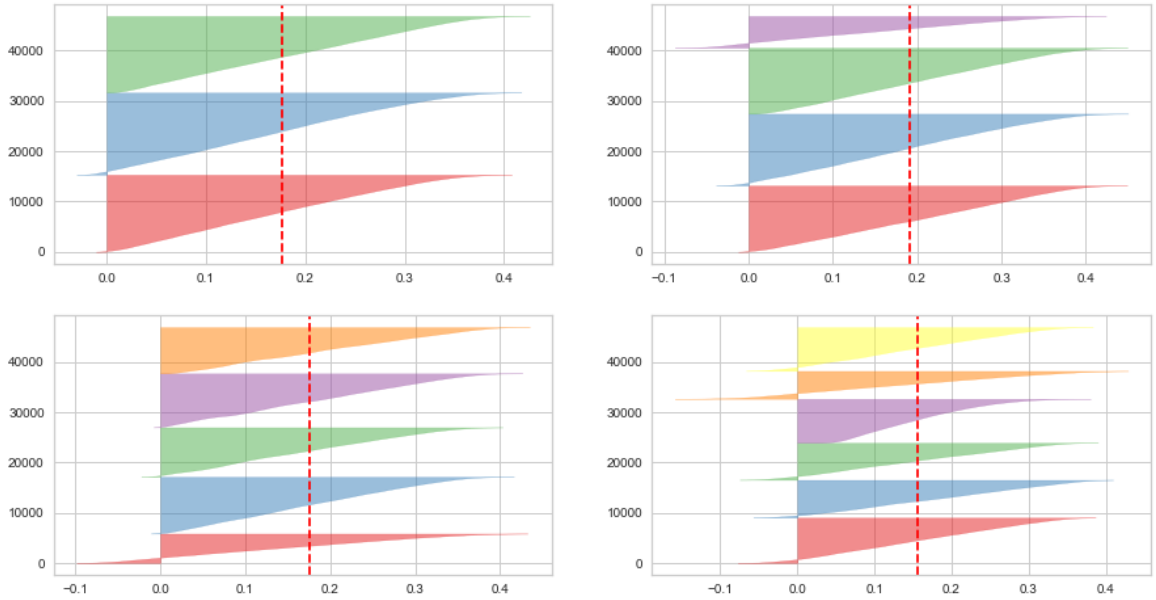


Figure 7. Silhouette for K-Means

	Silhouette Score	**Silhouette mean for each cluster**	**Std of clusters' silhouette**	**Number of negative single silhouette values**
K = 3	0,177	[0,17, 0,17, 0,19]	0,012	961
K = 4	0,19	[0,20, 0,20, 0,18, 0,15]	0,023	1689
K = 5	0,176	[0,20, 0,18, 0,14, 0,17, 0,18]	0,019	1524
K = 6	0,156	[0,13, 0,15, 0,18, 0,16, 0,16, 0,15]	0,012	3821

We can reach the highest silhouette score for K = 4, and then the second highest silhouette scores are reached by K = 3 and K = 5.
Comparing K = 4 and K = 5, we can though see a lower number of single negative silhouette values for K = 5 and a lower std of the clusters' silhouette values.

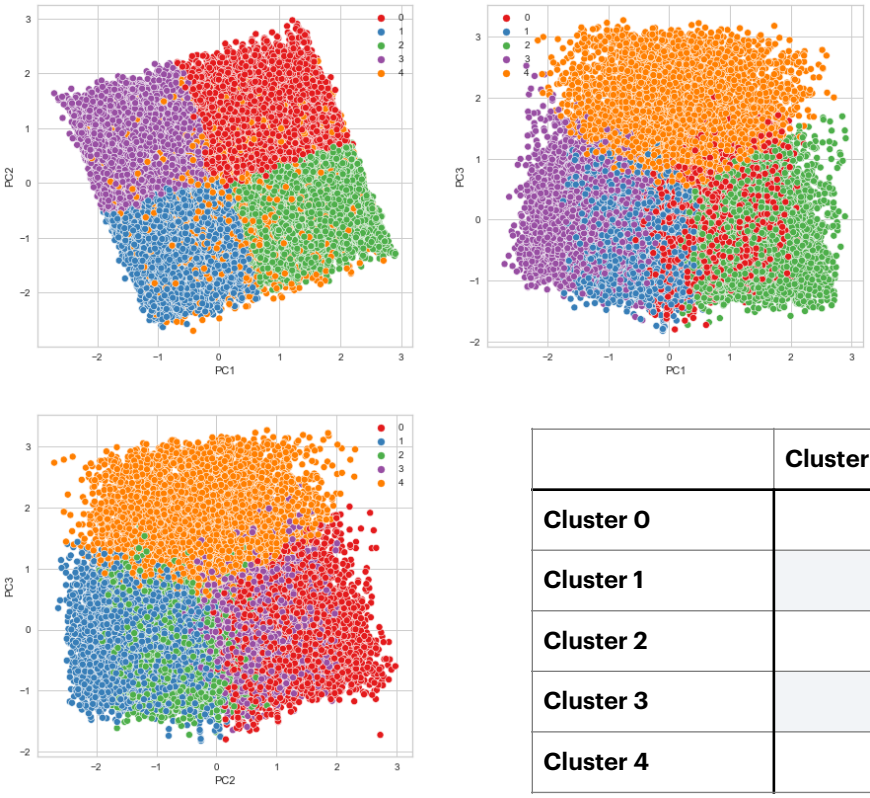
We also compared the consistency of the clusters using different values of K in the previous range.

	Std of the number of points in the biggest cluster
K = 3	37,07
K = 4	46,33
K = 5	45,51
K = 6	27,94

We can see that the best stability is reached by K = 6 and that K = 4 and K = 5 assume the worst results.
Cumulating the results of all these considerations we decided that in any case, K = 5 is the best choice for the number of clusters.

3.2.1.2. K-Means with k = 5

Plotting our clusters, obtained with K = 5 on the first three principal components, we can see a good separation of our clusters:



	Cluster's size
Cluster 0	9126
Cluster 1	11374
Cluster 2	10142
Cluster 3	10396
Cluster 4	5837

Figure 8. Principal Components for K-Means

Some clusters tend to be confused using two principal components, but exploiting the third principal component we can see a clear distinction among them.
Considering our parallel coordinates we can see a clear distincgion of our clusters on some of our features, in particular on TimeSpent, Extroversion and conscientiousness.

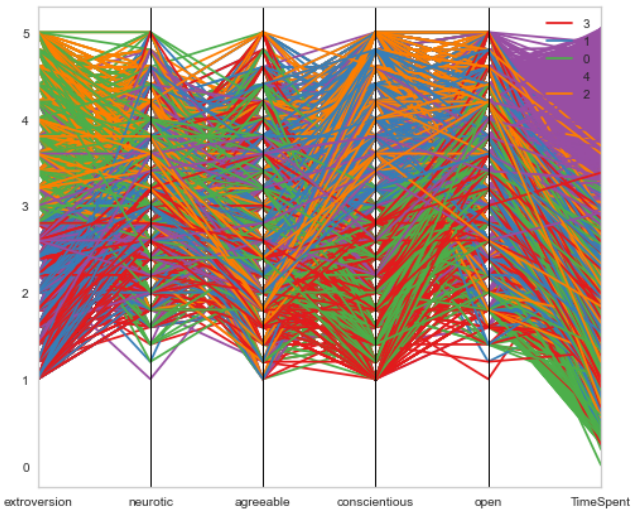


Figure 9. Parallel Coordinates for K-Means

3.2.2. PAM

3.2.2.1. Choosing the best value of K

Using the Elbow method we can see that K = 5 and K = 7 are the most relevant turning points.

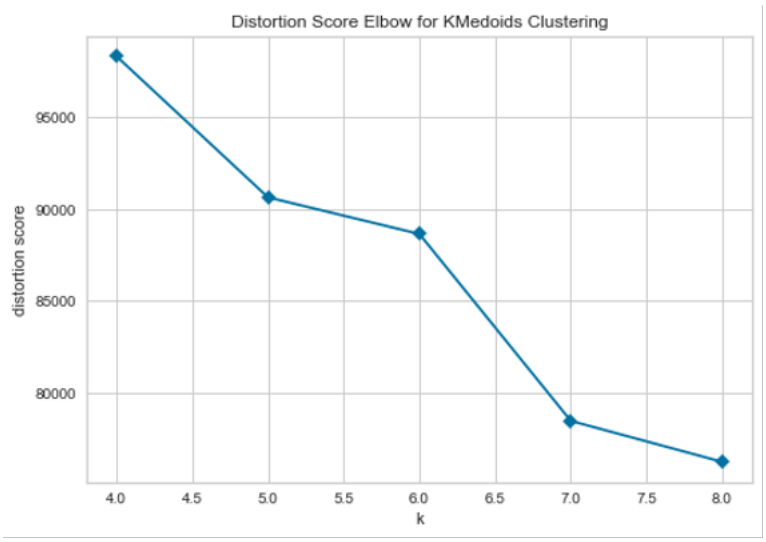


Figure 10. Elbow Method for PAM

Comparing the silhouette values in a close range, we can see that K = 4 reach the highest average global silhouette, followed by K = 5 and K = 7.
K = 5 presents, anyway, the best std of clusters' silhouette mean and the lowest number of negative single silhouette values

