This project has received funding from the European Union's Horizon 2020 reseapiarch and innovation programme under grant agreement No 870811

SPICE

Social cohesion, Participation, and Inclusion through Cultural Engagement

D3.5: Prototype clustering techniques

Deliverable information	
WP	WP3
Document dissemination level	PU Public
Deliverable type	R Document, report
Lead beneficiary	UCM
Contributors	UCM
Date	24/04/2022
Document status	Final
Document version	V1.0

INTENTIONALLY BLANK PAGE

Project information

Project start date: 1st of May 2020

Project Duration: 36 months

Project website: https://spice-h2020.eu

Project contacts

Project Coordinator

Silvio Peroni

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Department of Classical Philology and Italian Studies – FICLIT

E-mail: silvio.peroni@unibo.it

Project Scientific coordinator

Aldo Gangemi

Institute for Cognitive Sciences and Technologies of the Italian National Research Council

E-mail: aldo.gangemi@cnr.it

Project Manager

Adriana Dascultu

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Executive Support Services

E-mail: adriana.dascultu@unibo.it

SPICE consortium

No.	Short name	Institution name	Country
1	UNIBO	ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA	Italy
2	AALTO	AALTO KORKEAKOULUSAATIO SR	Finland
3	DMH	DESIGNMUSEON SAATIO - STIFTELSEN FOR DESIGNMUSEET SR	Finland
4	AAU	AALBORG UNIVERSITET	Denmark
5	OU	THE OPEN UNIVERSITY	United Kingdom
6	IMMA	IRISH MUSEUM OF MODERN ART COMPANY	Ireland
7	GVAM	GVAM GUIAS INTERACTIVAS SL	Spain
8	PG	PADAONE GAMES SL	Spain
9	UCM	UNIVERSIDAD COMPLUTENSE DE MADRID	Spain
10	UNITO	UNIVERSITA DEGLI STUDI DI TORINO	Italy
11	FTM	FONDAZIONE TORINO MUSEI	Italy
12	CELI	MAIZE SRL	Italy
13	UH	UNIVERSITY OF HAIFA	Israel
14	CNR	CONSIGLIO NAZIONALE DELLE RICERCHE	Italy

Executive summary

This document reviews the state of the art on community detection algorithms, emphasizing the importance of similarity assessment processes. After reviewing the similarity measures as essential tools to solve the problem of community detection, we describe our experiments with community detection methods with different test domains. We include experiments with synthetic data, and with some of the preliminary data from the SPICE case studies. The community model distinguishes between pre-existing interest groups, or explicit communities, and implicit hidden communities that are detected using community detection algorithms on the information of the user model and the analysis of the individual user *interpretations (pre-processed by semantic textual analysis).* This document also reviews the advances on visualization and explanation of the community model and how it serves to analyze the strength of social connections within a pre-existing group (explicit community), and to support social cohesion across groups, providing understanding of their differences and recognizing what they have in common.

Document History

Version	Release date	Summary of changes	Author(s) -Institution
V0.1	10/03/2022	First internal draft released	UCM
V0.2	18/03/2022	Text revisions	UCM
V0.3	02/04/2022	Internal review in WP3 and Version released to reviewers' partners for final integrations	UCM, UH
V0.4	16/04/2022	Version released for external review (reviewers Luigi Aspirno and Thomas A. Petersen)	UCM
V1.0	24/04/2022	Final version submitted to all partners	UCM
…			

Table of Contents

Introduction

The general aim of the SPICE project is to build social cohesion, both between and within citizen groups, by developing tools and methods to support citizen curation. We define *citizen curation* as a process in which cultural objects are used by citizens as a resource to develop their own personal interpretations. These interpretations are then shared and used ***within and across*** groups to reflect on similarities and differences in perspective. Additionally, citizens can use their interpretations to build representations of themselves and consider their perspective on culture shared with others.

For each case study, the first steps have been the formalization of the user and community models (**D3.1, D3.3**) that include a set of pre-existing interest groups or **explicit communities** (students from a certain school, teachers, members of an association, older people, asylum seekers, children with serious illnesses, children from lower socioeconomic groups, deaf people, and children from different religious and secular communities). In this context, individual user *interpretations (pre-processed by semantic textual analysis, see details in deliverable **D3.2**)* are used to analyze the strength of social connections within a pre-existing group (explicit community), to detect hidden **(implicit) communities** and to support social cohesion across groups, by both promoting tolerance and understanding of their differences and recognizing what they have in common.

Our goal is to find scrutable models to promote reflection about contents, show differences within groups (both explicit and implicit) to tackle preconceptions of homogeneity based on the pre-existing explicit communities; and show similarity between groups to tackle preconceptions of heterogeneity. The resulting community model will support the recommender system that won't be oriented to the typically popular contents or based on providing "more of the same" similar contents to the users (the so called, filter bubble). Instead, community model will support variety and serendipity to the recommendation results.

The input for the processes of the community model is the user model, the content model, and the semantic analysis of user interpretations. Community detection algorithms analyze 2 types of user attributes:

- Personal attributes (general personal and demographic info).
- Interaction attributes resulting from analysis of interpretations, or user generated content.

The community model keeps the list of implicit and explicit communities and enables visualization and explanation processes.

Figure 1 summarizes the processes of inter-group similarity, similarity computation (intra group and inter group similarities) based on semantic models (wikidata and WP6 ontologies) and community detection and explanation algorithms (clustering or graph analysis).

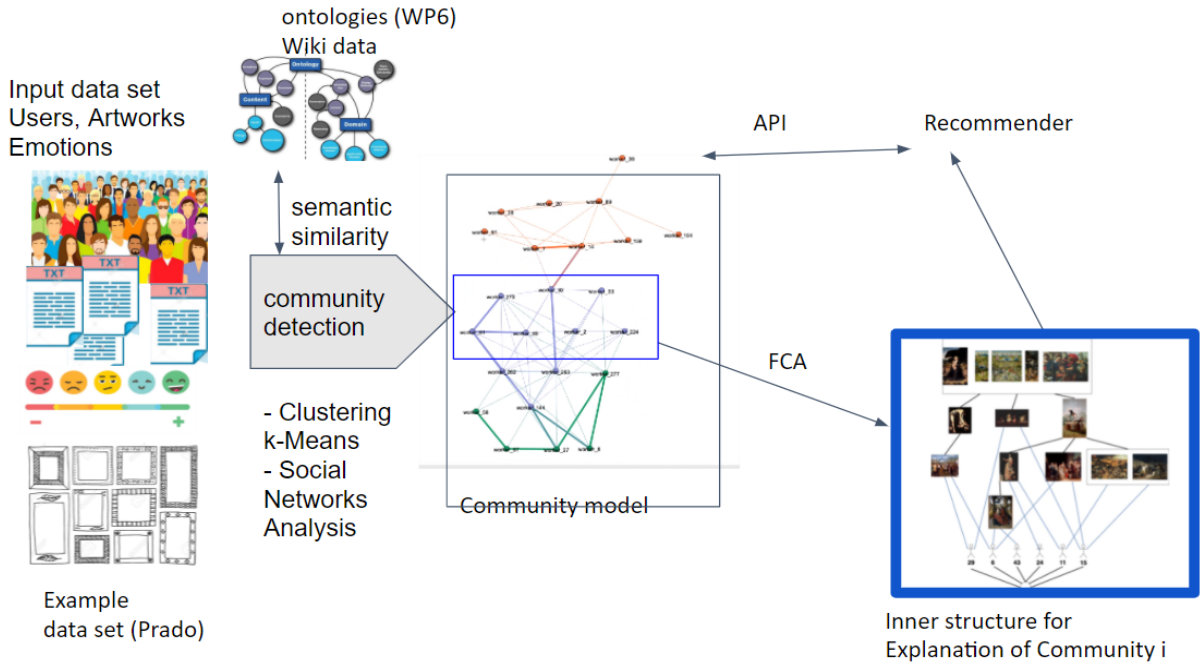

Figure 1: Process for community detection and explanation using FCA (formal concept analysis).

Note that in the processes above we have used visualization and configuration interfaces that are prototypes and are not intended to be the final interfaces.

This deliverable describes algorithms to detect, visualize and explain implicit communities. Membership of users to explicit communities would be asserted directly (and not inferred) and this is useful information in the analysis of preconceptions of homogeneity of opinions inside explicit communities.

Note that there is no clustering algorithm that can be universally used for every type of dataset and there is no similarity measure that can be used by every clustering algorithm on every dataset. Parameter settings are crucial in the performance of a clustering algorithm and similarity configuration (user-user, item-item) affects the results. Good data visualizations help users analyze, validate and explain the clusters generated by the algorithms.

In this document we first review the state of the art on community detection algorithms, emphasizing the importance of semantic similarity metrics in these processes. After reviewing the similarity measures as essential tools to solve the problem of community detection, we describe our experiments with community detection methods with different test domains. We include experiments with synthetic data, and some of the preliminary data from the SPICE case studies.

State of the art of community detection algorithms

In this document we will review a wide spectrum of community detection methods based on two approaches:

1. Clustering algorithms based on similarity functions, considering data points as user information that comprises not only demographics but also user interactions (see Deliverable D3.3 “Final User and Community Models” for more information about the input data in the community model).

2. Clustering algorithms based on graph analysis, considering **the user set as a networked graph** where users are nodes, links represent the existence of similarity relations between users, and the graph analysis algorithms detect compartments with high density links or subgraphs.

Mostly the clustering algorithms to implement the community detection processes was publicly available in machine learning libraries; we have experimented with different configurations and parametrizations in the community detection use case. . Obviously, we could not by any means perform an analysis of all existing

techniques, as their number is huge, but we have reviewed a good representation of them that are the most relevant for our application scenarios.

Clustering algorithms based on similarity functions

Clustering algorithms are non-supervised learning methods to separate the data points contained in a dataset into groups (or *clusters*) in a way that the similarity between the data points in the same group is maximized (internal homogeneity) while the similarity between data points in different groups is minimized (separation) [49]. This way, similar data points are in the same cluster while the clusters themselves are dissimilar among them. Partitioning the dataset into clusters requires in most cases the use of similarity measures (see section Discussion

After the study of different clustering methods, it is worth noting that there is no clustering algorithm that can be universally used for every type of dataset. Additionally, for those clustering methods that use similarity metrics, each dataset needs a study beforehand the selection of the similarity metrics employed because there is no similarity measure that can be used by every clustering algorithm on every dataset.

Parameter settings are crucial in the performance of a clustering algorithm. And some of them require the number of initial clusters as input. Some techniques are employed to define the potential number of these clusters. Additionally, feature selection and extraction are important steps for achieving a good performance in the clustering algorithms.

According to algorithm performance, we want to highlight that every algorithm has its own computational complexity, so it is important to take this into account when choosing a concrete algorithm. This is especially important for clustering algorithms in graph analysis because their complexity is high and depends on the number of nodes or even the edge density. It is also important to note that some clustering algorithms have the capability to rearrange clusters when new data points are added to the dataset without running the algorithm from scratch.

Finally, good data visualizations help users analyze, validate, and explain the clusters generated by the algorithms.

Similarity Measures).

When using a clustering algorithm, we must always follow the following process [76,77]:

1. Feature selection and extraction: it consists of choosing the set of features from the data points employed by the clustering algorithm. New features can also be derived by transforming and combining the original ones.
2. Clustering algorithm selection or design: it consists of choosing the most suitable algorithm according to the features selected in the previous step. This step also implies the definition of the similarity measures and other criterion functions that the algorithm employs.
3. Evaluation: it consists of the validation of the clustering results, using evaluation metrics. These metrics can be employed to compare results from different algorithms or to tune the input parameters of an algorithm.
4. Explanation: it consists of generating a meaningful interpretation of the clusters created by the algorithm.

This is not a one-shot process because evaluation and explanation results can trigger new trials and repetitions to find the most suitable partitions [77].

Factors for choosing a clustering algorithm

According to [25,49], several factors influence the selection of the clustering algorithm:

- The domain where the algorithm is applied.
- The size and sparsity of the dataset.
- The dimensionality of the dataset, i.e., the number of features for each data point.

- The type of features employed to describe the data points in the dataset.
- The correlation among these data features.
- How the algorithm deals with outliers and noisy data.
- The time complexity of the algorithm.
- The stability of the algorithm or, in other words, if the algorithm generates the same clusters despite the order in which the data points are provided to the algorithm.
- The use of different similarity measures.

Classification of clustering algorithms

According to [25,49] clustering algorithms can be classified in the following categories:

- Partition-based clustering.
- Hierarchical Clustering.
- Density-based clustering.
- Grid-based clustering.
- Model-based clustering.

The survey detailed in [76] adds some additional categories that we consider also interesting for our state-of-art review:

- Fuzzy-based clustering.
- Graph-theory-based clustering: these algorithms will be discussed in our section related to Clustering algorithms based on graph analysis.
- Ensemble-based clustering.

This categorization includes clustering algorithms for specific kinds of data such as data streams, spatial data, or large-scale datasets (Big Data). Additionally, there are categories for algorithms based on kernels, fractal theory, distribution, swarm intelligence, quantum theory, affinity propagation or spectral graph theory. These categories are very specific and some of the algorithms included here can also be classified in any of the categories previously enumerated.

Partition-based clustering

These methods classify the data points into k clusters, where all clusters are not empty, relocating the center of each cluster iteratively until an objective function based in inter and intra cluster similarity is satisfied.