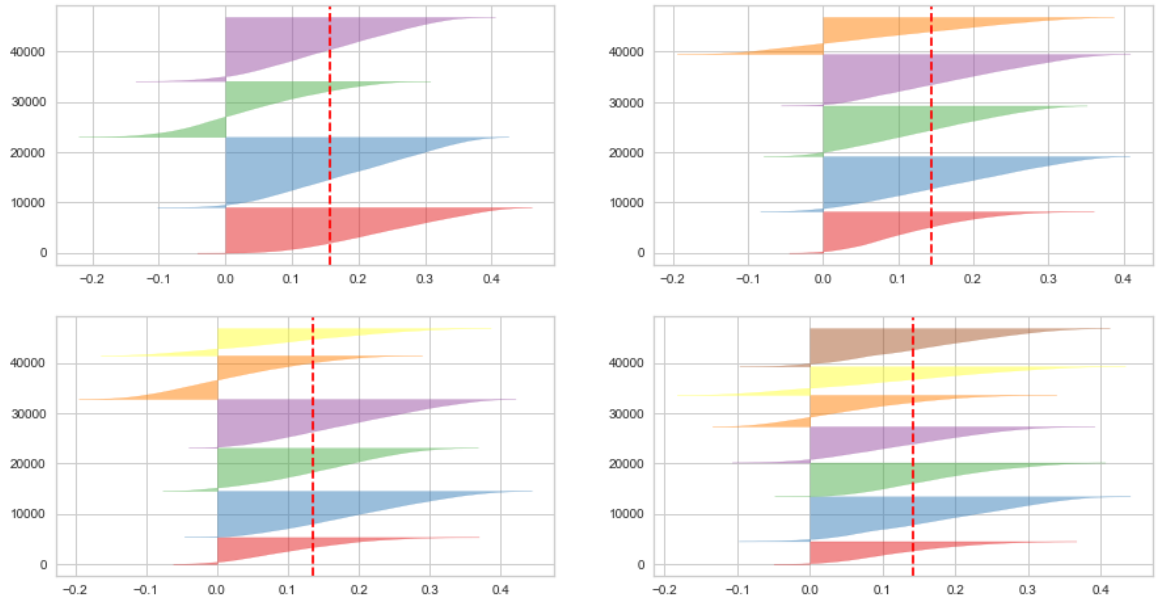
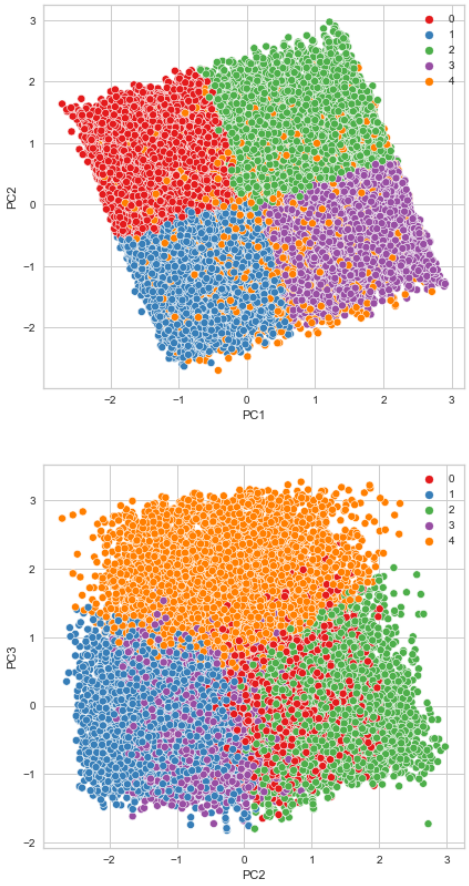


Figure 11. Sihouette for PAM

	Silhouette Score	**Silhouette mean for each cluster**	**Std of clusters' silhouette**	**Number of negative single silhouette values**
K = 4	0,158	[0,25, 0,19, 0,004, 0,16]	0,08	5575
K = 5	0,145	[0,12, 0,17, 0,14, 0,18, 0,09]	0,03	4194
K = 6	0,13	[0,11, 0,20, 0,15, 0,19, 0,02, 0,09]	0,06	6518
K = 7	0,14	[0,13, 0,19, 0,18, 0,14, 0,07, 0,12, 0,16]	0,04	5201

3.2.2.2. PAM with k = 5

Plotting our clusters on the first three principal components and using the Parallel Coordinates we can see similar results than the ones achieved with K-Means.



	Cluster's size
Cluster 0	10364
Cluster 1	11375
Cluster 2	10193
Cluster 3	9109
Cluster 4	5834

Figure 12. Principal Components for PAM

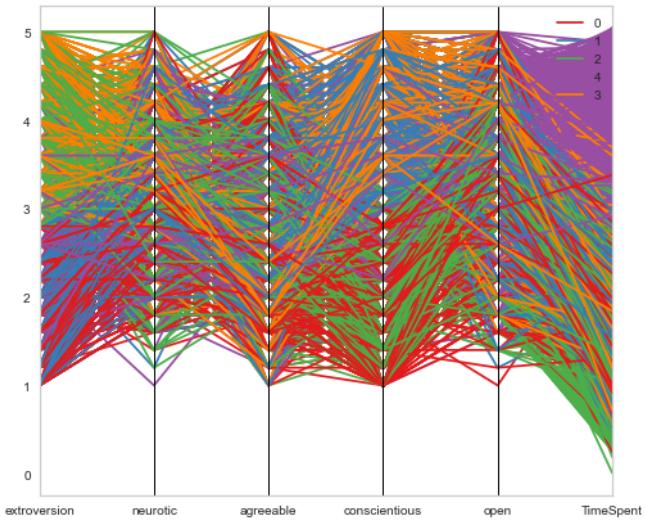


Figure 13. Parallel Coordinates for PAM

3.2.3. Comparing K-Means with PAM

Comparing our Partitioning Algorithms, by consulting the Pair confusion matrix, we can see that K-Means and PAM clustered in the same way the 99.8% of data, proving that the two algorithms found the same clustering structures.

	Time Needed	Silhouette Score	DBI	CHS
K-Means	0,35 seconds	0,176	1,48	10543,06
PAM	135,15 seconds	0,145	1,48	10543,14

Comparing the two algorithms using the Silhouette score, the time needed for determining clusters, the Calinski-Harabasz Score (CHS) and the Davies–Bouldin index (DBI) we can determine that K-Means is the best clusterer in the category of Partitioning Clustering Algorithms.

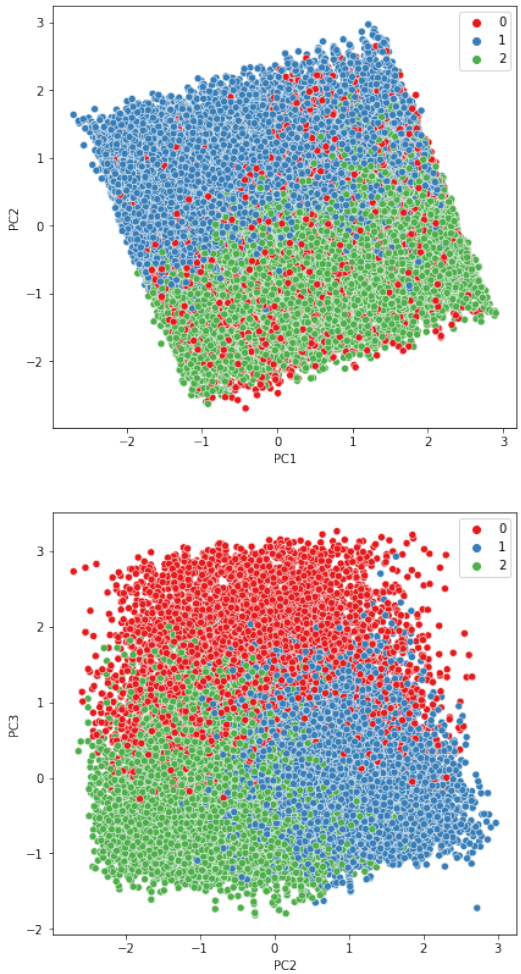
3.3. Hierarchical Algorithms

3.3.1. AGNES

Considering AGNES, we compared the linkage metrics that showed us the best results.

3.3.1.1. Complete Linkage Metric

We explored the dendrogram tree considering different possible cuts of the tree.
Comparing the silhouette values, the DBI, the CHS, the clusters on the principal components and the parallel coordinates we can determine that the cut for which we have 3 clusters is the best among all possible solutions.
For K = 3 we have the following results:



	Cluster's size
Cluster 0	5761
Cluster 1	20951
Cluster 2	20163

Figure 14. Principal Components for AGNES (Complete Linkage)

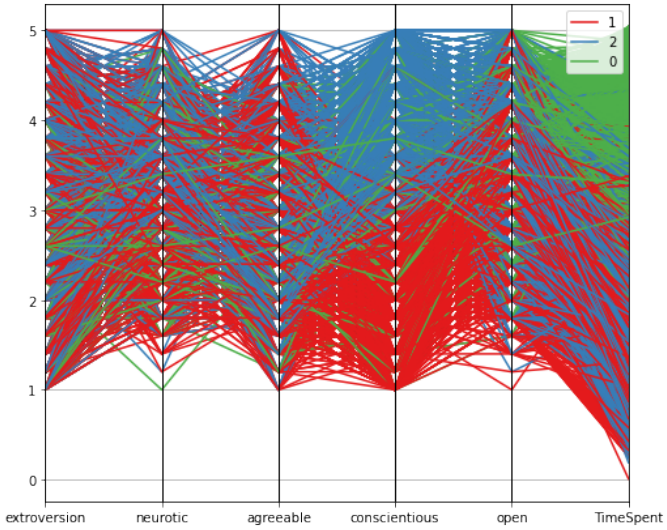


Figure 15. Parallel Coordinates for AGNES (Complete Linkage)

	Silhouette Score	Silhouette mean for each cluster	Std of clusters' silhouette	Number of negative single silhouett	CHS	DBI	Time
AGNES (Complete Linkage)	0,138	[0,09, 0,13, 0,16]	0,03	7791	7060	1.95	122 seconds

3.3.1.2. Ward Linkage Metric

Considering the ward linkage metric and cutting the dendrogram at different heights we obtain again that the height for which we find three different clusters is the best solution. Using this cut we obtain the following results.

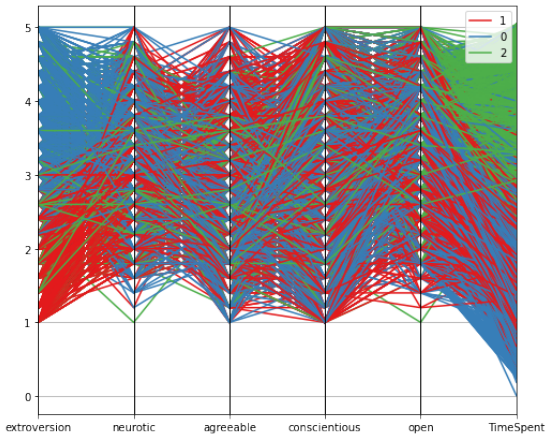
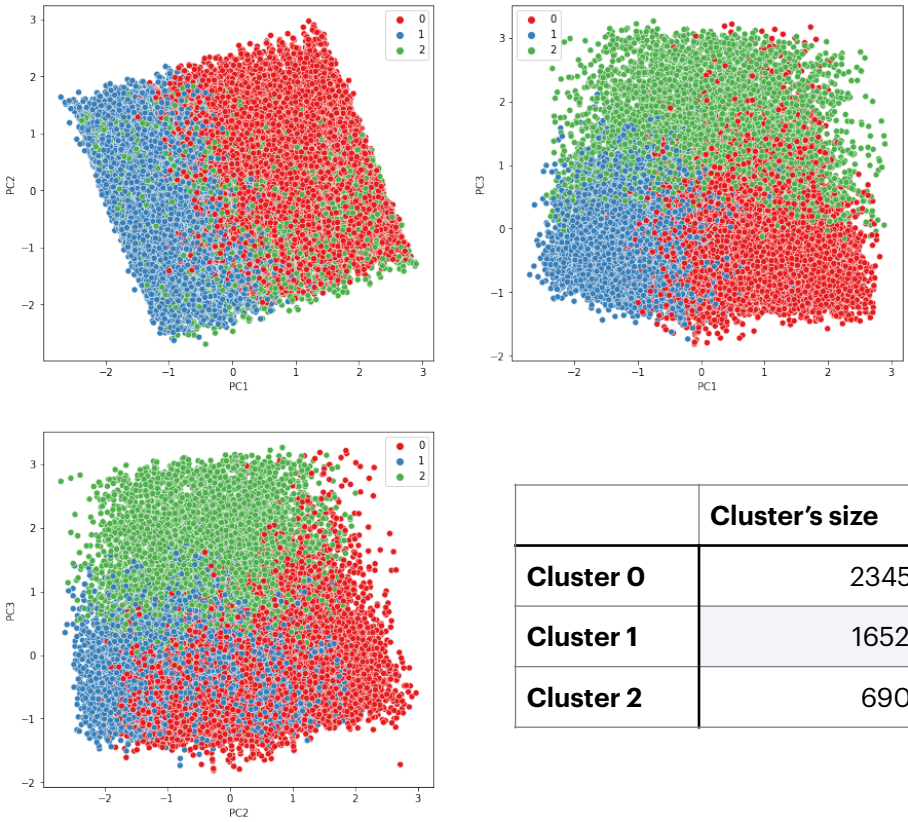


Figure 16. Parallel Coordinates for AGNES (Ward Linkage)



	Cluster's size
Cluster 0	23452
Cluster 1	16520
Cluster 2	6903

Figure 17. Principal Components for AGNES (Ward Linkage)

	Silhouette Score	Silhouette mean for each cluster	Std of clusters' silhouette	Number of negative single silhouette values	CHS	DBI	Time
AGNES (Ward Linkage)	0,145	[0,14, 0,18, 0,09]	0,03	7987	7748	1.88	148 seconds

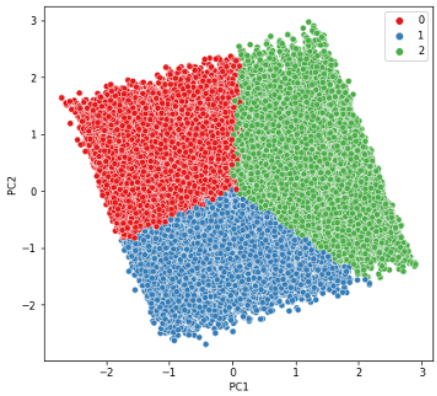
Comparing the two different linkage metrics, the Ward Linkage metric seems to produce better results.

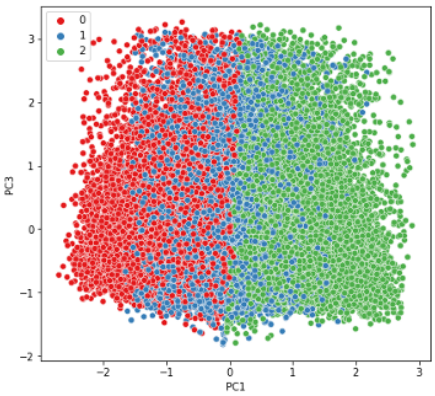
3.3.2. BIRCH

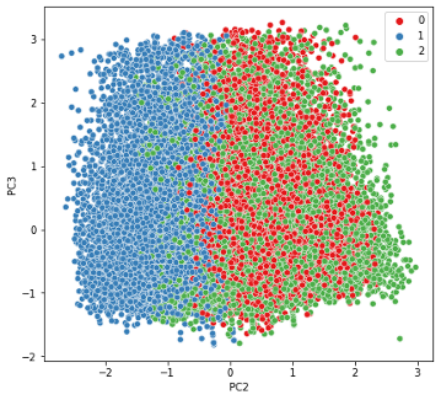
Considering the BIRCH algorithm we decided to explore a range of values for the radius threshold and the branching factor for which we saw the largest difference in terms of results.

Radius Threshold Range = [1.5, 1.6, 1.7, 1.8]
Branching Factor Range = [2, 3, 4]

The bests results seemed to be reached by L = 4 and R = 1.8, for which we obtained 3 clusters with the following results.







	Cluster's size
Cluster 0	15347
Cluster 1	15300
Cluster 2	16228

Figure 18. Principal Components for BIRCH

	Silhouette Score	Silhouette mean for each cluster	Std of clusters' silhouette	Number of negative single silhouette values	CHS	DBI	Time
BIRCH (R = 1.8, L = 4)	0,177	[0,20, 0,17, 0,17]	0,01	884	5087,64	1,81	0,88 seconds

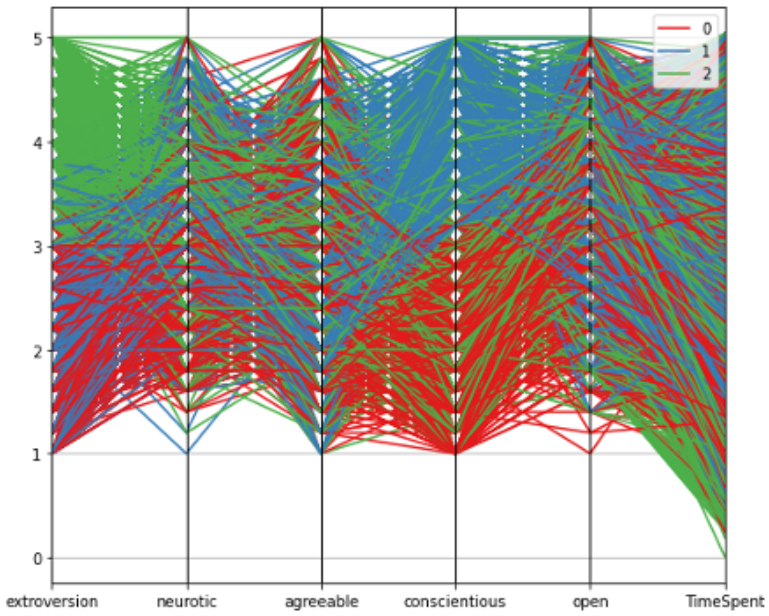


Figure 19. Parallel Coordinates for BIRCH

3.4. Density Based Algorithms

3.4.1. DBSCAN

For DBSCAN, we've used the following heuristic for determining the best combination of parameters.