K-means [48] is the best-known partition-based clustering algorithm. Initially, it randomly selects k points as the center of the clusters and assigns each point to the nearest cluster. This process is repeated iteratively until a convergence criterion is satisfied. The center of the resulting clusters can be a non-existing data point. The algorithm heavily depends on the number of clusters and the initial centers randomly chosen. Some algorithms, like ISODATA [6], estimate the initial value of k. Additionally, the algorithm is sensitive to noise and outliers. FCM (Fuzzy C-Means) clustering is a fuzzy version of k-means. Fuzzy clustering is also referred to as soft clustering and its first version was proposed by J.C. Dunn in 1973. It is a form of clustering in which each data point can belong to more than one cluster.

k-medioids [59] is a variation of the k-means algorithm where the center of a cluster is represented by the nearest data points to the real center. For this reason, it is more robust to noise and outliers. PAM, CLARA and CLARANS are algorithms based on k-medoids. PAM (Partitioning Around Medoids) [42] tries to minimize the average dissimilarity of data points to their closest cluster medoids (or points that represent the center of the cluster) using a greedy algorithm. CLARA (CLustering Large Applications) [42,66] uses a sampling approach to select the medoids and uses PAM algorithm internally. CLARANS [56,66] is also based on k-medoids to identify spatial structures present in the dataset.

k-modes [37] enhances k-means algorithm using it with data points that combine numerical and categorical features, replacing the means of clusters with modes and a frequency-based method.

Partition-based algorithms are, in general, simple, and efficient and they have a low time complexity. However, they can be drawn in a local optimal solution due to their sensitivity to the initial preset of clusters and centroids.

Hierarchical Clustering

These clustering methods create a hierarchy of the data points in the dataset. The result of this type of algorithms is a tree-like structure called *dendogram*, where the data points are represented by leaf nodes. Different sets of clusters are extracted from this structure, depending on the depth level where the dendogram is cut.

Hierarchical methods can be:

- Agglomerative: These methods start with one cluster per data point, and they iteratively merge two or more clusters until a termination condition is satisfied. These methods are also known as *bottom-up*.
- Divisive: These methods start with one cluster (the whole dataset), and they iteratively split the clusters until a stopping criterion is reached. These methods are also known as *top-down*.

Agglomerative algorithms need to estimate the similarity between clusters to choose the clusters that will be merged on each iteration. The main inter-cluster similarity measures, also known as *linkages*, are the following:

- Single linkage: Cluster similarity is computed as the distance between the closest pair of data points in target clusters.
- Complete linkage: Cluster similarity is computed as the distance between the farthest pair of data points in target clusters.
- Average linkage: Cluster similarity is computed as the average distance between the data points in both clusters.

Most hierarchical clustering algorithms follow an agglomerative approach. For example, BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [82] is an agglomerative algorithm that creates a clustering feature tree in an incremental and dynamic way, so it performs well with large datasets. However, it is sensitive to the order of the data points and does not work well if clusters are not spherical. CURE (Clustering Using REpresentatives) [32] is another agglomerative algorithm, robust to outliers, that identifies clusters having non-spherical shapes and wide variances in size in large datasets employing a combination of random sampling and partitioning. ROCK (RObust Clustering using linKs) [33] is an agglomerative algorithm that deals with data with Boolean and categorical attributes using the concept of links and common neighbors, so it can also be considered as a graph theory-based algorithm. The same consideration occurs with CHAMELEON [41], which uses a graph partition clustering algorithm and the interconnectivity and closeness to build the clusters.

Divisive algorithms are not commonly used due to their complexity. DIANA (DIvisive ANAlysis Clustering) and MONA (MONothetic Analysis) [42] are the only divisive hierarchical clustering algorithms found in literature. These algorithms perform well with arbitrary shapes and data types. Additionally, they normally show high stability and the relationships among clusters can be extracted by analyzing the dendogram. However, it is necessary to preset the number of clusters and some of them suffer from high time complexity.

Density-based clustering

Density-based algorithms split the dataset into regions of high density that contain the data points that belong to this cluster. These algorithms can discover clusters of arbitrary shapes and they are robust to outliers.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [24] is one of the most popular density-based clustering algorithms. It was designed to discover clusters of arbitrary shapes. The main idea is the neighborhood of a datapoint that belongs to a cluster contains a minimum number of points or density. The

algorithm distinguishes between core points and border points because the minimum number of points will be different in both cases.

OPTICS (Ordering Points to Identify the Clustering Structure) [3] produces a density-based clustering ordering, that can be represented by a reachability plot. Then, a hierarchical cluster structure can be generated using this ordering.

DENCLUE (DENsity-based CLUstEring) [36,38] is another density-based clustering algorithm modeling density as the influence a datapoint has within its neighborhood. Clusters can be determined mathematically by identifying density attractors. Evaluation results reveal that it generates good clusters even in data sets with large amounts of noise and it works faster than DBSCAN.

Mean-shift [19] analyzes feature spaces derived from real data, where the number of clusters is not known and each cluster has its own shape. It employs a mean shift procedure to detect modes in the feature space and delineate clusters based on the location of these modes. It is employed in image segmentation, to find clusters of pixels with similar colors.

These algorithms are suitable for arbitrary shapes but most of these algorithms are sensitive to the input parameters. Additionally, the cluster quality is reduced when the density space is not even.

Grid-based clustering

These algorithms modify the original dataset and transform it into a grid structure of fixed size to reduce the processing time. Grid-based algorithm complexity is independent from the number of data points, and it is determined by the number of cells in the grid structure, instead. Some of these algorithms are considered both density-based and grid-based clustering algorithms.

STING (a **St**atistical **In**formation **G**rid approach) [72] creates a spatial grid with a hierarchical structure of cells. It explores statistical information stored in grid cells. Each level in the hierarchy corresponds to a different resolution so the statistical information of cells in higher levels can be computed as an aggregation of the statistical information of the cells in a lower level. Its complexity depends on the number of cells on the lowest level and authors highlight that this algorithm outperforms others like DBSCAN while maintaining a similar quality of the discovered clusters.

Wave Cluster [67] is another grid-based algorithm that considers the multidimensional spatial data as a multidimensional signal and it uses a wavelet transform method, a signal processing technique that decomposes a signal into different frequency sub-band, to convert the spatial data into the frequency domain. It does not assume a specific shape for the clusters, and it is not affected by outliers, and it expects an estimation of the number of clusters. It outperforms other density-based algorithms with similar cluster qualities.

CLIQUE (**CL**ustering **I**n **QUE**st) [1] is another algorithm based on automatically identifying the subspaces of high dimensional data that allow better clustering than original space. It partitions each dimension in the original data space into non overlapping rectangular cells and a cluster is defined as the maximal set of connected dense cells. Due to its simplicity, it runs faster than other algorithms, but it shows some problems with the quality of the clusters discovered.

In general, these algorithms are faster and have a high scalability, and they are suitable for parallel processing. However, they are sensitive to cell size and structure, and they have problems of accuracy and low quality of clusters.

Model-based clustering

These algorithms assume that each cluster in the dataset can be modeled by a mathematical model, so the algorithm looks for the model that best fits the dataset. There are two main approaches: based on **statistical learning** and based on **neural networks**.

COBWEB [26] is an example of model-based algorithm based on statistical learning. It creates a classification tree based on an incremental method for concept learning. This tree can be interpreted as a hierarchical

clustering structure, where nodes represent concepts using a statistical description. Its main drawback is its complexity and the assumption that the attributes are independent. However, this algorithm has been extended by others like CLASSIT [28], for continuous data, or AutoClass [15], which uses Bayesian statistical analysis to estimate the number of clusters.

EM (Expectation-Maximization) methods are also employed as model-based algorithm based on statistical learning. EM iteratively refines the maximum likelihood parameter estimation by calculating the expectation while keeping a set of fixed parameters. EM is employed in MCLUST [61], a software package for software clustering in FORTRAN for parameterized gaussian mixture models, and the performance of different variations of this algorithm are compared in [50].

SOM (Kohonen's Self-Organizing Map) [45] is one of the most popular clustering algorithms based on neural networks. It learns patterns in data by adjusting their interconnection weights to best fit the data. Other works, like the Clusot algorithm [14], use SOM and alleviate its inherent interpretation problem for nonexpert users.

Model-based clustering algorithms can describe more specifically the data in some concrete areas, providing a significant advantage over other clustering algorithms. However, generally, model-based algorithms suffer from high complexity. Additionally, the premise cannot be correct for certain datasets, and they are sensitive to model parameters.

Fuzzy-based clustering

All the algorithms described up to now classify each data point in a cluster. However, fuzzy clusters allow a degree of membership to different clusters for each datapoint [79]. The most popular algorithm of this family is Fuzzy C-Means (FCM), based on k-means [10]. As a partition-based algorithm, it iteratively looks for the center of each cluster but, instead of classifying a datapoint strictly to a cluster, it assigns a value ranging between 0 and 1 to measure the likelihood to belong to a cluster. Fuzzy C-Shells (FCS) [20], or the approximation of cluster centers using mountain methods [78] are other examples of this family of clustering methods

The main advantage of these algorithms is that it is more reasonable to classify a datapoint to different clusters than to a unique group, so they obtain a relatively high accuracy. However, they inherit the problems of the partition-based algorithms, like the sensitivity to initial parameters and the need of presetting the number of clusters.

Ensemble-based clustering

This type of technique is based on aggregating the results generated by of a set of clustering algorithms [71]. This set can be a combination of different clustering algorithms, the same algorithm with different parameters or initial conditions, different representations of the data points, different subsets of data points or executions using different feature spaces. The function that combines the results is known as consensus function and it is responsible for improving the results of the individual clustering algorithms. The surveys in [71,76] contain a classification and a catalog of several consensus functions.

These algorithms are scalable, and the execution of different algorithms can be done in parallel. Additionally, the ensemble benefits from the strengths of the algorithms employed. However, the main problem of this type of methods relies on the definition of the consensus function and on the selection of the initial clustering methods. If they are not carefully selected, the results provided by each one can be incompatible and should not be combined by the consensus function.

Evaluating clustering algorithms

To test the quality of the clustering algorithms we can use different metrics. These evaluation metrics can be categorized into two groups based on the information contained in the dataset: internal and external indices [25,49,76].

Internal indices evaluate the goodness of a clustering algorithm in terms of the features known from the dataset. These assessment indices are applicable in the algorithms that use the concept of centroid. Some of these metrics are the following:

- *Compactness* measures the average distance between data points in the same cluster with respect to the centroids. Good clustering algorithms will create clusters with high compactness.
- *Separation* measures the distance between different clusters computing the distance between centroids. Good clustering algorithms will create clusters with high separation.
- *Davies-Bouldin Index* measures the overlap between clusters. If this index is close to 0 then the clusters are separated and compact.
- *Dunn Validity Index* combines separation and compactness. A large value of this index indicates that the clusters are compact and well separated.
- *Silhouette coefficient* also computes a combination of the average distance between a data point and other data points in the same cluster and average distance between nearest clusters to measure the goodness of the clustering algorithm.

Additionally, clustering algorithms can be evaluated in terms of *time complexity*, especially when dealing with big amounts of data, and *stability*, that measures the variation of the outputs when running the algorithm more than one time.

If the clusters are known, i.e. we have assigned the class labels in the dataset, we can assess the performance of a clustering algorithm using the same evaluation techniques employed in supervised learning, like Accuracy, Confusion Matrix, Precision, Recall and F-measures. However, there are specific measures, called external indices, for clustering algorithms:

- *Cluster accuracy* is the percentage of data points in the correct cluster.
- *Rand Index* and *Adjusted Rand Index* considers not only the data points in the correct cluster but also the data points that are classified in a different cluster by the algorithm.
- *Normalized Mutual information* is based in information theory, and it computes the amount of information shared by two clusters.
- *Fowlkes–Mallow's index* is based on the number of correct and incorrect data points classified by the clustering algorithm. It is computed as the geometric means of precision and recall.

Clustering algorithms based on graph analysis

This category refers to the algorithms that are applied to graphs, where nodes represent data points and edges represent relationships among data points. According to graph theory, communities or clusters are groups of nodes having similar properties, affiliations or roles that are different from other nodes in the network [27,39]. Therefore, community detection refers to the methods employed to identify these groups using the structural properties of the graph.

Community detection algorithms are based on the concept that the number of edges among nodes inside the community is larger than the number of edges with nodes in the rest of the graph. For this reason, community detection algorithms are effective only if graphs are sparse, i.e., if the number of edges is of the order of the number of nodes of the graph [27].

Most of the algorithms proposed for community detection were designed to discover disjoint communities. However, during the last few years, the number of alternatives to discover overlapping communities has increased.

Algorithms for disjoint communities

These algorithms assign each node to a unique cluster. According to [39], these algorithms can be classified into the categories described in the following subsections.

Traditional algorithms

Traditional algorithms were the early methods employed for community detection in graphs. These algorithms are categorized in the following techniques:

- **Graph partitioning**: These algorithms split the nodes of a graph into a concrete number of groups, with a predefined size, in a way that the number of the edges among clusters is minimum. The Kernighan-Lin (KL) algorithm uses a heuristic approach that minimizes the difference between the intra-community and inter-community links [43]. Their main drawback is that the number and size of the clusters is needed beforehand.
- **Hierarchical graph clustering**: Like in the hierarchical clustering algorithms described previously, these algorithms create a hierarchical structure (a dendogram) of the graph identifying groups of similar nodes. On one hand, agglomerative algorithms start with nodes in separate clusters, and they are combined iteratively according to a similarity score [68]. On the other hand, divisive algorithms start with all the nodes in one cluster that will be split iteratively removing edges that connect nodes with low similarity. One of the most important methods following this approach is the Girvan-Newman [53]. This method removes edges with the highest betweenness, those that connect nodes from different communities. Once the dendogram is created, a graph partition is selected using the split with highest modularity.
- **Partitional clustering**: These algorithms split the network into a predefined number of K clusters using the distances among nodes [63]. The goal is to maximize/minimize a given function that uses that distance. Some examples of this distance function are:
  - Minimum k-clustering: It uses the diameter of a cluster, which is the largest distance between two points of a cluster.
  - k-clustering sum: It uses the average distance between all pairs of points of a cluster.
  - k-center: It computes the maximum distances of each cluster node from the centroid
  - k-median: It computes the average distance of each cluster in the node with the centroid.