MinPts = 2 * number of dimensions = 12 (Sander et al., 1998)
Eps = 0.6 (exploiting the k-dist function)

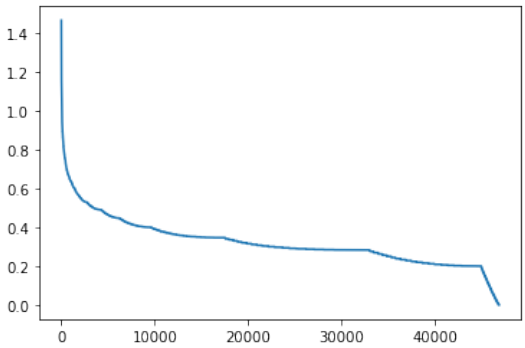
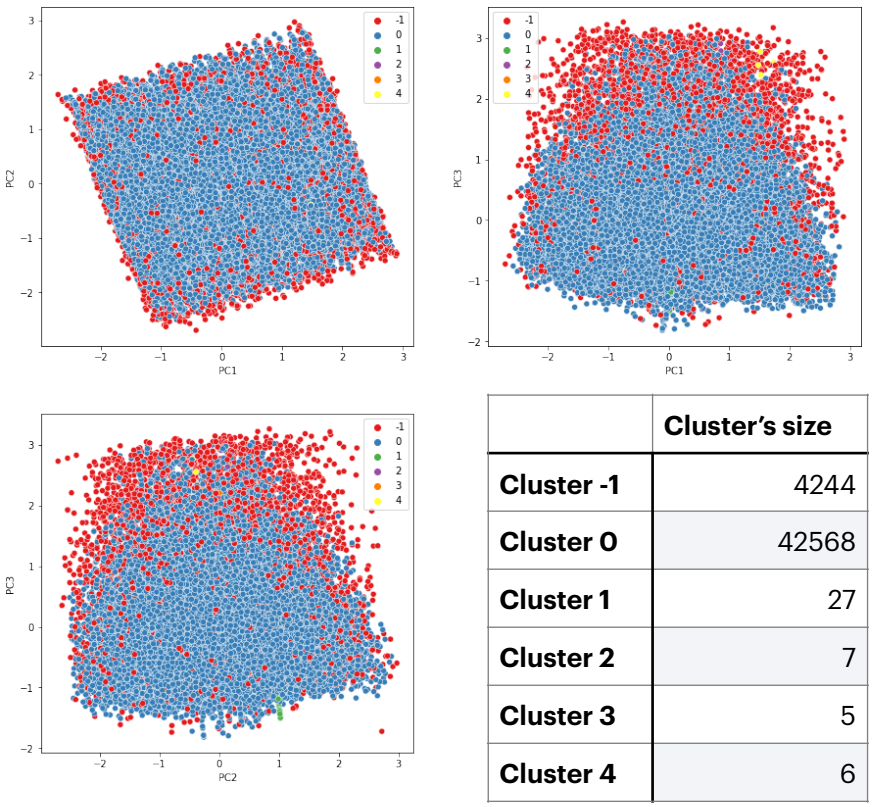


Figure 20. K-dist plot for DBSCAN

However, this combination of values didn't produce a good clustering structure:



	Cluster's size
Cluster -1	4244
Cluster 0	42568
Cluster 1	27
Cluster 2	7
Cluster 3	5
Cluster 4	6

Figure 21. Principal Components for DBSCAN

The reason for this bad results was probably due to the fact that our distribution data is not suited for density based algorithms.
We also tried to evaluate different values for eps and MinPts but without reaching good results.

3.5. Spectral Clustering

We decided to apply another strategy, considering all 100 attributes (all answers and time for each) and apply a spectral clustering using the NG-Jordan-Weiss algorithm, suited for high dimensional data.
Considering these other features, we can also assess the clustering tendency.

H = 0.728

Due to the huge time needed by spectral clustering to determine clusters, we compared the results considering just a lower number of instances but reaching weak results, seeing that this algorithm group almost all points in the same cluster.

	Cluster's size
Cluster 0	1155
Cluster 1	1
Cluster 2	1
Cluster 3	1
Cluster 4	1
Cluster 5	1
Cluster 6	1
Custer 7	1

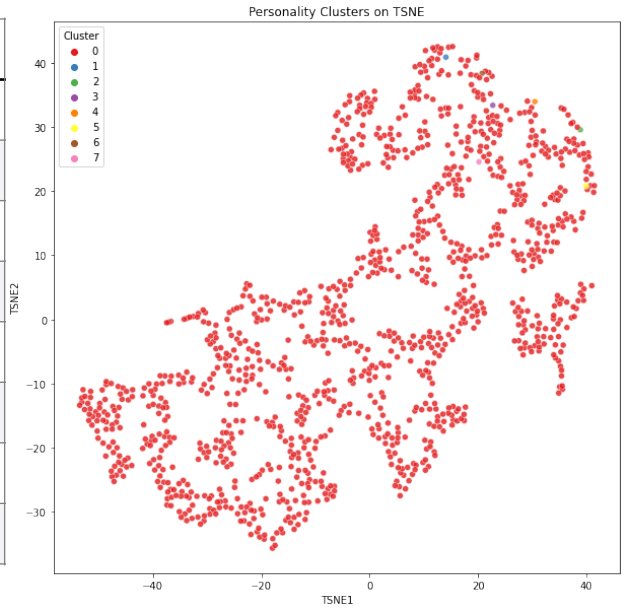


Figure 22. TSNE for Spectral Clustering

We decided to reject this approach considering the computational time needed by spectral clustering to perform clustering and the weak results.

3.6. Comparison of all clustering algorithms

Comparing clustering algorithms that provided the best results, according to all our selected metrics, we can clearly see that the best two clustering algorithms are K-Means and BIRCH.
The results achieved by KMeans seemed anyway preferable.
In fact, even if numeric results are comparable, we can reach these results with a higher number of clusters using KMeans, and achieving also a better interpretability.

	Silhouette Score	Silhouette mean for each cluster	Std of clusters' silhouette	Number of negative single silhouette values	CHS	DBI	Time
K-Means	0,176	[0,20, 0,18, 0,14, 0,17, 0,18]	0,019	1524	10543,06	1,48	0,35 seconds
PAM	0,145	[0,12, 0,17, 0,14, 0,18, 0,09]	0,03	4194	10543,14	1,48	135,15 seconds
AGNES (Ward Linkage)	0,145	[0,14, 0,18, 0,09]	0,03	7987	7748	1.88	148 seconds
BIRCH	0,177	[0,20, 0,17, 0,17]	0,01	884	5087,64	1,81	0,88 seconds

3.7. Data Streaming Clustering

We also decided to compare data streaming clustering algorithms using MOA, the most popular open source framework for data stream mining, related to the WEKA project.
It includes a collection of machine learning algorithms for clustering and classifications and techniques for concept drift detection.

3.7.1. Concept Drift Evaluation

Using the Basic Concept Drift Performance Evaluator, we can see that 17 statistical changes are detected considering the stream of our 50.000 instances.

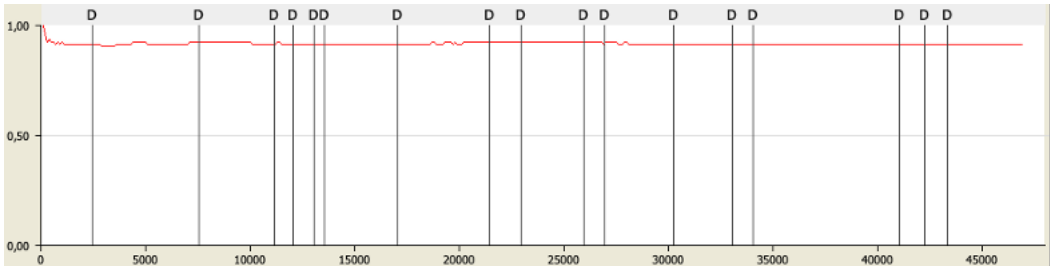


Figure 23. Output from the MOA's Concept Drift Evaluator

3.7.2. MOA Algorithms

We decided to compare three different streaming clustering algorithms in our analysis: withKmeans, StreamKM and BICO.

- StreamKM++ computes a small weighted sample of the data stream and it uses the k-means++ algorithm as a randomized seeding technique to choose the first values for the clusters. To compute the small sample, it employs coreset constructions using a coreset tree for speed up.

- BICO (an acronym for "BIRCH meets coresets for k-means clustering") combines the data structure of BIRCH with the theoretical concept of coresets for clustering. BIRCH decides heuristically how to group the points into subclusters. The goal of BICO is to find a reduced set that is not only small, but also offers guarantees of approximating the original point set.

- WithKmeans determines the closest kernel and check whether instance fits into closestKernel. If the instance fits, it is put into the kernel,

otherwise it applies a free-memory strategy to insert a new kernel (trying to forget oldest kernels); then, it merges the two closest kernels.

	sizeCoreset	numClusters	length
StreamKM	10000	5	100000

	numClusters	maxClusterFeatures	Projections
BICO	5	100	10

	Horizon	maxNumKernels	kernelRadiFactor	K
WithKmeans	1000	100	2	5

Using the configuration of parameters previously described, we obtain the following average global silhouette comparison with a frequency of 1000 observations.

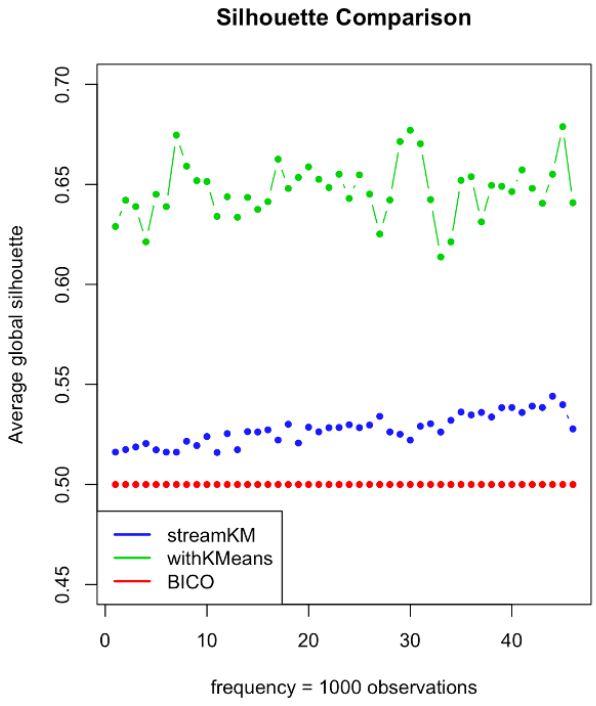


Figure 24. MOA Silhouette Comparison

From this comparison we've decided that withKMeans was the best streaming clustering algorithm.