  The main limitation of these algorithms is the specification of the number of clusters at the beginning.
- **Spectral clustering**: These algorithms use the eigenvectors of a similarity matrix extracted from a graph. These eigenvectors reveal implicit properties hidden in the former matrix. The most common methods of this category are unnormalized spectral clustering with Laplacian matrix [60], normalized spectral clustering with symmetric Laplacian matrix [40] and normalized spectral clustering with a Laplacian matrix of random walks [55].

Modularity-based algorithms

Many algorithms try to discover communities defining groups that maximize a defined quality function. The most popular function for assessing the quality of a group of nodes is *modularity* [54], which represents the actual density of edges in a cluster compared with the expected density in a random graph.

Modularity optimization can be achieved using different alternatives like simulated annealing [47], extremal optimization [13], spectral optimization [16], or genetic algorithms [17], among others [27,39]. However, one of the most popular are greedy algorithms [18,53], like the Louvain Method [11]. This algorithm is simple and tries to maximize the modularity using an iterative process where nodes are aggregated in other communities if the modularity of the new community is enhanced.

Dynamic algorithms

These algorithms discover communities running processes on the graph. The most common types of processes employed are the following:

- **Spin models**: these methods use the Potts model, that describes a system of spins in different states (nodes) and the interactions between them (nearest neighbors) [65].
- **Random walks**: these methods use random walkers to find communities, supposing that a random walker should spend more time inside a community due to the high density of internal edges. Random

walks are employed to define a distance measure, so close nodes are supposed to belong to the same community [83]. One of the most used algorithms of this kind is Markov Clustering Algorithm (MCL) [23], which simulates a flow diffusion method in a graph using random walks.

- **Synchronization**: these methods suppose that nodes are oscillators and nodes in the same community will synchronize earlier [12]. These methods are not reliable when communities differ in size.

Algorithms for overlapping communities

These algorithms can classify a node in more than one group [74]. Some community detection algorithms that generate disjoint communities (described previously) have been adapted to detect overlapping communities, like CONGA (Cluster-Overlap Newman Girvan Algorithm) [30], which extends the Girvan-Newman algorithm. However, there are also specific methods for finding overlapping communities, as we will see in the following subsections.

Label propagation

These methods define communities as groups of nodes that share the same label. The label of a node is defined using an iterative propagation process based on a neighbor majority voting, known as Label Propagation Algorithm (LPA) [62]. COPRA [31] and SLPA (Speaker-Listener Propagation Algorithm) [75] are well-known example of community detection method that employs the LPA.

Clique Percolation Method

It is based on the concept that the internal edges of a community are likely to form cliques due to their high density [58]. A k-clique is a complete graph or k nodes, where every node is linked with all the nodes in the graph. The k-cliques are translated over the graph to find new communities and adjacent cliques can share nodes. The algorithm has been extended to the analysis of weighted, directed and bipartite graphs and faster implementations have been developed [46]. Its main drawback is the need to predefine the k value for running the algorithms.

Local expansion and optimization

These methods are based on the maximization of a local function that promotes the quality of densely connected nodes. The work in [7] proposes this method, that uses Iterative Scan (IS) for greedy optimization, and Rank Removal (RaRe), for removing vertices with highest importance, determined by a centrality score like betweenness centrality or PageRank.

Link partitioning

Link partitioning-based methods identify overlapping community structures by partitioning links instead of nodes. They convert the original network into a line graph and then identify the non-overlapping link communities using disjoint community detection methods. When the final line graph is converted again in the original graph, nodes can belong to multiple communities. Link clustering (LC) [2] or map equation for link communities (MELC) [44] are examples of this kind of methods.

Statistical inference-based methods

These methods aim to deduce properties of a graph based in a model that assumes some connectivity patterns among nodes. Generative models based on Bayesian inference are an example of this kind of methods. Early works [35] chose as model the planted partitioning model, which supposes that the clusters have equal size, and a pair of vertices is connected with probability p if they belong to the same cluster. Further works employ different models, like the Dynamic Stochastic Block Model (DSBM) [80] or the Dynamic Bayesian Overlapping Community Detector (DBOCD) [29], among others.

Other methods are based on stochastic block models. Block modeling is a common approach in social network analysis to decompose a graph in classes of nodes with common properties. In this way, a simpler description of the graph is attained. Structural equivalence organizes nodes in classes where the probability of an edge between a node with all other nodes of the graph are the same for nodes in the same class. The work in [64] describes the use of this model.

Nonnegative matrix factorization approaches.

Nonnegative matrix factorization (NMF) is a recent approach for the discovery of structural properties of graphs with overlapping communities. NMF improves interpretability in spectral methods that use eigenvalues, whose meaning is hard to explain. Some variants of this approach are described in [39].

Fuzzy methods

Fuzzy community detection algorithms assign each node a soft membership factor vector to communities. Zhang et al. [81] proposed an algorithm based on the spectral clustering framework that uses Fuzzy C-Means (FCM) to obtain a soft assignment of nodes to one of the parameter-specified k communities, The work in [52] uses simulated annealing for optimizing a function that employs node similarity and fuzzy memberships of nodes to communities. Other example of these kind of methods is the work in [73], which uses a disjoint community detection method with a local optimization for assigning community memberships to graphs.

Discussion

After the study of different clustering methods, it is worth noting that there is no clustering algorithm that can be universally used for every type of dataset. Additionally, for those clustering methods that use similarity metrics, each dataset needs a study beforehand the selection of the similarity metrics employed because there is no similarity measure that can be used by every clustering algorithm on every dataset.

Parameter settings are crucial in the performance of a clustering algorithm. And some of them require the number of initial clusters as input. Some techniques are employed to define the potential number of these clusters. Additionally, feature selection and extraction are important steps for achieving a good performance in the clustering algorithms.

According to algorithm performance, we want to highlight that every algorithm has its own computational complexity, so it is important to take this into account when choosing a concrete algorithm. This is especially important for clustering algorithms in graph analysis because their complexity is high and depends on the number of nodes or even the edge density. It is also important to note that some clustering algorithms have the capability to rearrange clusters when new data points are added to the dataset without running the algorithm from scratch.

Finally, good data visualizations help users analyze, validate, and explain the clusters generated by the algorithms.

Similarity Measures

Similarity and its complementary notion of distance are essential to solve a broad range of AI domains and applications, including the problem of community detection, since they can serve as an organizing principle by which individuals classify objects, form concepts, and generalize.

From a general perspective, similarity between a pair of objects would be typically computed using description features by attribute-value pairs. These features can be simple, textual or, in some applications, it may be necessary to use derived features obtained by inference based on domain knowledge. In yet other applications, objects are represented by complex structures (such as graphs or first-order terms) and require an assessment of their structural similarity. *Community detection* relies heavily on the definition and use of semantic similarity measures over complex graph structures representing citizens, opinions, artworks, contributions, reflections, and emotions.

In the SPICE project we have defined similarity measures between users and items based on different types of attributes (see also Deliverable D3.3: “Final User and Community Model”):

- From the user model
  - **Personal features (or attributes):** according to Deliverable Document D3.3 “Final User and Community Models”, user models represent the individuals that are interacting with the system.

Users are described using different types of attributes (demographic, cultural, skills...) contained in the user model. For example, User1 (age 22) (nation Italy) (religion Catholic).

- **Interaction attributes:** Content/emotions analysis module (D3.2 "Semantic annotation of social curatorial products") generates information about user interests based on emotions, sentiment, attitudes, and others (see also D6.3). This information is stored in the user model as user generated content as users interact with items generating their own content (reflection, opinion, interpretations, comments, reaction to opinions…). Interaction attributes link users and items. These interactions can be represented as a tuple (user, item, interaction value), where the interaction value can be an emotion, a positive-negative rating, a conceptual value, etc. For example, (user1 likes Item1; user1 hates Item1; Item1 evokes fear on user1...). Note that users can generate content about subjects (or beliefs) that are related to a museum activity instead of a specific item from the content model. We consider this also as interaction attributes, represented as a tuple (user, activity-concept, interaction value) and stored in the user model. For example, in MNCN, (user1 climate-change fear).

- From the content model.
  - Museums store collections of items or organize activities with associated subjects (e.g., sustainability). According to Deliverable Document D4.1 "Distributed Linked Data Infrastructure", the linked data infrastructure supports the storage of museum collections using the SPICE Ontology Network (cf. D6.2 and D6.5). Similarity between items is important as it contributes to the user similarity. For example, user1 and user2 are similar due to their personal attributes, but also because user1 has positive rating (polarity/emotion) with item1, user2 has positive rating with item2 and item1 and item2 are similar (using an item-item similarity measure).

Related work about Similarity

From psychology [22], we have different theoretical approaches to similarity: Common element approach, Template models, Geometric models, Feature models and Geon theory, among others. Most of the typical similarity measures used in computer science refer to geometric and feature models.

The geometric approach stresses the representation of similarity relationships among the objects.

Using a set of features, similarity is given by the distance between objects in this space; the closer two objects are, the more similar they are. This approach assumes (1) that objects can be represented by values on a few continuous dimensions (features) and (2) that similarity can be represented by distance in a coordinate space. Common element approach computes similarity using only the proportion of common elements. The well-known Tversky's "Contrast Model" (1977) [70] systematizes this feature-based approach highlighting the fact that similarity depends not only on the proportion of features common to the two objects but also on their unique features. In AI the notion of similarity, or distance, between objects described by attributes, has long been used in many contexts. Different similarity measures have a performance strongly related to the type of the features (or attributes) representing the compared objects and to the importance of each attribute. Thus, it is very different to deal with only continuous data, with ordered discrete data or non-ordered discrete data. Besides we can distinguish between surface and structural similarity.

While distance functions for propositional (i.e., feature-vector) representations have been thoroughly studied in the past, work on distance functions for structured representations has been carried out in different communities such as graph matching, inductive logic programming, case-based reasoning, relational learning, or graph mining and is much less understood. Specifically, a significant amount of work that requires the use of a distance or similarity function for structured representations of data usually employs ad-hoc functions for specific applications [57].

When we need to compare object-oriented structured representations, similarity assessment should allow us to compare two differently structured objects. The structure of an object could be described by an object class that defines the set of attributes (also called slots) together with a type (set of possible values or sub-objects) for each attribute. Object classes are arranged in a hierarchy of classes, that is, usually a n-ary tree in which sub-classes inherit attributes as well as their definition from the parent class (predecessor).

Moreover, we distinguish between simple attributes, which have a simple type like Integer or Symbol, and so-called relational attributes. There are some standard measures to compare simple values like numeric, strings or symbols (see next section). Relational attributes hold complete objects of some (arbitrary) class from the class hierarchy. They represent a directed binary relation, e.g., a part-of relation, between the object that defines the relational attribute and the object to which it refers. Relational attributes are used to represent complex case structures. The ability to relate an object to another object of an arbitrary class (or an arbitrary sub-class from a specified parent class) enables the representation of cases with different structures in an appropriate way. Similarity measures for such object-oriented representations are often defined by the following general scheme [9]: The goal is to determine the similarity between two objects. The object similarity is determined recursively in a bottom-up fashion, i.e., for each simple attribute (for example, age), a local similarity measure determines the similarity between the two attribute values (18 and 34), and for each relational slot an object similarity measure recursively compares the two related sub-objects. Then the similarity values from the local similarity measures and the object similarity measures, respectively, are aggregated (e.g., by a weighted sum) to the object similarity between the objects being compared (see Figure 2).

When dealing with conceptual background domain models, like graphs, networks or taxonomies, another possibility is the representational approach that assigns similarity meaning to the path joining two individuals. In general, a graph-based semantic similarity measure is a mathematical tool used to estimate the strength of the semantic interaction between entities (concepts or instances) based on the analysis of ontologies [34]. Similarity is computed for a given pair of individuals. An individual is defined in terms of the concepts of which is an instance and the properties asserted for it, which are represented as relations connecting the individual to other individuals or primitive values (fillers). Note that the application of this measure is strongly dependent on the availability of an ontology or conceptual model that represents the application domain. For comparing ontological entities, graph-based measures are classified into two basic approaches: path-based, which compare the concepts according to properties of paths in graphs, and node-based, that use properties of concepts in the ontology graph for comparing concepts. In path-based approaches, concepts are compared according to properties of paths in graphs. The most common property is the shortest path that connects nodes in an ontology (or concept taxonomy). The shorter the path is, the higher the similarity is. The path-based approaches suffer from a significant drawback: they typically consider all edges equivalent, indicating a uniform distance. Concerning the node-based approaches, they use properties of concepts in the ontology graph for comparing concepts. We cite some of the most well-known measures, which are based on the lowest common subsumer (LCS) property: Armengol and Plaza's [5], Bergmann [8], Resnik's [10] and Lin's [11] measures. The main limitation is that they are applicable only on taxonomies. The work in [57] classifies the distance and similarity functions on graph-based representations in four types: (1) graph matching, (2) based on edit distances, (3) based on the types of relationships and refinement operators and (4) based on kernels. Two main categories of graph-based semantic measures are distinguished: (1) similarity measures adapted to taxonomies and (2) relatedness measures adapted to semantic graphs composed of different types of relationships [34] [4].

Next section describes what is the use of similarity in the SPICE project.

Similarity computation for SPICE community model

In SPICE we have designed community detection algorithms that rely on assessing *similarity between users* (user-user). Similarity between users is computed combining similarity between users' attributes from the user model (personal and interaction attributes). Similarity between two users is defined as a linear combination of the local similarity values results between the user personal attributes $UserAtr_i$ (**Errore. L'origine riferimento non è stata trovata.**).

Figure 2 Content similarity as a linear combination of local similarity values

Note that the set of attributes describing user1 and user2 could be different. This type of **content similarity** compares the common properties of the two objects. Besides, as we have the list of explicit communities for each user, and communities are organized in a taxonomy (see D6.5), we can combine content similarity with **similarity by position**.

Personal attributes (UserAtr) belong to the user model (see D3.1 for more details on the user model).