4. Application

Personality Clustering is an application whose aim is to help people find others with a similar personality.
The application will group people into different clusters and then, users will be recommended to the people in the same cluster that have the closest values of personality traits.
Users will be able to decide if send or not a friend request, and once accepted, users will be able to see other people's email, that will be used to start a conversation.
People can also see stats on their personality and the average personality of people in the same cluster.

4.1. Requirements

4.1.1. Main Actors

We can distinguish two different actors in our application:

- Anonymous Users, which can only register or login
- Standard Users, who are the primary users of the application. They can send friend requests to recommended friends, accept/decline incoming friend requests, see stats on their personality and edit their information.

4.1.2. Functional requirements

The following are the functional requirements required for our application.

- The application must handle a login process in order to allow anonymous users to enter the application.
- The application must handle a registration process, so that anonymous users can register into the application and answer the quiz.
- The application must provide standard users the possibility to browse recommended friends and send them a request.
- The application must provide standard users the possibility to browse recommended friends nearby and send them a request.

- The application must provide standard users the ability to accept/decline incoming friend requests.
- The application must provide standard users an interface to see stats on their personality and the personality of their cluster.
- Standard users must be able to edit their information settings.

4.1.3. Non-functional requirements

The following are the non-functional requirements required for our application.

- The application must be able to apply standard/streaming clustering algorithms.
- Continuity: The application must provide continuity of the service, even when clusters are updated.
- Speed: The application must perform clustering algorithms fast.
- Usability: The application must be user friendly with a simple and intuitive user interface.

4.2. Use Case Diagram

The following UML use case diagram shows the user interaction with the application.

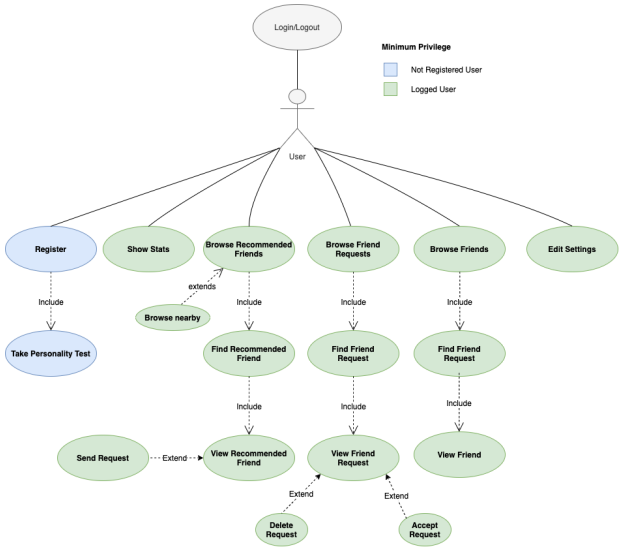


Figure 25. Use Case Diagram

4.3. Analysis Classes

In the following diagram we can see the main entities of the application and the relationships among them.

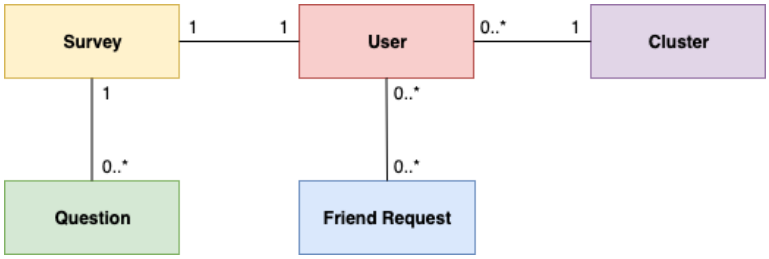


Figure 26. Analysis Classes Diagram

Each user may send/receive one or more friend requests to/from other users. Each user belong to one cluster in a certain moment, while one cluster can involve 0 or more users.
One user has associated a survey and each survey is associated with a user. Each survey is composed by 0 or more questions and each question is related to a survey.

4.4. Data Model

We decided to use two different databases to store our data:

- A document database, using MongoDB DBMS, to store the complete data of every user.
- A graph database, using Neo4J DBMS, to store the relationships among users.

4.4.1. MongoDB

Inside our MongoDB database we only have the User collection, which contains a document for every user. The document contains all the user's personal information and the full survey as an embedded document.
The survey object contains a field for every set of questions regarding a different personality. The question document itself is an embedded document containing the name of the question, the value of the answer and the time spent on the question in seconds. Each set of questions is stored as an array of Question objects.

Figure 27. User Collection Example

For each instance of the dataset, a user was fetched from https://randomuser.me and become a user of our application.

4.4.2. Neo4j

The only node present in our graph database is User.
This node contains all the essential informations about each user: *name, mongoId, country, picture*. The mongoId attribute was used to link each user to its document in the MongoDB database.

To link users in our graph database there are two different relationships:

- SIMILAR_TO: every time a user logins all the users from its cluster are fetched and the application computes the user's distance from each one of them. The first 30 users ordered by descending distance that are not already friends with the user are linked with a SIMILAR_TO relationship. The

relation has a weight property that is equal to the inverse of the squared distance between two users.

- FRIEND_REQUEST: every time a user sends a friend request to another user a directed relationship with UNKNOWN status will be added between the users. If the target user accepts the request the status will change to ACCEPTED, otherwise if it's declined the status will become REFUSED. If the status is not UNKNOWN the link is treated as undirected in the queries.

Because the mongoId property must be unique for every user and always present we added a constraint to our database:

```
neo4j$ CREATE CONSTRAINT mongoId_constraint FOR (n:User) REQUIRE
       (n.mongoId) IS NODE KEY
```

Figure 28. Constraint on Graph DB

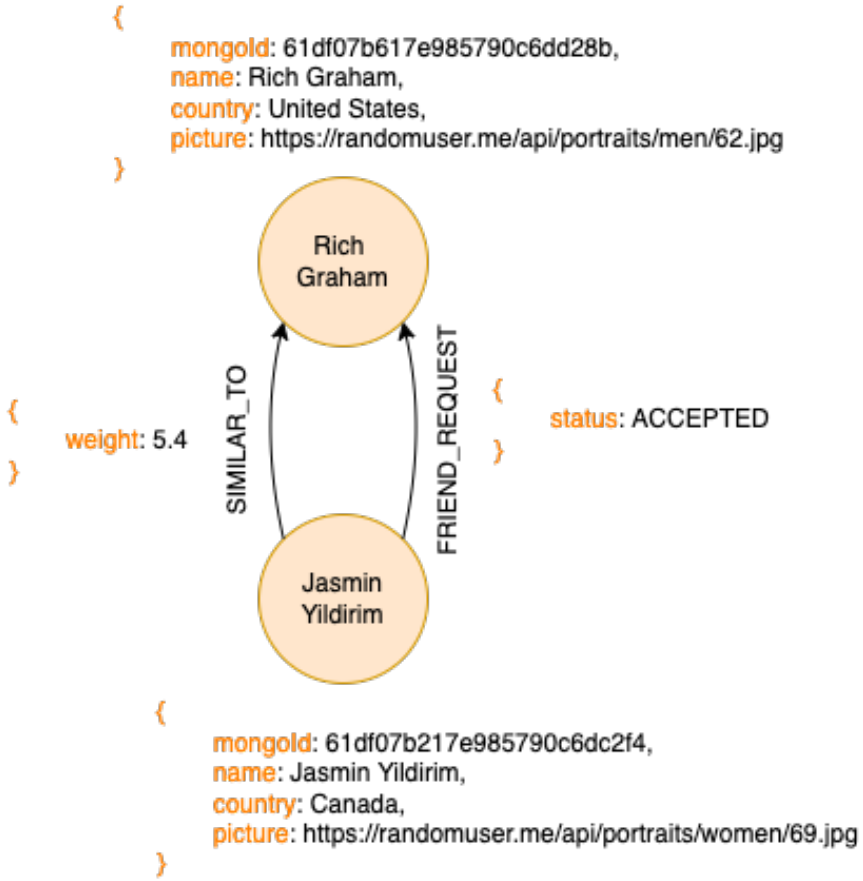


Figure 29. Example of Graph DB instances

4.5. System Architecture

The architectural pattern used for the design of the overall system is the client-server pattern. The client and the server interact with the HTTP protocol sending JSON data to each other that is then converted to Java DTO (Data Transfer Objects) classes.