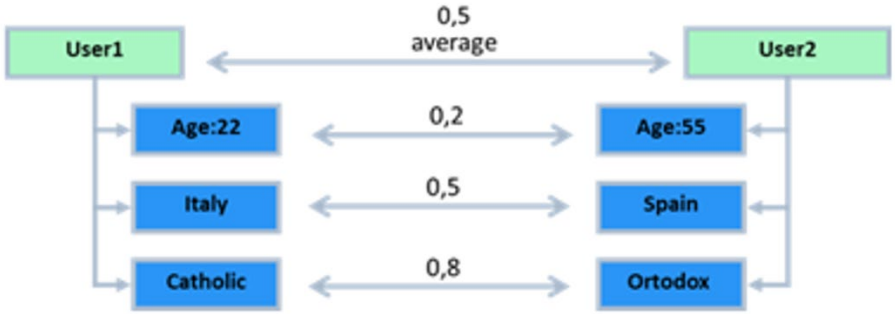

Figure 3 Similarity using personal attributes

Content similarity also considers *interaction attributes* atr_i, for example, user1 likes $Item_1$; $user_1$ hates $Item_1$; $Item_1$ evokes fear on $user_1$. In the following example, Ugo watches in the Prado the Francisco-Goya painting "saturn-devouring-one-of-his-children". He "likes" it and tags it with the comment "Scary!". Another user, Anne, likes the painting "The 3rd of May" 1808 also from the same painter Goya. Similarity between Anne and Ugo using interaction attributes will evaluate if they have similar interactions (same sentiment or emotion) on similar artworks (similarity item-item). In this example they share similar emotion on similar artworks (both are Goya paintings).

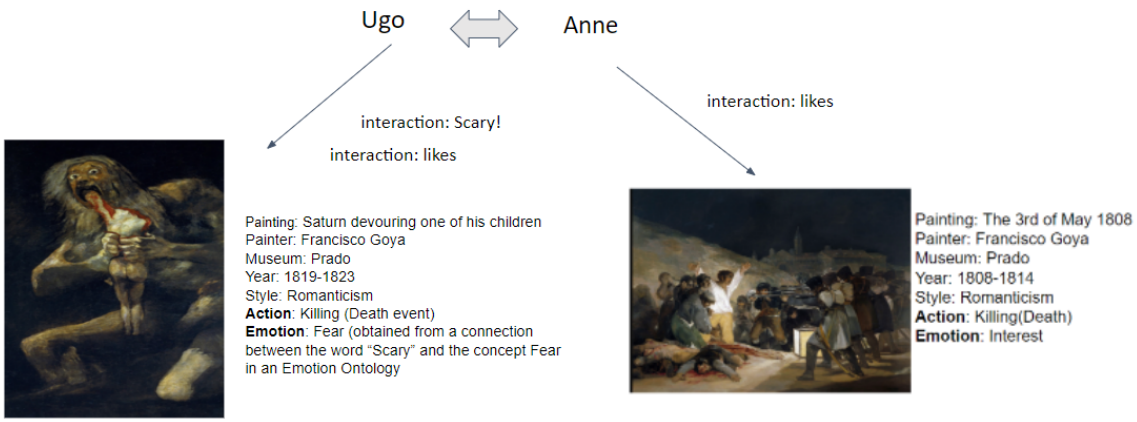

Figure 3 Similarity using interaction attributes

To compute similarity between items (item1-item2) we use the set of descriptive attributes (ItemAtr) (from the content model): author, size, colour, year, artistic movement, style (ARCO ontology WP6). For example, in the Figure 4 example, both artworks are compared using local comparison on their attribute's values (see Figure 4). Note that some attributes (like the theme) are connected and reason with the ontology models described in WP6. Besides, we can combine content similarity for some of the attributes with the use of the similarity measure by position using the taxonomies (see D6.5) for others (like the theme or authors).

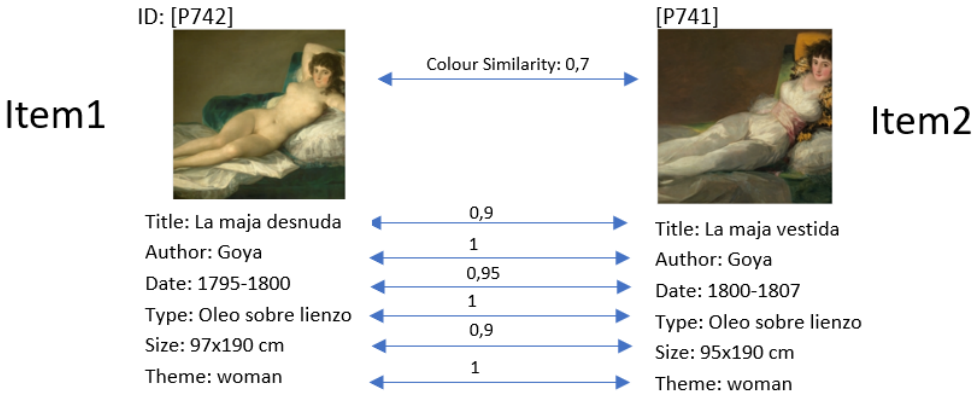

Figure 4. Item-Item Similarity using content attributes

Local Similarity functions

For each application scenario, we would need to specify Local Similarity measures for each user attribute and item attribute. To do so, we use a **catalog of local similarity measures** defined in the attribute types used to compare values (numbers, strings, symbols) between attributes. For example, to compare the age, nationality, or colors. New local similarity measures can be defined when needed (see subsection Design of new similarity measures).

For numeric-valued features, the local (feature-specific) similarity measure most often computes the complement of the absolute difference, normalised to [0,1] by dividing by the range of permissible values:

$$sim_i(x_i, x_i') = 1 - \frac{|x_i - x_i'|}{max_i - min_i}$$

Local similarity measures can be simple or complex (graph or position based).

For example, a simple local similarity measures to compare colors or countries would be the use of a similarity table like the one in Figure 5 or we can use the distance if they are in a taxonomy (see subsection similarity in ontologies).

	red	green	cyan	blue
red	1.0	0.1	0.2	0.1
green	0.1	1.0	0.8	0.1
cyan	0.2	0.8	1.0	0.8
blue	0.1	0.1	0.8	1.0

Figure 5: An example of similarity table for colors

When the values of a categorical feature have an ordering, it may be possible to assign integer codes to the values, thus turning the categorical feature into a numeric-valued feature (January=1, February=2). However,

this does assume that the similarity of two categorical values is proportional to the (complement of) the difference between their numeric codes.

Global Similarity functions

We use Euclidean distance and Manhattan distance as two instantiations of the **Minkowski function**:

$$SIM_{Minkowski}(A,B) = \frac{1}{p}\left[\sum_{i=1}^{p} w_i \cdot [sim_i(a_i,b_i)]^r\right]^r$$

Where w_i is the weight or importance assigned to attribute i. We can suppose weights w_i are established manually during the configuration process. In section Perception of similarity, we describe an approach to learn these weights.

When r=1 we have Manhattan distance, when r=2 we have the Euclidean distance and when r=∞ it converges to the Chebyshev distance.

$$SIM_{Chebyshev}(A, B) = max_{i=1...n}|a_i - b_i|).$$

Also, when p = 1 (i.e., when comparing scalars), this similarity function corresponds to the absolute value of their difference.

Another option is the use of the cosine similarity as a global similarity function that computes the cosine of the angle between two vectors (a1, a2, … ap) and (b1, b2, … bp) and it is defined as:

$$SIM_{cosine}(A,B) = \frac{A.B}{|A| \cdot |B|}$$

Similarity based on position on taxonomies

In the previous section, we have described content similarity that is based on the aggregation of local similarity between attributes. A particular situation occurs when the compared values (for example, colors, religions, or countries) are organized in taxonomies. If this is the case, we can use similarity functions based on position. We describe these position similarity functions from a general perspective as they can be used both as local function to compare values of any of the compared attributes; and, to compare users that are in a taxonomy, for example, the taxonomy of explicit communities (see D6.5 - WP6).

Let K be an inner node of a certain concept hierarchy (Figure 6), then LK denotes the set of all leaf concepts from the sub-tree starting at K. Further, K1 < K2 denotes that K1 is a successor node (subconcept) of K2. Moreover, <K3,K4> stands for the most specific common object class of K3 and K4, i.e., <K3,K4> > K3 and <K3,K4> > K4 and it does not exist a node K'< <K3,K4> such that K' > K3 and K' > K4 holds.

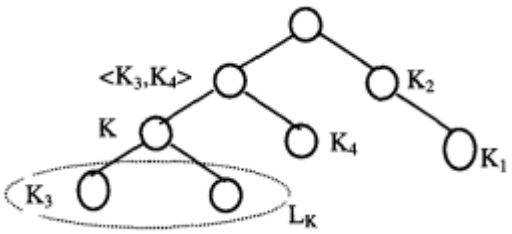

Figure 6: An example of concept hierarchy

Similarity is computed for a given pair of objects where an individual can represent either a user itself or, in general, the value of an attribute. As described before, any individual is defined in terms of the concepts of which that individual is an instance and the attributes asserted for it (if any). In graph representations these attributes can be relations connecting the individual to other individuals or primitive values (like scalars, numbers, or symbols). We then can use different measures depending on the position of individuals.

- One simple similarity by position would be:

$$SIM\ (K_3, K_4)\frac{depth < K_3, K_4 >}{MAX(depth)}$$

For example, in the topic taxonomy (see Figure 7), we compute similarity based on the position of concepts (or individuals):

SIM(bronze age, antiquity) = 2/3 and SIM(bronze age, geometry) = 0

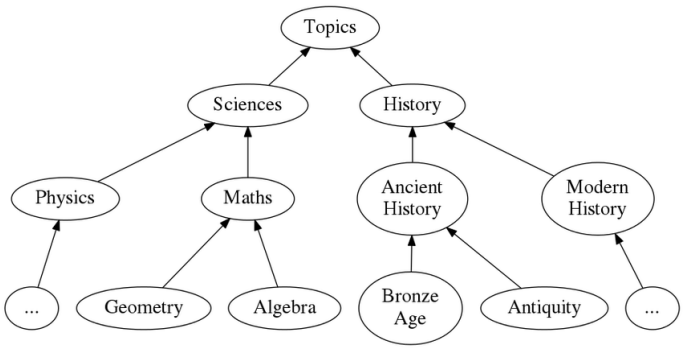

Figure 7: An example of topic taxonomy

- Another option is to use the cosine similarity [69] to reflect the similarity by the position of 2 individuals in a certain taxonomy. We also use the cosine similarity [69] to reflect the similarity by the position of 2 concepts (or individuals) in a certain taxonomy. Note that vectors representing these individuals are not explicitly built. Instead, we use the following formula to compute similarity by position where the similarity between two concepts is given by the number of their common superconcepts, and their total number of superconcepts.

$$SIM(A,B) = \frac{|Interse\ (\sup(A),\ \sup(B))|}{\sqrt{\sup(B)\ .\sqrt{|\sup(A)|}}}$$

Where sup(C) are all the concepts in the taxonomy that are superconcepts of C. In the example (Figure 6) superconcepts of Algebra = {Maths, Sciences, Topics}

- Jaccard measure uses the superconcepts sets and computes the ratio of Intersection over Union. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

We also are exploring how to use similarity metrics based on graph embeddings consider all the information about users, artifacts, interactions, interpretations etc. expressed as a graph (RDF Graph) according to the ontology, and different algorithms (RDF2Vec, RESCAL, Trans-E etc.) might be used for computing embeddings of the nodes of the graph. Then, you can use this vectorial representation of nodes to compute the similarity (e.g. cosine similarity) between two nodes.

All the measures that we have used for community detection are described in Annex: Similarity measures catalog.

Perception of similarity

In previous section we have supposed that the weights that determine the importance of each feature (attribute) has been manually established during the configuration process by the museum curators.

There are other advanced options to learn these weights. Our research [21] proposed a similarity function that reflect perception. Specifically, it reflects either perception of one specific user u, or more interestingly perception of a group of users p sharing a common profile (explicit community). Because this measure is a weighted similarity measure, we also address the fundamental problem of learning a weight model for features. i.e., it is necessary to give a greater similarity contribution to an important attribute than to other less important ones regarding perception of similarity. Regarding perception of similarity related work exists in the field of human psychology, where similarity is defined a relationship that holds between two perceptual or conceptual objects and serves to classify objects, form concepts and make generalizations [70]. As it is noted in [22], similarity between objects is not solely dependent on the characteristics of those objects. It is also affected by the context, and by other present and immediately past stimuli, as well as long-term experience with related objects.

Design of new similarity measures

Although the catalogue offers a variety of similarity measures, it may be needed to define or adjust new similarity measures for a certain case study. This section describes some issues that affect the design of similarity measures. They are most often discussed in the context of similarity measures for vector representations although they can also affect structured or graph representations.

The first issue is that we must ensure that the local similarity functions produce meaningful results when the feature values are null. Note that there can be at least two meanings for null: not-applicable, and applicable-but-unknown. We need to be aware of the specific semantic of null or having different tokens that are handled differently by the local similarity functions or by the global similarity measure's aggregation.

The second issue is aggregating local similarities for numeric-valued features that have different ranges of values. The solution is to scale either the features values or the similarity which is what we did above using min-max scaling by dividing by the range of permissible values.

We need also to consider whether the features are independent of each other, or whether their values are correlated. Informally speaking, if two features are not independent, then their local similarity values result in a degree of double-counting. That is related with the weights used by the global similarity measure when aggregating the local similarity values. Weights come from a domain expert (museum curators); however, this may not be an easy task, because while experts may have a sense of which features are more important than others, rarely will they be able to quantify the importance. An alternative is to learn them automatically: this requires that the similarity measure be used within some tasks for which we have an evaluation measure; then the learning algorithm can search for the weights that give highest validation dataset performance for that evaluation measure.

Application Scenarios

We are experimenting the community model with the case studies of the SPICE project plus a synthetic domain with artificial users of the Prado Museum. We have postponed some of the experimentation work with IMMA, GAM and DMH datasets for next semester. The table in Figure 8 summarizes the data and algorithms employed up to now.