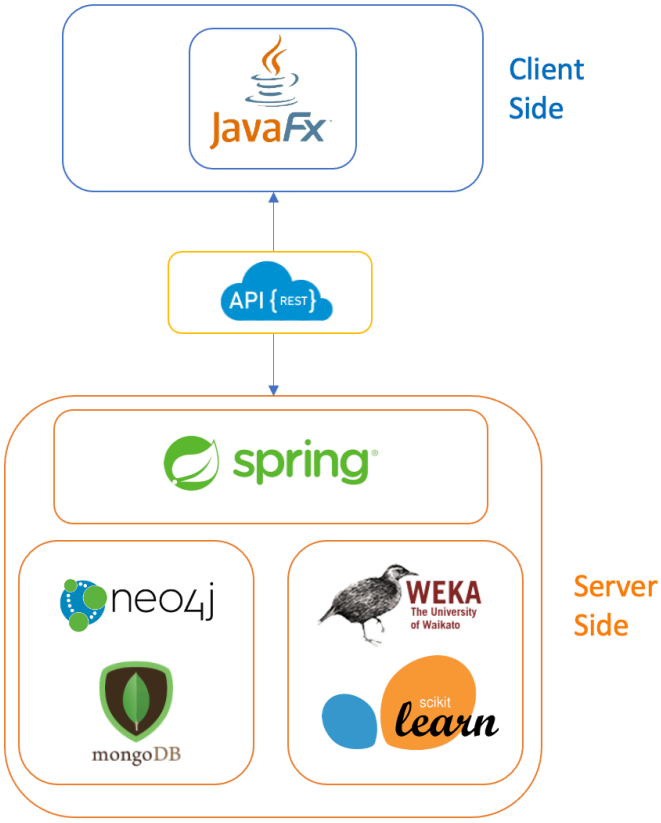


Figure 30. System Architecture

4.5.1 Frameworks

The application code will be written in Java, using JavaFX and SceneBuilder for the GUI.
Concerning the database management systems, Neo4j and MongoDB have been used.
The server implements a REST API with the Spring Boot Java Framework.
The server will be able to select features using the Variance Threshold Filter written in Python.
Regarding python, the Pandas and Scikit-Learn libraries have been used.
Apache Maven has been used as a tool for building and managing the whole application, while for version control, Git has been used.

4.5.2. Server Side

The server implements a REST API with the Spring Boot Java Framework. Every functionality of the server is accessed using Uniform Resource Identifiers, also known as URIs (e.g. http://localhost:8080/users).
The server is divided into different layers, each of them enclosed in a specific package:

- The controller layer is in charge of handling a request from the moment when it is intercepted to the generation of the response and its transmission. This layer calls one or more service layer functions and manages the deserialization of the request and the serialization of the response through the DTO layer;
- The service layer encapsulates the business logic of the server. The service layer of this server also contains the functions needed for the clustering process;
- The DAO (Data Access Object) layer is responsible for encapsulating the details of he persistence layer and provide a CRUD interface for a single entity;
- The model layer contains the entities the represent the data stored in the databases;
- The DTO layer is used to decouple data representation to model objects
- The data layer contains specific data structures needed for passing data from controllers to services and vice versa;

For the MongoDB access, Spring Data MongoDB was used, which provides integration with the MongoDB document database. Key functional areas of Spring Data MongoDB are a POJO (Plain Old Java Object) centric model for interacting with a MongoDB DBCollection and easily writing a Repository style data access layer.

4.5.3. Client Side

On the client side, we can find the Presentation Layer and the remaining part of the Logic Layer.
The Presentation Layer consists of a Graphical User Interface with which the users can interact.
The portion of the Logic Layer in the client, is in charge of processing requests coming from the user, interact with the server and provide the information requested.

All the source code for the application can be found on GitHub at the following links:

- Server GitHub Repository: https://github.com/jacopocecch/data-mining-backend
- Client GitHub Repository: https://github.com/jacopocecch/data-mining-frontend
- Python Analysis Repository: https://github.com/terranovaa/PersonalityClusteringAnalysis
- Executables and Database Dumps have been uploaded on Google Drive for the Github limitations on uploadable file size : https://drive.google.com/drive/folders/1cfe6sfUdhwK00R_F9YhIy2ywKq5gS0U_?usp=sharing

5. User Manual

5.1. Login

The user can login to the application using email and password.
If the user is not registered yet, it can use the Register button to start the registration.

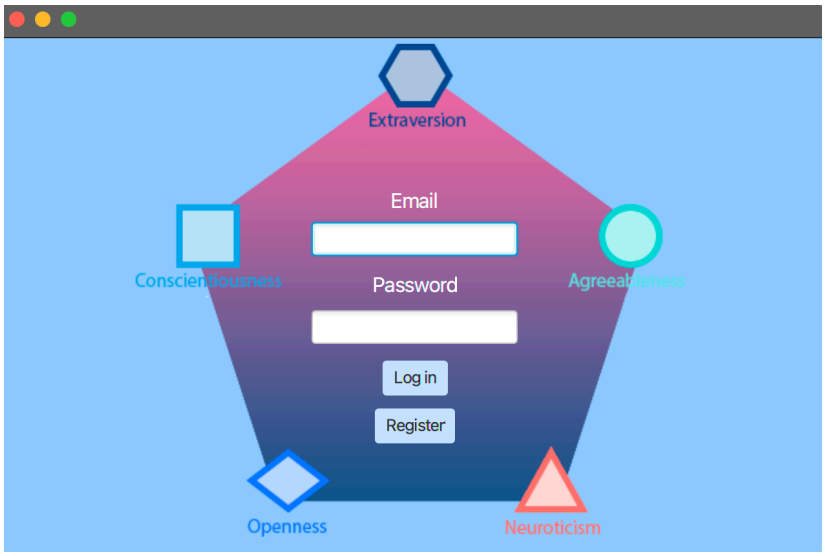


Figure 31. Login Page

5.2. Registration

The user can use the registration form to register into the application providing personal information.

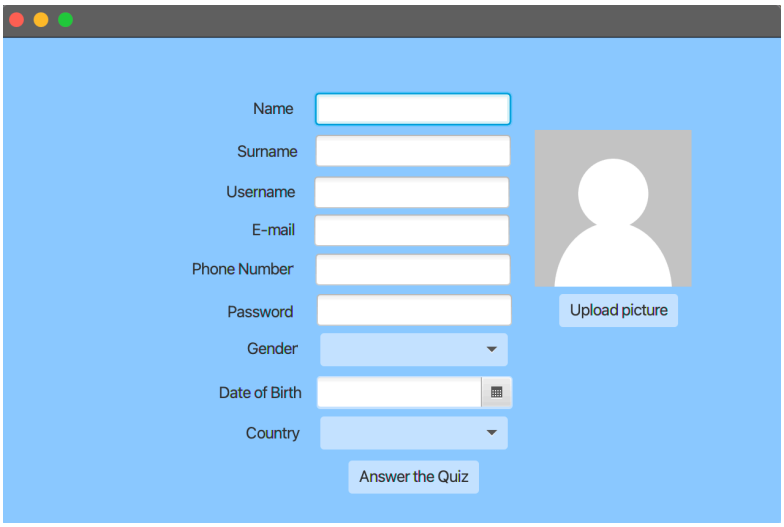


Figure 32. Registration Page

Once provided all the information needed, the user can start to answer the quiz.

Figure 33. Survey's questions Page

Fifty different question will require to be answered using a value between 1 (Disagree) and 5 (Agree). The default value is 3 (Neutral).
Once answered to all the questions, the user will receive information about its personality traits and the cluster in which he will be assigned.

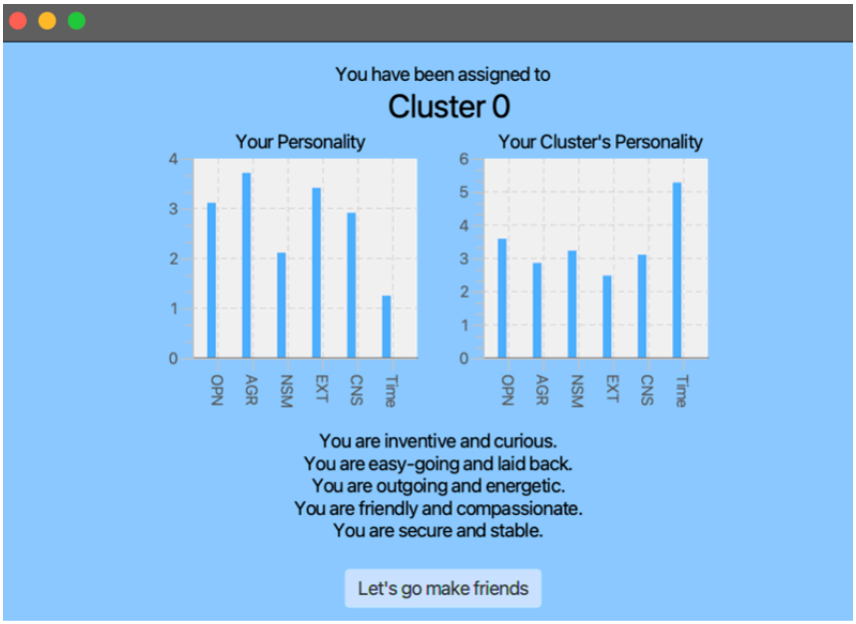


Figure 34. Stats Page

5.3. Home

In the home page the user will be able to browse recommended users and send them a request.