We have preliminary results using data of first experiments defined by the user journeys developed in WP2 Workshop #3 (see also D2.3).

The datasets of case studies Prado Museum, MNCN, HETCH, IMMA and GAM are in the github repository: https://github.com/spice-h2020/prototype-clustering/tree/main/data

Case Study	Dataset			Algorithm	Visualization
	#Users	#Items	#Interactions		
Prado Museum	171	30	1759	Similarity Clustering and Social Network algorithms	Graphs
MNCN	61	-	900	Agglomerative Clustering	Graphs and FCA
HECHT	78	-			
IMMA	8	11	107	Similarity by interactions	Graphs
GAM	128	12		Emotion similarity Clustering	-

Figure 8: Table resuming the Case studies where the clustering algorithms have been applied

In different meetings and following different refinement processes, case studies have identified some of the target communities and demographic clusters (see D2.3 for details) that have been refined and formalized as explicit communities in an ontology (WP6). Note that more target communities can be defined for further experimentations. Some examples of these target communities are senior citizens both free going and residents at a senior care centre, rural dweller families and asylum seekers in DMH case study; school children (more than 12 years old) and teachers from different types of schools in MNCN, Deaf communities in GAM; healthcare workers, Black and Irish, migrant communities in IMMA and students from different types of schools (Jewish/Arab) in Hecht. Next subsections describe the advances on experiments for the different application scenarios.

Prado Museum (synthetic dataset)

In this first scenario, we have used an artificial dataset applied to users and artworks from Museo del Prado of Madrid (Spain). This dataset is an excerpt from Wikiart Emotion Dataset [51]. It contains 30 artworks from the Prado Museum (30) and 1760 annotations of emotions from 171 different users. We started the implementation of the community model tools using this dataset until we obtained data from case studies. We can divide the solutions implemented in this dataset into two groups: community detection based on clustering and community detection based on graph analysis.

Community detection based on clustering

We have implemented a full architecture where we can define different similarities functions to apply in different clustering algorithms. We have defined a common interface for all similarity functions and, based on this interface, we implemented different similarities to compare users and artworks (see Annex: similarity catalog for additional details on the implementation).

In the Community Module the `SimilarityCommunityDetection` class is used to detect communities based on a clustering algorithm. Besides the name of the clustering algorithm and its specific parameters (see section: Clustering algorithms based on similarity), it receives a dataset (we have tested with a synthetic dataset with users and their emotions to artworks) and a configuration of the similarity functions; and generates the list of clusters (sets of users) detected.

```
community_detection = SimilarityCommunityDetection(users_emotions_df)
result = community_detection.calculate_communities(metric='euclidean', n_clusters=5)
```

During testing we need to experiment with different configuration of algorithms and different similarity functions and weights. This has been done using simple interfaces (Figure 9).

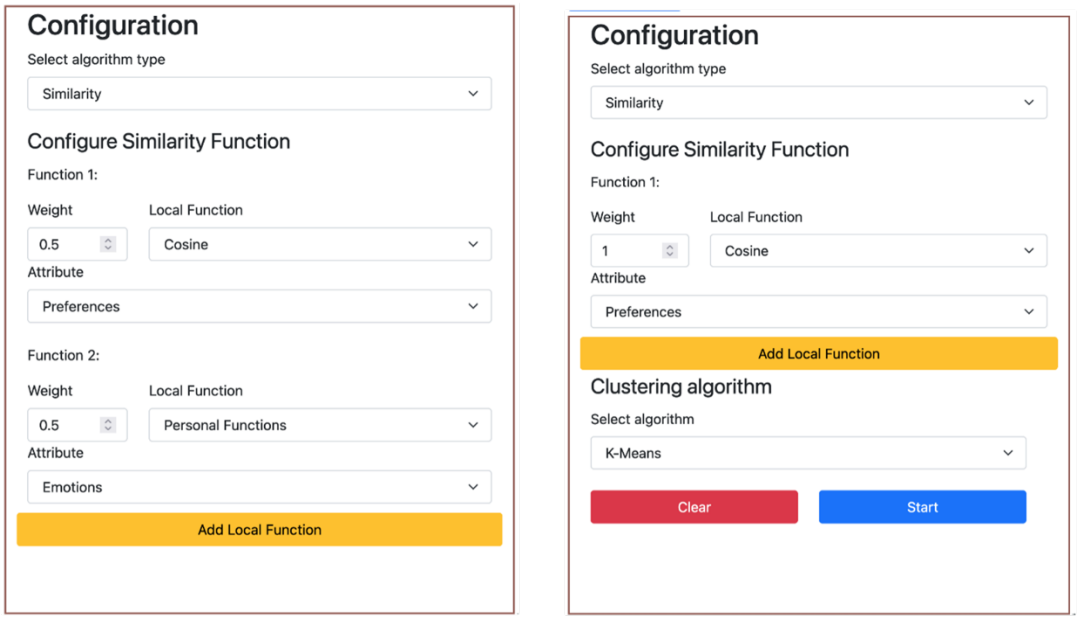

Figure 9: A simple interface for algorithm and similarity configuration

We have experimented using the following similarity functions to compare users:

- **Emotion similarity:** compare the similarity between two users using the emotions felt when interacting with the same picture.
- **Euclidean similarity:** applied in numeric lists, for example, it is used to compare attribute representing the number of pictures seen by users.
- **Cosine similarity:** applied in a numeric list, for example, a vector that contains 1 or 0 depending on if users felt a positive or negative emotion in each artwork.

To compare artworks, we tested:

- **Colour similarity**: This measure uses the weighted euclidean distance between the dominant colour of each painting in HSV space. The dominant colour is the center of the biggest cluster when applying k-means on the artwork image pixels in RGB space.
- **Wikidata content similarity:** This measure employs the knowledge about the elements depicted in an artwork stored in Wikidata. For each artwork, we created a list of contents collecting the values for the "depict" property in Wikidata. The first test over these lists highlighted those common contents were not frequent so we enlarge the list of contents using the concept hierarchy defined in Wikidata with the properties "instance of" and "subclass". The final list is computed traversing this hierarchy up to 2 levels. Finally, the similarity measure is computed using Jaccard over the list of contents.
- **Knowledge similarity:** This similarity uses the information about the artist and the art movement that the artworks belong to. This information is extracted from the WikiArt Emotion Dataset.

Community detection based on graph analysis

The second way to detect communities is by applying algorithms based on graph analysis. To do that, we use graphs based on the emotion that users felt for each artwork. Using this information, we created a graph that each node that represents a user and edges relates to users who felt a concrete emotion in the same artwork. As a result, we created a graph per emotion included in the dataset. Using these graphs, we implemented a class to detect communities. Its functionalities are implemented in `GraphCommunityDetection` class. We have experimented with two graph algorithms to detect implicit communities:

- **Markov Cluster Algorithm**: based on simulation of flows in graphs.
- **Greedy Clustering Algorithm**: comparing the edges inside a cluster and the edges outside the cluster.

Again, we use a simple interface to configure both algorithms (Figure 10).

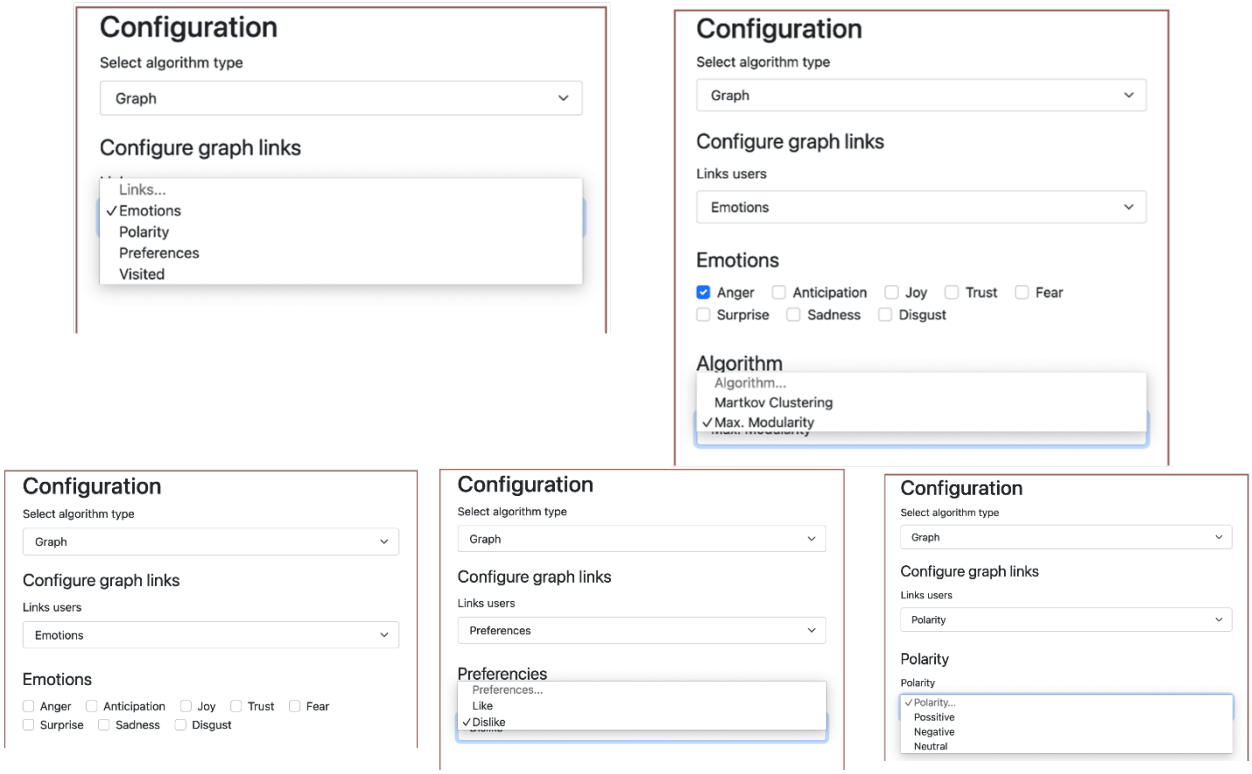

Figure 10: A simple interface for configuring graph algorithms.

The information to configure links could be users' emotions about artworks, users' polarities about artworks, users' preferences about artworks seen, information (yes/no) if users visited or not an artwork. Based on your selection in Step 2, you should select the values used to create the force links between users in the graph.

Visualization

We have tested different options to visualize the communities detected and different ways to explain the common features of each community. A prototype of the visualization tool (Figure 11) is available at https://jljorro.github.io/communities_visualization/

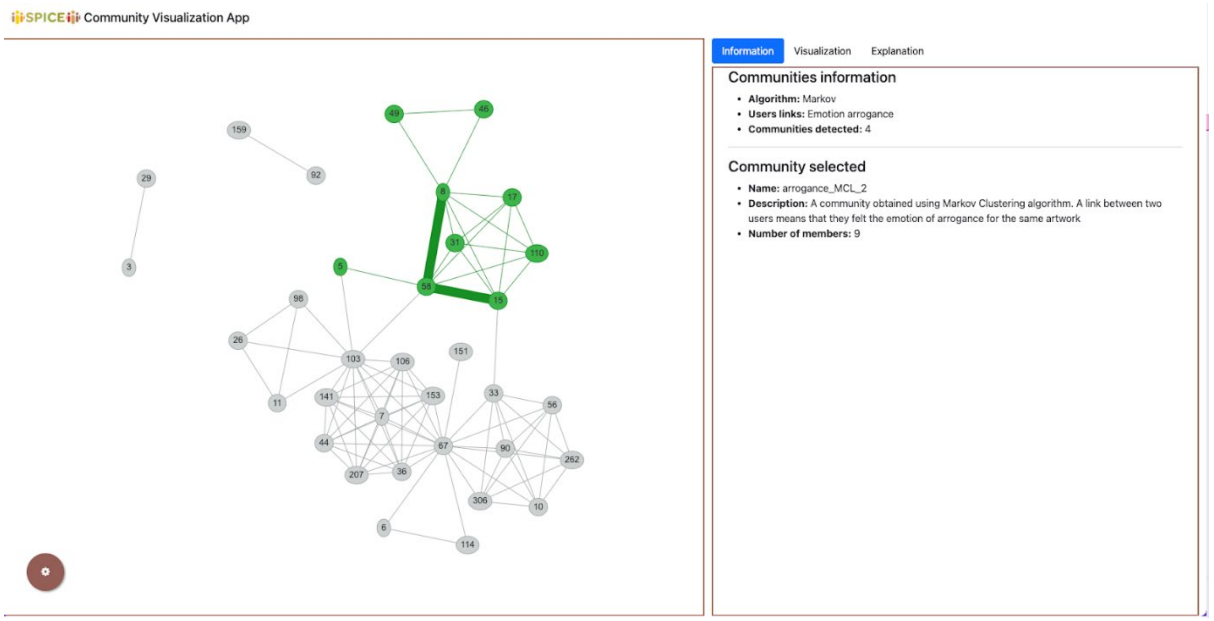

Figure 11: Prototype of a visualization tool (https://jljorro.github.io/communities_visualization/)

MNCN (Madrid)

The experimentation in the Museo Nacional de Ciencias Naturales of Madrid (Spain) involves visits from 61 students from Schools and Secondary Schools of the Madrid Community, namely, Colegio Arcangel (5th and 6th grade), Colegio nuestra señora de val (6th grade – 2 groups), Colegio San Francisco de Asis (2º ESO secondary course), and CEPA Aluche (adult course). These groups are used as the starting point as ***explicit communities***.

Regarding ***personal attributes***, the experiment does not collect individual personal information about the individual students. Instead, personal attributes are inherited from the properties about their schools and grades (explicit communities). The grade represents the course in which the student is enrolled (and so her age). We have information about the school type financing (private or public) and the location zone: zone where the school is located (Madrid city or suburbs).