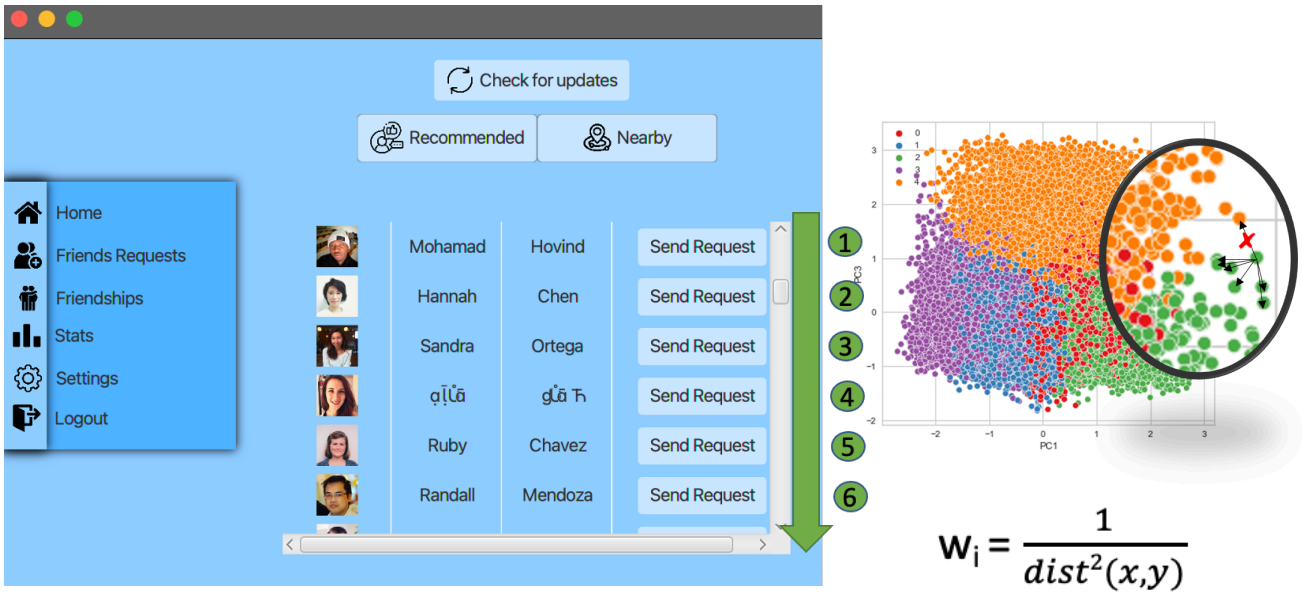


Figure 35. Home Page

The recommended users shown are the thirty nearest neighbors inside the same cluster and they will be ordered according to the euclidian distance on the space of the features, in increasing order.
The user will also be able to browse nearby friends, which are recommended friends in the same country.
Using the 'Check For Updates' button, the user can update the list if the clusters have changed.

5.4. Friends Requests

The user will be able to browse incoming friend requests and accept or decline them.

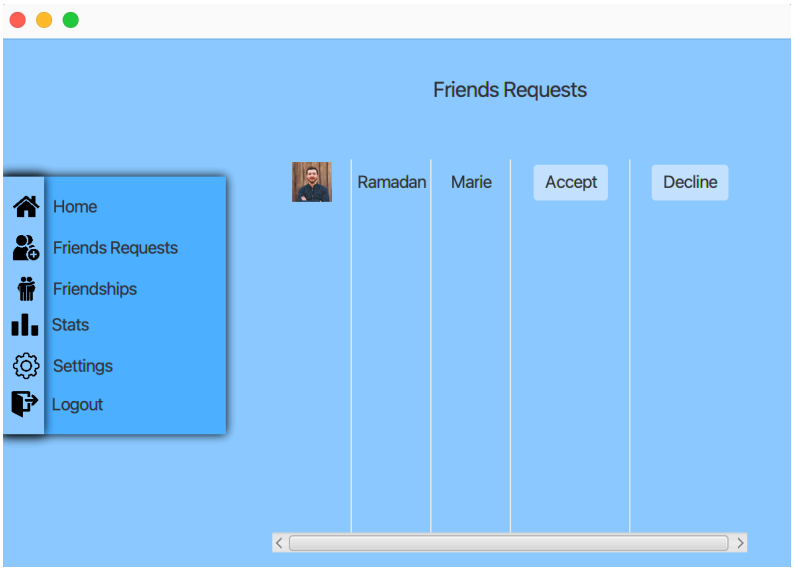


Figure 36. Friends Requests

5.5. Friendships

The user will be able to browse friends and see their information to start a conversation.

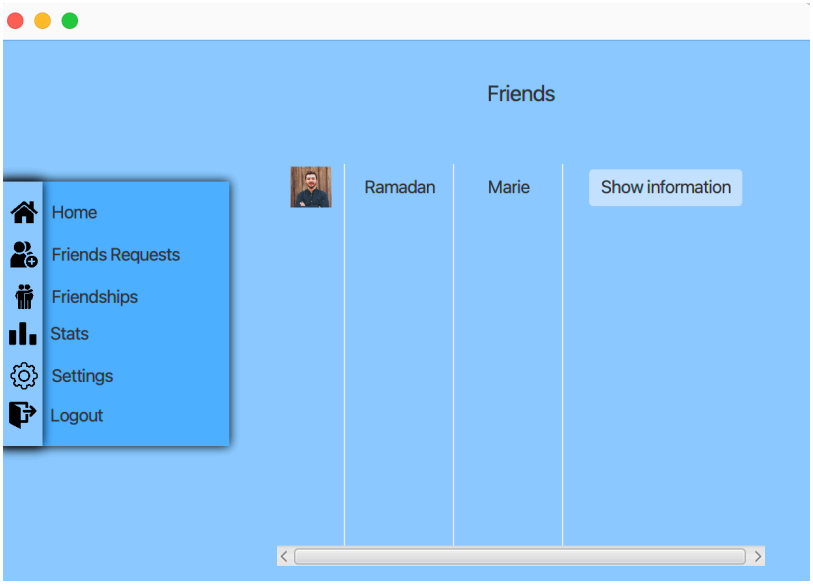


Figure 37. Friendships

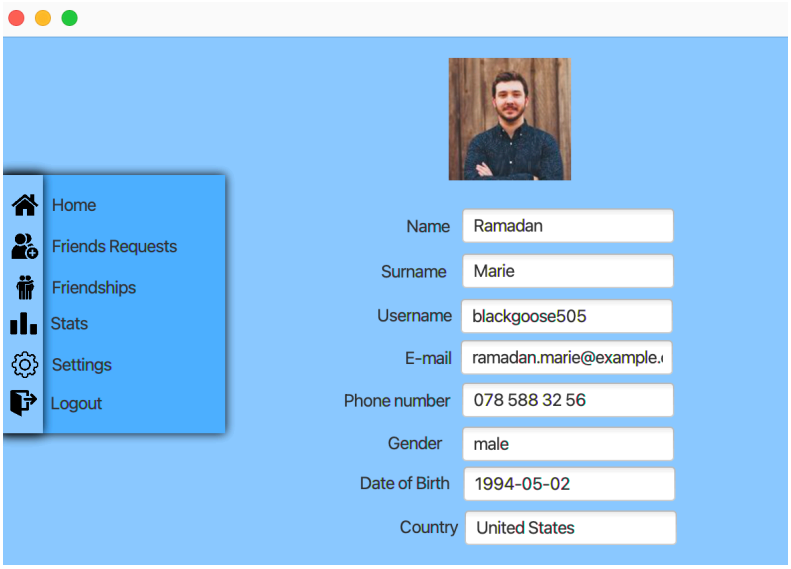


Figure 38. User Information

5.6. Stats

In the stats menu, the user can see statistics about its personality and the average personality of people in the same cluster.

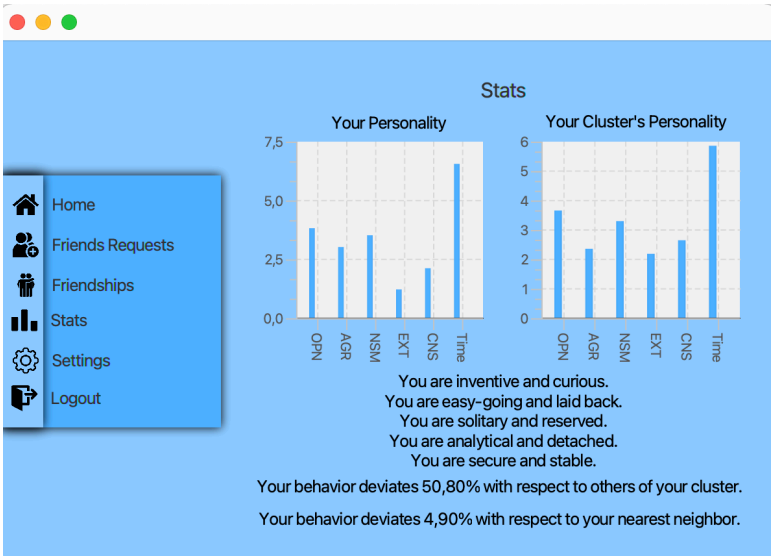


Figure 39. Stats Page

A brief description of the personality in terms of big 5 personality traits will also given to the user, depending on its answers to the survey.
The user can also have information about the deviation of its behavior from the cluster's average behavior, and its deviation from the nearest neighbor in the cluster.

5.7. Settings

Using the settings tab, the user will be able to change personal information.