User generated content is collected using a questionnaire where students mark different yes/no options related to museum t*opics*. So, *in the case of MNCN,* ***interaction attributes*** do not refer to specific museum items but to museum abstract *topics*, namely, responsible consumption, pollution and means of transport and invasive species.

From this input information we applied algorithms to extract implicit communities, generate explanations to describe the communities detected and apply a visualization technique where museum responsible and school's manager have been reflecting on the communities detected and the intra-groups and inter-groups relations, comparisons with other schools, and how location or financing affects the topics of interest. Interaction attributes reflect the students' answers to 5 questions about the topics of interest. These attributes are formalized in a binary vector for each user, where the value 1 corresponded that this option was selected by the user and 0 if not. Then, we could compare the answers pattern of all users to detect implicit communities.

In a second approach we experimented with a dataset where we grouped types of student's answers:

- **Responsible consumption:** we classify all answers in 3 options based on actions in which the student would be willing to make more responsible consumption (reduce consumption, change transport, and recycle).
- **Pollution**: classification of the current transport based on their pollution (high, medium, low).

- **Type of transport:** these options could be private or public transport.

These datasets are included in the GitHub repository, folder data/MNCN.

Community detection

Community detection process uses the information about users (personal, interaction and explicit communities) to get a list of implicit communities. These communities are visualized and explained using the common properties between its members.

Community detection uses `ExplainedCommunitiesDetection` a variation of Agglomerative Clustering with cosine similarity on the user vectors where instead of k we detect communities complying with significant percentages. That means that we do not specify the number of communities to detect in this population, but the algorithm detects a community when it detects a significant (75%) percentage of users with a common property. For example, 75% of the *community A* members refer to use private transport. We incrementally increase the k value (the number of communities to detect) until all communities detected can be explained. This implementation is included in the GitHub repository and Examples 4 and 5 use it to search the communities of MNCN (https://github.com/spice-h2020/prototype-clustering/tree/main/examples/MNCN).

The process generates explanations of a community identified by the number of clusters. For example, using the first dataset of answers, and configuring the percentage of common answers of users in 94%, we obtained 10 communities. Figure 12 shows an example of an explanation based on attributes for implicit community 6.

```
---------------------
COMMUNITY - 6
        - N. Members: 3
        - Properties:
                - Reduce shower time
                - Walking to more places
                - Reduce my waste generation
                - I would be willing to change means of transport
```

Figure 12: Example of explanations based on attributes.

The explanation showed us the answer that we selected by 94% of users of the community (or more)

Visualization

Explanation and visualization techniques allow museum curators and school managers to analyse and compare the implicit and explicit communities and do reflection on the topics of interest. We use an external tool (Gephi) to visualize the community graphs. Gephi[1] allows to apply different layouts to visualize graphs and to apply some methods to filter the information, for example, filter edges by some value or nodes by some properties.

We visualize a graph where each node represents a student and an edge between two nodes represents a *similarity* link between both students. As all users are interconnected, we chose a clearer visualization (Figure 13) where edges are not visible. Instead, similarity values are used as forces and the more similar students are closer in the graph. In addition, for each node, we include the explicit communities where the student is included, that is the values extracted from their schools, and the number of implicit communities detected by our algorithm. Figure 13 shows the visualization of the students classified in the implicit communities. Each colour (or number) corresponds to a specific *implicit community*. Figure in the right emphasizes students that belong to the *explicit community*: private schools. We can analyse what are the implicit communities detected among private schools' students. For example, many of them belong to community 6 and 5 and very few members belong to communities 1,22, or 11 (see descriptions of community groups).

[1] https://gephi.org

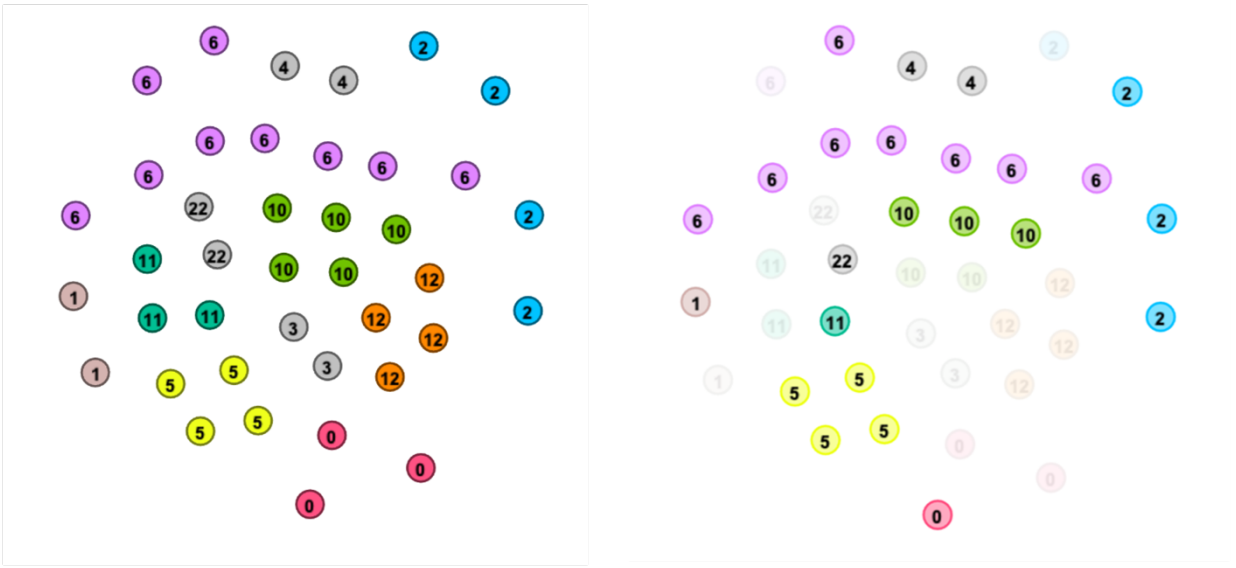

Figure 13: Visualization of MNCN communities using Gephi

This type of visualization allows us to detect if there is a relationship between some implicit communities in respect to explicit communities. Based on these results, we wanted to implement a visualization tool that can compare both types of communities and include different interactions elements to extract information (for example, it shows common properties of users included in two communities).

As future work we want to analyse the hierarchy over the implicit communities.

IMMA (Dublin)

IMMA has performed the experiences Slow Looking, Deep viewpoints and Viewpoints (see more details at D2.3 and D7.5). In this section we analyze the initial data and the preliminary results although further experiments are planned.

Regarding ***personal attributes***, the experiences do not collect individual personal information about the individual citizens. Instead, personal attributes are inherited from the properties about the explicit communities identified from the target and action groups, namely:

- University staff and Students from DCU MELLIE Programme
- Asylum seekers from Dublin City University MELLIE Programme
- Young people living with long-term illnesses as part of HELIUM's Youth Advisory Committee
- Healthcare workers from St James's Hospital
- Black and mixed-race Irish activists from the group Black and Irish
- Migrants from the New Communities Partnership Network
- Secondary school children living in Dublin from the Fighting Words Programme

Each user is explicitly asserted as a member of a group. Users could inherit the common attributes in the communities. These common attributes need to be described by the use case experts in the museums (see D7.4 and D7.5).

User generated content is collected using a set of questions with textual answers that refer to museum items. In this use case ***interaction attributes*** refer to specific museum items. Regarding interaction attributes, the experiment includes 107 interactions within 11 museum items from the IMMA viewpoints artworks, available at Linked Data Hub (Figure 14).

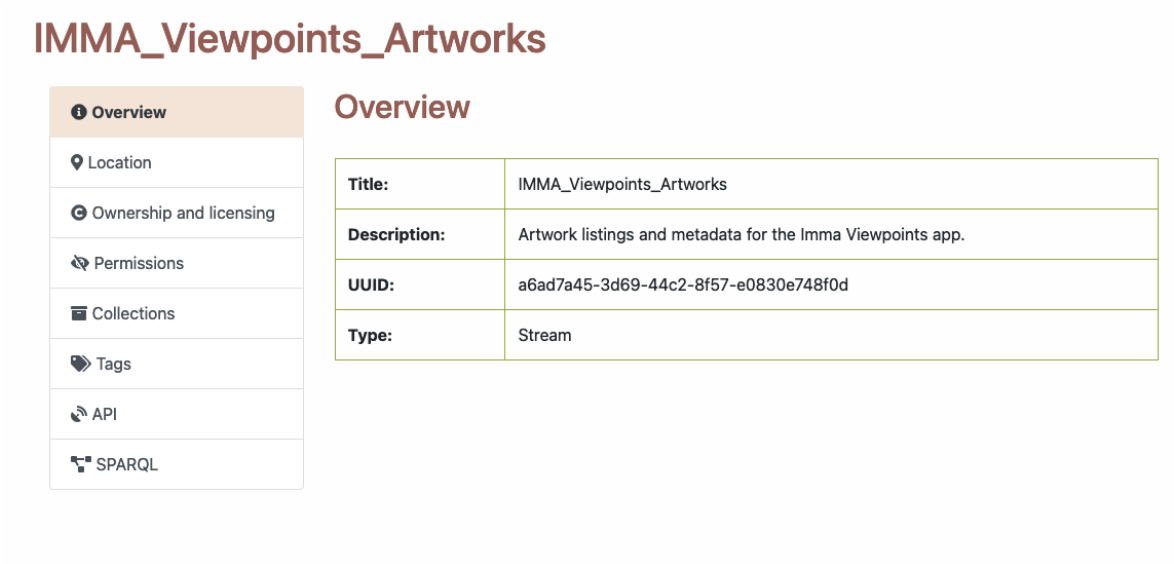

Figure 14: Dataset from IMMA Viewpoints in Linked Data Hub (https://spice.kmi.open.ac.uk/dataset/details/46)

These interactions attributes have been extracted from free text questions. Here we show some examples of questions/answers:

How do you think you would feel if you unexpectedly encountered this object in your house or in your garden? Would it be a welcome guest or an uncomfortable presence? "Uncomfortable presence, an alien object"

"How might you recreate this piece using materials around your own home? Is there anything that you could repurpose to create this piece yourself?" "Autumn leaves"

"What would your best friend think about this artwork?" "That it's a bit mad and fun"

In the internal representation of data (Figure 15), we observe that Emotion Score = null meaning that the sentiment analysis process will be included to complement these textual answers.


```
{
  "_id": "615703c6c500f82baf4adabc",
  "questionID": "60c736d71f567d7ebf323aad",
  "questionAsked": "What would your best friend think about this artwork?",
  "artworkID": "60c0927e0ee8234e932e7a1d",
  "response": "That it's a bit mad and fun",
  "datetimeSubmitted": "2021-10-01T12:49:10.020Z",
  "userResponseID": "P4M8Uto1UwzTd3OKOns3LZ7hIxBheAg1",
  "type": "response",
  "approved": true,
  "emotionScore": null,
  "_datasetid": "123bdcb8-5243-4979-9913-c4e7cc4bfcd8",
  "_timestamp": 1633096984,
  "_timestamp_year": 2021,
  "_timestamp_month": 10,
  "_timestamp_day": 1,
  "_timestamp_hour": 15,
  "_timestamp_minute": 3,
  "_timestamp_second": 4,
  "_updated": true
},
```


Figure 15: Internal representation of the interactions available for IMMA Viewpoints.

From this input information we applied algorithms to extract implicit communities, generate explanations to describe the communities detected and apply a visualization technique. Unfortunately, the initial dataset (Viewpoints) is too small to apply clustering algorithms and extract conclusions about the best configuration for the algorithms or the best suitable visualization techniques. We will extend experiments with additional data from the application scenarios.

We have used `ExplainedCommunitiesDetection` as a variation of *Agglomerative Clustering* where instead of k we detect communities complying with significant percentages. We have configured the clustering algorithm with the cosine similarity. This implementation is included in the GitHub repository [2]. An example of an explanation based on attributes for the obtained communities is shown in Figure 16.

```
--------------------
COMMUNITY - 0
        - N. Members: 2
        - Properties:
                - The Drummer
--------------------
COMMUNITY - 1
        - N. Members: 2
        - Properties:
                - 217 5° Arc x 12
                - SENTINEL VIII
--------------------
COMMUNITY - 2
        - N. Members: 3
        - Properties:
                - 8 Limestones cut to a specific size from rough blocks 150 x 50 x 50cm split into parts and reassembled into their original form
                - Back of Snowman
                - Barrel
                - Recurring Line: North/ South
                - The Drummer
                - Untitled / Corrections D
--------------------
N. USERS WITHOUT COMMUNITY - 4
```

Figure 16: An example of explanation based on attributes for IMMA case study.

HECHT (Haifa)

In the experiments on HECHT museum there is a complete dataset[3] that contains a complete description of demographic **personal attributes** from 60 students:

- Type of School System: GJ: State-Jewish, GA: State-Arab, GJR: State-religious, GH : State-Haredi, MJR : Integrates religious-secular Jew, IH : Haredi-Independent.
- Gender: F: Female, M: Male, O : Other.
- Grade Level: value between 7 and 12.
- Ethnic Group: J: Jew, C : Christian, AC : Arab-Christian, AM : Arab-Muslim, DZ : Druze, BD : Bedouin, CRK : Circassian, DK: Do not know
- Politics group: VL: Very left, L : Left, C : Center, R : Right, VR : Very Right, DK : Do not know
- Level of religiosity: S: secular, M : Traditional, R : religious, VR : Very religious, H : Haredi.

Community detection algorithms identify implicit communities within these personal attributes. Besides, some **explicit communities** of interest are defined using values from these attributes. This experiment is interested in the analysis of politics and religious groups.

Regarding **interaction attributes**, the experiment performs a questionnaire completed by different students that refers to different topics of interests, like BELIEFS, OPENESS, KNOWING HISTORY

Examples of questions that refer to beliefs and openness: (answers in values 1 to 6, where 1 "Strongly disagree" and 6 "Strongly agree"):

- There are beliefs that are so important, and I will never leave them – no matter how good the arguments against them
- A person should constantly examine his beliefs in the light of new information or evidence
- I tend to classify people as for or against me
- I believe that if we allow students to hear controversial opinions – it may only confuse and adversely affect them

[2] https://github.com/spice-h2020/prototype-clustering/tree/main/examples/IMMA

[3] https://github.com/spice-h2020/prototype-clustering/blob/main/data/HECHT/AllData15122021.xlsx

- I think there are many wrong ways, but only one right way – in almost every case
- I consider myself an open and tolerant person towards other people's lifestyles
- I think we should turn to senior clerics for decisions on moral matters

We do not have results until the date of this draft deliverable. We are working to classify the information of dataset and define composed (global) similarity functions to this domain. In this domain the weights that determine the importance of features is an important issue. Weights are experimentally learnt from the interaction with the community model and the museum curators. An interface to experiment with different sets of weights is used to help this process.

GAM

The Galleria D'Arte Moderna (GAM) Turin general target audience is people from the deaf community. In the initial experiment[4] we have 12 items and 128 users with personal data: gender, age, museum, interest. The interactions are based on a questionnaire that includes emojis for representing the feelings that the artwork evoked a user. We have a representation of what kind of emotions are represented by each emoji so as a future work we will work on adapting the emotion similarity measures based on these emojis. We are still in the process of experimentation with this scenario.

DMH

Design museum Helsinki has a brief description of 4 communities described by DMH leaders. These communities are:

- Senior citizens at a senior care centre: Senior citizens living in Kustaankartano (a senior care centre in Helsinki) who meet and interact with each other and use technologies such as the "Digital wall" and Virtual Reality headsets assisted by a mediator(s).
- Senior citizens (free going): Senior citizens living in Helsinki, Kuopio or other towns and can attend workshops and exhibits at the Design Museum or a local library hosting Design Museum's event.
- Rural dweller family: Families composed of parents and children who live in rural areas and have access to a local library without having to travel too far (distance is subjective).
- Asylum seekers: Individuals and families who are claiming asylum in Finland and have not yet received a permit for a long-term stay.

However, until the date of this deliverable we have not got a dataset to detect these communities in this case study and this work is planned as future work.

Conclusions

In this document we have reviewed the community model advancements, first prototypes and results on some of the SPICE use cases. We have reviewed the state of the art of community detection algorithms. We distinguish between clustering and graph analysis algorithms. Note that there is no community detection algorithm that can be universally used for every type of dataset and there is no similarity measure that can be used by every algorithm on every dataset. Parameter settings are crucial in the performance of a clustering algorithm and similarity configuration (user-user, item-item) affects the results.

We have reviewed different approaches to similarity computation and described a catalog of preexisting similarity functions that will help the museum curators to configure and experiment with the communities in each case study. Good data visualizations also will help curators and final users to analyze, validate and explain the clusters generated by the algorithms. We have reviewed the factors for choosing a clustering algorithm and discuss how to evaluate the results.

[4] https://github.com/spice-h2020/prototype-clustering/tree/main/data/GAM

We have experimented with community detection methods with different test domains, and we have described the results. Our goal is to find scrutable models to promote reflection about contents, show differences *within* groups (both explicit and implicit) to tackle preconceptions of homogeneity based on the pre-existing explicit communities. Besides similarity *between* groups allows to tackle preconceptions of heterogeneity. The resulting community model will support the recommender system (see D3.6) that won't be oriented to the typically popular contents or based on providing "more of the same" similar contents to the users (the so called, filter bubble). Instead, community model will support variety and serendipity to the recommendation results.

References

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 1998. Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD Rec.* 27, 2 (June 1998), 94–105. DOI:https://doi.org/10.1145/276305.276314

[2] Yong-Yeol Ahn, James P. Bagrow, and Sune Lehmann. 2010. Link communities reveal multiscale complexity in networks. *Nature* 466, 7307 (August 2010), 761–764. DOI:https://doi.org/10.1038/nature09182

[3] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: ordering points to identify the clustering structure. *ACM SIGMOD Rec.* 28, 2 (June 1999), 49–60. DOI:https://doi.org/10.1145/304181.304187

[4] Nejat Arinik, Vincent Labatut, and Rosa Figueiredo. 2021. Characterizing and Comparing External Measures for the Assessment of Cluster Analysis and Community Detection. *IEEE Access* 9, (2021), 20255–20276. DOI:https://doi.org/10.1109/ACCESS.2021.3054621

[5] Eva Armengol and Enric Plaza. 2001. Similarity Assessment for Relational CBR. In *Case-Based Reasoning Research and Development*, David W. Aha and Ian Watson (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 44–58. DOI:https://doi.org/10.1007/3-540-44593-5_4

[6] Geoffrey H. Ball and David J. Hall. 1967. A clustering technique for summarizing multivariate data. *Behav. Sci.* 12, 2 (March 1967), 153–155. DOI:https://doi.org/10.1002/bs.3830120210

[7] Jeffrey Baumes, Mark K Goldberg, Mukkai S Krishnamoorthy, Malik Magdon-Ismail, and Nathan Preston. 2005. Finding communities by clustering a graph into overlapping subgraphs. *IADIS AC* 5, (2005), 97–104.

[8] Ralph Bergmann. 1998. On the Use of Taxonomies for Representing Case Features and Local Similarity Measures. Retrieved from http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-1071

[9] Ralph Bergmann and Armin Stahl. 1998. Similarity measures for object-oriented case representations. In *Advances in Case-Based Reasoning*, Barry Smyth and Pádraig Cunningham (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 25–36. DOI:https://doi.org/10.1007/BFb0056319

[10] James C. Bezdek, Robert Ehrlich, and William Full. 1984. FCM: The fuzzy c-means clustering algorithm. *Comput. Geosci.* 10, 2–3 (January 1984), 191–203. DOI:https://doi.org/10.1016/0098-3004(84)90020-7

[11] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp.* 2008, 10 (October 2008), P10008. DOI:https://doi.org/10.1088/1742-5468/2008/10/P10008

[12] S. Boccaletti, M. Ivanchenko, V. Latora, A. Pluchino, and A. Rapisarda. 2007. Detecting complex network modularity by dynamical clustering. *Phys. Rev. E* 75, 4 (April 2007), 045102. DOI:https://doi.org/10.1103/PhysRevE.75.045102

[13] Stefan Boettcher and Allon G. Percus. 2001. Extremal optimization for graph partitioning. *Phys. Rev. E* 64, 2 (July 2001), 026114. DOI:https://doi.org/10.1103/PhysRevE.64.026114

[14] D. Brugger, M. Bogdan, and W. Rosenstiel. 2008. Automatic Cluster Detection in Kohonen's SOM. *IEEE Trans. Neural Netw.* 19, 3 (March 2008), 442–459. DOI:https://doi.org/10.1109/TNN.2007.909556

[15] Peter Cheeseman and John Stutz. 1996. Bayesian classification (AutoClass): theory and results. In *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence, USA, 153–180.

[16] Mingming Chen, Konstantin Kuzmin, and Boleslaw K. Szymanski. 2014. Community Detection via Maximization of Modularity and Its Variants. *IEEE Trans. Comput. Soc. Syst.* 1, 1 (March 2014), 46–65. DOI:https://doi.org/10.1109/TCSS.2014.2307458

[17] Chenlong Liu, Jing Liu, and Zhongzhou Jiang. 2014. A Multiobjective Evolutionary Algorithm Based on Similarity for Community Detection From Signed Social Networks. *IEEE Trans. Cybern.* 44, 12 (December 2014), 2274–2287. DOI:https://doi.org/10.1109/TCYB.2014.2305974

[18] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Phys. Rev. E* 70, 6 (December 2004), 066111. DOI:https://doi.org/10.1103/PhysRevE.70.066111

[19] D. Comaniciu and P. Meer. 2002. Mean shift: a robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 5 (May 2002), 603–619. DOI:https://doi.org/10.1109/34.1000236

[20] R.N. Dave and K. Bhaswan. 1992. Adaptive fuzzy c-shells clustering and detection of ellipses. *IEEE Trans. Neural Netw.* 3, 5 (September 1992), 643–662. DOI:https://doi.org/10.1109/72.159055

[21] Belén Díaz-Agudo, Guillermo Jimenez-Diaz, and Jose Luis Jorro-Aragoneses. 2021. User Evaluation to Measure the Perception of Similarity Measures in Artworks. In *Case-Based Reasoning Research and Development*, Antonio A. Sánchez-Ruiz and Michael W. Floyd (eds.). Springer International Publishing, Cham, 48–63. DOI:https://doi.org/10.1007/978-3-030-86957-1_4

[22] Donald S. Blough. 2001. The Perception of Similarity Department of Psychology, Brown University. In *Avian Visual Cognition*. Cognition Press. Retrieved from http://www.pigeon.psy.tufts.edu/avc/toc.htm

[23] Stijn van Dongen and Cei Abreu-Goodger. 2012. Using MCL to Extract Clusters from Networks. In *Bacterial Molecular Networks*, Jacques van Helden, Ariane Toussaint and Denis Thieffry (eds.). Springer New York, New York, NY, 281–295. DOI:https://doi.org/10.1007/978-1-61779-361-5_15

[24] M. Ester, H. P. Kriegel, J. Sander, and Xu Xiaowei. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. (December 1996). Retrieved January 14, 2022 from https://www.osti.gov/biblio/421283-density-based-algorithm-discovering-clusters-large-spatial-databases-noise

[25] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y. Zomaya, Sebti Foufou, and Abdelaziz Bouras. 2014. A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis. *IEEE Trans. Emerg. Top. Comput.* 2, 3 (2014), 267–279. DOI:https://doi.org/10.1109/TETC.2014.2330519

[26] Douglas H. Fisher. 1987. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn.* 2, 2 (September 1987), 139–172. DOI:https://doi.org/10.1007/BF00114265

[27] Santo Fortunato. 2010. Community detection in graphs. *Phys. Rep.* 486, 3–5 (February 2010), 75–174. DOI:https://doi.org/10.1016/j.physrep.2009.11.002

[28] John H. Gennari, Pat Langley, and Doug Fisher. 1989. Models of incremental concept formation. *Artif. Intell.* 40, 1 (September 1989), 11–61. DOI:https://doi.org/10.1016/0004-3702(89)90046-5

[29] Mahsa Ghorbani, Hamid R. Rabiee, and Ali Khodadadi. 2016. Bayesian Overlapping Community Detection in Dynamic Networks. (2016). DOI:https://doi.org/10.48550/ARXIV.1605.02288

[30] Steve Gregory. 2007. An Algorithm to Find Overlapping Community Structure in Networks. In *Knowledge Discovery in Databases: PKDD 2007*, Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras,

Stan Matwin, Dunja Mladenič and Andrzej Skowron (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 91–102. DOI:https://doi.org/10.1007/978-3-540-74976-9_12

[31] Steve Gregory. 2010. Finding overlapping communities in networks by label propagation. *New J. Phys.* 12, 10 (October 2010), 103018. DOI:https://doi.org/10.1088/1367-2630/12/10/103018

[32] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 1998. CURE: an efficient clustering algorithm for large databases. *ACM SIGMOD Rec.* 27, 2 (June 1998), 73–84. DOI:https://doi.org/10.1145/276305.276312

[33] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 2000. Rock: A robust clustering algorithm for categorical attributes. *Inf. Syst.* 25, 5 (July 2000), 345–366. DOI:https://doi.org/10.1016/S0306-4379(00)00022-3

[34] Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. 2015. Semantic Similarity from Natural Language and Ontology Analysis. *Synth. Lect. Hum. Lang. Technol.* 8, 1 (May 2015), 1–254. DOI:https://doi.org/10.2200/S00639ED1V01Y201504HLT027

[35] M. B. Hastings. 2006. Community detection as an inference problem. *Phys. Rev. E* 74, 3 (September 2006), 035102. DOI:https://doi.org/10.1103/PhysRevE.74.035102

[36] Alexander Hinneburg and Daniel A. Keim. 1998. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (KDD'98), AAAI Press, 58–65.

[37] Zhexue Huang. 1998. Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Min. Knowl. Discov.* 2, 3 (September 1998), 283–304. DOI:https://doi.org/10.1023/A:1009769707641

[38] Abdellah Idrissi, Hajar Rehioui, Abdelquoddouss Laghrissi, and Sara Retal. 2015. An improvement of DENCLUE algorithm for the data clustering. In *2015 5th International Conference on Information & Communication Technology and Accessibility (ICTA)*, IEEE, Marrakech, 1–6. DOI:https://doi.org/10.1109/ICTA.2015.7426936

[39] Muhammad Aqib Javed, Muhammad Shahzad Younis, Siddique Latif, Junaid Qadir, and Adeel Baig. 2018. Community detection in networks: A multidisciplinary review. *J. Netw. Comput. Appl.* 108, (April 2018), 87–111. DOI:https://doi.org/10.1016/j.jnca.2018.02.011

[40] Jianbo Shi and J. Malik. 2000. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 8 (August 2000), 888–905. DOI:https://doi.org/10.1109/34.868688

[41] G. Karypis, Eui-Hong Han, and V. Kumar. 1999. Chameleon: hierarchical clustering using dynamic modeling. *Computer* 32, 8 (August 1999), 68–75. DOI:https://doi.org/10.1109/2.781637

[42] Leonard Kaufman and Peter J. Rousseeuw. 2009. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons.

[43] B. W. Kernighan and S. Lin. 1970. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Syst. Tech. J.* 49, 2 (February 1970), 291–307. DOI:https://doi.org/10.1002/j.1538-7305.1970.tb01770.x

[44] Youngdo Kim and Hawoong Jeong. 2011. Map equation for link communities. *Phys. Rev. E* 84, 2 (August 2011), 026110. DOI:https://doi.org/10.1103/PhysRevE.84.026110

[45] Teuvo Kohonen. 2001. *Self-organizing maps* (3rd ed ed.). Springer, Berlin ; New York.