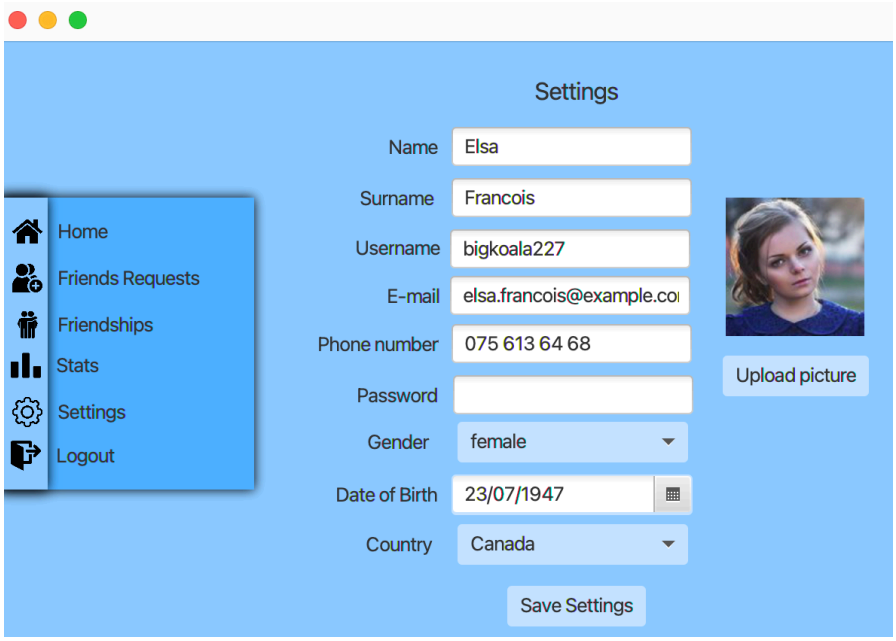


Figure 40. Settings Page

6. Implementation

6.1. Clustering Implementation

Clustering is performed at the initialization of the server.
If some of the users will be grouped in a different cluster than the previous, the value of the cluster ID in the database is updated.
During this updating process, continuity of service is guaranteed, which means that a user will be able to register or login anyway, even if clusters are changing.
The registration of the user in the streaming mode will produce an evolution of the clusters, while in standard mode will produce an evolution of the clusters only if the threshold is reached.
On the server side, the application.properties file is used to set the clustering mode:

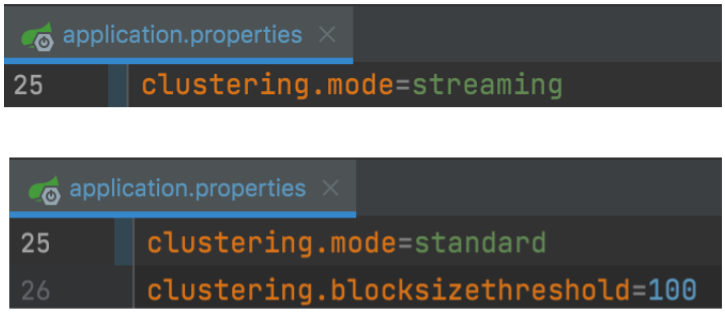


Figure 41. application.properties file

If standard clustering mode is set, a block processing algorithm is used.
Each time we reach the block size threshold, the K-Means Algorithm will be run again.

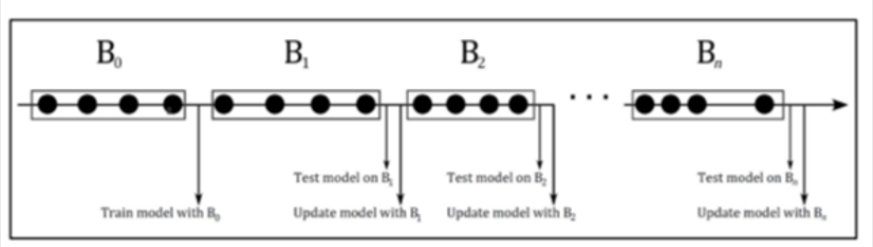


Figure 41. Block Processing

If the streaming mode is set, when the user logins, the system checks if the cluster has changed and if so, updates the databases opportunely.

6.2. Client Side

In this section, the main packages of the client of the application and the contained classes are described.

6.1.1. com.unipi.datamining

This package contains the main class, that starts the application.

Classes:

- PersonalityClustering: this class extends Application, implements the start method and provide all basic functionalities for the application. (MORE INFO)

6.1.2. com.unipi.datamining.gui

This package contains the FXML Document Controllers and the classes for input fields' validation.

Classes:

- FXMLHomeDocumenController: this class implements Initializable and represents the controller for the GUI of the home of the application.
- FXMLRegisterDocumenController: this class implements Initializable and represents the controller for the GUI of the registration form and the survey form of the application.
- FXMLLoginDocumenController: this class implements Initializable and represents the controller for the GUI of the login form of the application.
- LoaderFXML: this class provides the method that allows to load the correct FXML file.
- ValidationForm: this class provides the methods for validating input fields.

6.1.3. com.unipi.datamining.util

This package contains the utilities for the application.

Classes:

- ConfigurationParameters: this class provides the configuration parameters of the application and it is built starting from an XML file.
- UtilGUI: this class provides methods that contains utilities for the GUI.

6.1.4. com.unipi.datamining.entities

This package contains the entities of the application.

Classes:

- User: this class stores all the information of a user, in particular personal information, the cluster and the survey associated.
- Question: this class stores all the information of a question, in particular the question label, the answer and the time.
- Survey: this class stores all the information of a survey, in particular the list of questions and answers.
- FriendRequest: this class stores all the information of a friend request, in particular the user associated and the status.
- SimilarUser: this class stores all the information of the relationship between two similar users, in particular the weight.
- Cluster: this class stores all the information regarding mean values of personality traits for a cluster.

6.1.5. com.unipi.datamining.dtos

This package contains the dtos used to communicate with the application server.

Classes:

- UserDto: this class stores all the information of a user, in particular personal information, the cluster and the survey associated.
- QuestionDto: this class stores all the information of a question, in particular the question label, the answer and the time.
- SurveyDto: this class stores all the information of a survey, in particular the list of questions and answers.
- FriendRequestDto: this class stores all the information of a friend request, in particular the user associated and the status.
- SimilarUserDto: this class stores all the information of the relationship between two similar users, in particular the weight.
- ClusterDto: this class stores all the information regarding mean values of personality traits for a cluster.
- LoginDto: this class stores all the information regarding the login of a user, in particular username and password.

6.1.6. com.unipi.datamining.beans

This package contains the beans used to represent the rows In the JavaFX table view.

Classes:

- UserBean: this class stores the GUI information of a user.

6.1.6. com.unipi.datamining.API

This package provides all the communication with the server.

Classes:

- API: this class contains the methods required for the communication with the server.

6.3. Server Side

In this section, the main packages of the server of the application and the contained classes are described.

6.2.1 com.unipi.data.mining.backend

This package contains the main Application class that starts the Spring Boot Application.

6.2.1 com.unipi.data.mining.backend.configs

This package contains all the configuration classes.

- ClusteringConfigurationProperties: this class contains the configurations for the clustering classes. The values of the variables are taken from the respective fields in the application.properties configuration file.
- Config: this is the configuration class. It contains the configuration for the CORS Mapping and the DTO mapper.

6.2.2 com.unipi.data.mining.backend.controllers

This package implements the controller layer of the application

6.2.2.1 com.unipi.data.mining.backend.controllers.errors

This package is responsible for handling the exceptions and sending them back to the client in an interpretable way

- ControllerExceptionHandler: this class contains the code for generating the error response message based on the type of exception that was thrown by the application.
- ErrorBody: the body of the response with a status code and a message.
- ErrorResponse: the error response class.

6.2.2.2 com.unipi.data.mining.backend.controllers.services

This package handles all the requests and responses.

- ServiceController: this abstract class contains al the instances for the DTO and service layer that need to be used by the controllers.
- UserController: this controller extends ServiceController and implements all the functions needed by the client by defining a different URI for each one. It implements serialization/deserialization and call the corresponding service layer functions.
- ClusteringController: this class starts the clustering process during the application startup depending on the configuration parameters.

6.2.3 com.unipi.data.mining.backend.daos

This package implements the Data Access Object for the Neo4j database. There is no DAO class for MongoDB because we used Spring Data MongoDB.

- Neo4jUserDao: this class contains the Neo4j driver and implements all the CRUD operations and Cypher queries needed by the service layer on the User node and its relationships.

6.2.4 com.unipi.data.mining.backend.data

This package implements the data layer.

- ClusterValues: this class is needed by the service layer for storing the summary of the survey values for clustering and computing the distances between users.
- Distance: this class is needed by the service layer for storing the distance between two users.
- Login: this class is needed by the service layer to perform the login operation.

6.2.5 com.unipi.data.mining.backend.dtos

This package implements the DTO layer. Each DTO class corresponds to an entity or data class and it is used for the serialization/deserialization at the control layer.

- Mapper: the mapper class implements the mapping between DTO and data/entity classes

6.2.6 com.unipi.data.mining.backend.entities

This package implements the model layer.

6.2.6.1 com.unipi.data.mining.backend.entities.mongodb

This package contains the entities that correspond to the MongoDB documents.

6.2.6.2 com.unipi.data.mining.backend.entities.neo4j

This package contains the entities that correspond to the Neo4j nodes and relationships.

6.2.7 com.unipi.data.mining.backend.repository

This package contains the Spring Data MongoDB repository, which implements CRUD methods automatically. It is possible to specify a query with the @Query annotation.

6.2.7 com.unipi.data.mining.backend.service

This package implements the service layer.

- Utils: this class implements utility methods needed by both clustering and user service.

6.2.7.1 com.unipi.data.mining.backend.service.clustering

This package implements all the clustering logic.

- ClusteringService: this abstract class contains all the instances of classes needed for the clustering and implements some function that are needed by both standard clustering and MOA clustering.

- Clustering: this class extends ClusteringService and contains the functions needed for the standard KMeans clustering.
- MOAClustering: this class extends ClusteringService and contains the functions needed for the WithKMeans streaming clustering.

6.2.7.2 com.unipi.data.mining.backend.service.db

This package implements the user service layer.

- EntityService: this abstract class contains all the instances of classes needed by the UserService class.
- UserService: this class extends EntityService and implements all the business logic between the Controller Layer and the DAO Layer.

6.2.7.3 com.unipi.data.mining.backend.service.exceptions

This package contains custom defined Runtime Exceptions.

6.2.8 resources

This package contains the application resources

- script.py: this is the script used for the attribute selection
- application.properties: this configuration file contains the credentials for the database access and the configuration properties for the clustering.