[46] Jussi M. Kumpula, Mikko Kivelä, Kimmo Kaski, and Jari Saramäki. 2008. Sequential algorithm for fast clique percolation. *Phys. Rev. E* 78, 2 (August 2008), 026109. DOI:https://doi.org/10.1103/PhysRevE.78.026109

[47] Jian Liu and Tingzhan Liu. 2010. Detecting community structure in complex networks using simulated annealing with -means algorithms. *Phys. Stat. Mech. Its Appl.* 389, 11 (June 2010), 2300–2309. DOI:https://doi.org/10.1016/j.physa.2010.01.042

[48] James MacQueen and others. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 281–297.

[49] Vivek Mehta, Seema Bawa, and Jasmeet Singh. 2020. Analytical review of clustering techniques and proximity measures. *Artif. Intell. Rev.* 53, 8 (2020), 5995–6023. DOI:https://doi.org/10.1007/s10462-020-09840-7

[50] Marina Meilă and David Heckerman. 2001. An experimental comparison of model-based clustering methods. *Mach. Learn.* 42, 1 (2001), 9–29.

[51] Saif Mohammad and Svetlana Kiritchenko. 2018. Wikiart emotions: An annotated dataset of emotions evoked by art. In *Proceedings of the eleventh international conference on language resources and evaluation (LREC 2018)*.

[52] Tamás Nepusz, Andrea Petróczi, László Négyessy, and Fülöp Bazsó. 2008. Fuzzy communities and the concept of bridgeness in complex networks. *Phys. Rev. E* 77, 1 (January 2008), 016107. DOI:https://doi.org/10.1103/PhysRevE.77.016107

[53] M. E. J. Newman. 2004. Fast algorithm for detecting community structure in networks. *Phys. Rev. E* 69, 6 (June 2004), 066133. DOI:https://doi.org/10.1103/PhysRevE.69.066133

[54] M. E. J. Newman. 2006. Modularity and community structure in networks. *Proc. Natl. Acad. Sci.* 103, 23 (June 2006), 8577–8582. DOI:https://doi.org/10.1073/pnas.0601602103

[55] Andrew Ng, Michael Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Adv. Neural Inf. Process. Syst.* 14, (2001).

[56] R.T. Ng and Jiawei Han. 2002. CLARANS: a method for clustering objects for spatial data mining. *IEEE Trans. Knowl. Data Eng.* 14, 5 (September 2002), 1003–1016. DOI:https://doi.org/10.1109/TKDE.2002.1033770

[57] Santiago Ontañón. 2020. An overview of distance and similarity functions for structured data. *Artif. Intell. Rev.* 53, 7 (October 2020), 5309–5351. DOI:https://doi.org/10.1007/s10462-020-09821-w

[58] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (June 2005), 814–818. DOI:https://doi.org/10.1038/nature03607

[59] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert Syst. Appl.* 36, 2 (March 2009), 3336–3341. DOI:https://doi.org/10.1016/j.eswa.2008.01.039

[60] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. 1990. Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM J. Matrix Anal. Appl.* 11, 3 (July 1990), 430–452. DOI:https://doi.org/10.1137/0611030

[61] Chris Fraley Adrian E Raftery. 1999. MCLUST: software for model-based cluster analysis. *J. Classif.* 16, (1999), 297–306.

[62] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* 76, 3 (September 2007), 036106. DOI:https://doi.org/10.1103/PhysRevE.76.036106

[63] Matthew J. Rattigan, Marc Maier, and David Jensen. 2007. Graph clustering with network structure indices. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, ACM Press, Corvalis, Oregon, 783–790. DOI:https://doi.org/10.1145/1273496.1273595

[64] J. Reichardt and D. R. White. 2007. Role models for complex networks. *Eur. Phys. J. B* 60, 2 (November 2007), 217–224. DOI:https://doi.org/10.1140/epjb/e2007-00340-y

[65] Jörg Reichardt and Stefan Bornholdt. 2006. Statistical mechanics of community detection. *Phys. Rev. E* 74, 1 (July 2006), 016110. DOI:https://doi.org/10.1103/PhysRevE.74.016110

[66] Erich Schubert and Peter J. Rousseeuw. 2019. Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. In *Similarity Search and Applications*, Giuseppe Amato, Claudio Gennaro, Vincent Oria and Miloš Radovanović (eds.). Springer International Publishing, Cham, 171–187. DOI:https://doi.org/10.1007/978-3-030-32047-8_16

[67] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. 1998. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of the 24th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., 428–439.

[68] Huawei Shen, Xueqi Cheng, Kai Cai, and Mao-Bin Hu. 2009. Detect overlapping and hierarchical community structure in networks. *Phys. Stat. Mech. Its Appl.* 388, 8 (April 2009), 1706–1712. DOI:https://doi.org/10.1016/j.physa.2008.12.021

[69] Amit Singhal and I. Google. 2001. Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.* 24, (January 2001).

[70] Amos Tversky. 1977. Features of similarity. *Psychol. Rev.* 84, 4 (1977), 327–352. DOI:https://doi.org/10.1037/0033-295X.84.4.327

[71] Sandro Vega-Pons and José Ruiz-Shulcloper. 2011. A survey of clustering ensemble algorithms. *Int. J. Pattern Recognit. Artif. Intell.* 25, 03 (2011), 337–372.

[72] Wei Wang, Jiong Yang, Richard Muntz, and others. 1997. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., 186–195.

[73] Xiaohua Wang, Licheng Jiao, and Jianshe Wu. 2009. Adjusting from disjoint to overlapping community detection of complex networks. *Phys. Stat. Mech. Its Appl.* 388, 24 (December 2009), 5045–5056. DOI:https://doi.org/10.1016/j.physa.2009.08.032

[74] Jierui Xie, Stephen Kelley, and Boleslaw K. Szymanski. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.* 45, 4 (August 2013), 1–35. DOI:https://doi.org/10.1145/2501654.2501657

[75] Jierui Xie, Boleslaw K. Szymanski, and Xiaoming Liu. 2011. SLPA: Uncovering Overlapping Communities in Social Networks via a Speaker-Listener Interaction Dynamic Process. In *2011 IEEE 11th International Conference on Data Mining Workshops*, IEEE, Vancouver, BC, Canada, 344–349. DOI:https://doi.org/10.1109/ICDMW.2011.154

[76] Dongkuan Xu and Yingjie Tian. 2015. A Comprehensive Survey of Clustering Algorithms. *Ann. Data Sci.* 2, 2 (2015), 165–193. DOI:https://doi.org/10.1007/s40745-015-0040-1

[77] R. Xu and D. WunschII. 2005. Survey of Clustering Algorithms. *IEEE Trans. Neural Netw.* 16, 3 (2005), 645–678. DOI:https://doi.org/10.1109/TNN.2005.845141

[78] R.R. Yager and D.P. Filev. 1994. Approximate clustering via the mountain method. *IEEE Trans. Syst. Man Cybern.* 24, 8 (August 1994), 1279–1284. DOI:https://doi.org/10.1109/21.299710

[79] M.-S. Yang. 1993. A survey of fuzzy clustering. *Math. Comput. Model.* 18, 11 (December 1993), 1–16. DOI:https://doi.org/10.1016/0895-7177(93)90202-A

[80] Tianbao Yang, Yun Chi, Shenghuo Zhu, Yihong Gong, and Rong Jin. 2011. Detecting communities and their evolutions in dynamic social networks—a Bayesian approach. *Mach. Learn.* 82, 2 (February 2011), 157–189. DOI:https://doi.org/10.1007/s10994-010-5214-7

[81] Shihua Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. 2007. Identification of overlapping community structure in complex networks using fuzzy -means clustering. *Phys. Stat. Mech. Its Appl.* 374, 1 (January 2007), 483–490. DOI:https://doi.org/10.1016/j.physa.2006.07.023

[82] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. *ACM SIGMOD Rec.* 25, 2 (1996), 103–114. DOI:https://doi.org/10.1145/235968.233324

[83] Haijun Zhou. 2003. Distance, dissimilarity index, and network community structure. *Phys. Rev. E* 67, 6 (June 2003), 061901. DOI:https://doi.org/10.1103/PhysRevE.67.061901

Annexes

A. Example of community detection code

An example of this type of community detection is implemented in Example 3 in Prototype Clustering Repository (https://github.com/spice-h2020/prototype-clustering).

Community-Module / examples / example3.py in main

<> Edit file | Preview changes

```
from community_module.community_detection.graphsCommunityDetection import GraphCommunityDetection


def main():
    # Step 1: Load emotion relations (two users felt same emotion in a same artwork)
    f = open('../data/prado-dataset/emotions_graphs.json')
    emotion_relations = json.load(f)
    # Step 2: Select an emotion to search communities in the graph
    anger_relations = emotion_relations['anger']
    # Step 3: Convert information in Graph object
    nodes = []
    edges = []
    for row in anger_relations:
        from_id = row[0]
        to_id = row[1]
        extra = row[2]
        if not (from_id in nodes):
            nodes.append(from_id)
        if not (to_id in nodes):
            nodes.append(to_id)
        edges.append((from_id, to_id))
    G = nx.Graph()
    G.add_nodes_from(nodes)
    G.add_edges_from(edges)

    # Step 4: We apply community detection algorithmusing Markov clustering
    community_detection = GraphCommunityDetection(G)
    result = community_detection.calculate_communities(algorithm='Greedy')

    # Step 5: Print communities detected by algorithm
        for user, community in result.items():
        print('User: {}, Community: {}'.format(user, community))

if __name__ == '__main__':
    main()
```

Figure 17: Source code of an example using graph algorithms

main Community-Module / examples / example4.py / <> Jump to Go to file

jljorro Incluidos ejemplos del MNCN Latest commit f1303a8 on 30 Nov 2021 History

1 contributor

34 lines (24 sloc) | 1.08 KB Raw Blame

```
"""
============================================================
Example 4: MNCN 1
============================================================

"""
import pandas as pd

from context import community_module
from community_module.community_detection.similarityCommunityDetection import SimilarityCommunityDetection
from community_module.community_detection.explainedCommunitiesDetection import ExplainedCommunitiesDetection

def main():

    # Load User profiles
    data_df = pd.read_csv('./data/MNCN/user_profiles.csv')

    # Select indexes from profiles
    indexes = list(range(1,16))

    # Filter indexes selected
    data = data_df.iloc[:,indexes]

    # Apply community detection algorithm
    community_detection = ExplainedCommunitiesDetection(data, SimilarityCommunityDetection, 'cosine')
    n_communities, users_communities = community_detection.search_all_communities(answer_binary=True, percentage=1.0)

    for c in range(n_communities):
        community_data = community_detection.get_community(c, answer_binary=True)
        print(community_data)


if __name__ == '__main__':
    main()
```

Figure 18: Source code of an example using ExplainedCommunityDetection algorithm

B. Similarity functions catalog

Similarity function			Types	Implementation
Euclidean	Global/local	Content	numeric vectors	sklearn.metrics.pairwise.euclidean_distances
Manhattan	Global/local	Content	numeric vectors	sklearn.metrics.pairwise.manhattan_distances
Jaccard	Local	Position	sets/multisets	sklearn.metrics.jaccard_score
Cosine	Global/local	Position	vectors	metrics.pairwise.cosine_similarity
Simple Position	Local	Position	instances or concepts in taxonomies	Github
Colour_sim	Local	content	colors	Class DominantColorSimilarity(Similarity) Local github
Emotion_sim	Local	Position	emotions	Local Github
Wikipedia content_sim	Local	Content	artworks	Local Github

Minkowski

Description

Minkowski distance is a metric in a normed vector space which can be considered as a generalisation of both the Euclidean distance and the Manhattan distance. It is named after the German mathematician Hermann Minkowski.

$$\lim_{p\to\infty}\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{\frac{1}{p}}=\max_{i=1}^{n}|x_i-y_i|.$$

Minkowski distance is typically used with p=1 or p=2 which correspond to the Manhattan distance and the Euclidean distance, respectively. We use direct implementations for Euclidean and Manhattan and not the general one.

Euclidean

Description

Euclidean similarity = 1-Euclidean distance.

In mathematics, the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.

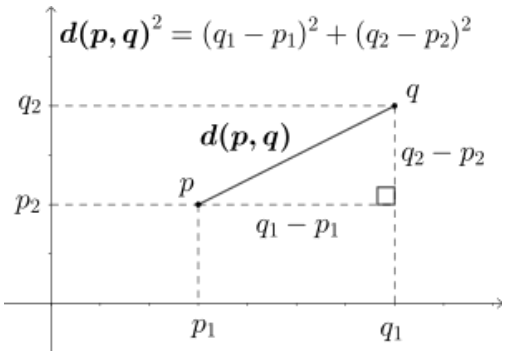

Image from: https://en.wikipedia.org/wiki/Euclidean_distance#/media/File:Euclidean_distance_2d.svg

For points given by Cartesian coordinates in n-dimensional Euclidean space, the distance is

$$d(p,q)=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\cdots+(p_i-q_i)^2+\cdots+(p_n-q_n)^2}.$$

Euclidean similarity is used as a global similarity function to aggregate the results of local similarity values of every attribute of the compared objects. Euclidean similarity is used as a local similarity if the attribute values are numeric vectors (n>=1). Note n=1 refers to simple numeric values.

Implementation:

We have used the implementation from the scikit-learn library

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.pairwise

sklearn.metrics.pairwise.euclidean_distances computes the distance between two points in the euclidean-space n-dimensional. Each point would be represented as a vector array X and Y. In SKLearn, for efficiency reasons, the euclidean distance between a pair of vector x and y is computed as:

dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))

This formulation has two advantages over other ways of computing distances. First, it is computationally efficient when dealing with sparse data. Second, if one argument varies but the other remains unchanged, then dot(x, x) and/or dot(y, y) can be pre-computed.

Example:

It is applied as a local sim measure to compare the number of pictures seen by users (size 1 vector) and it is used as global sim measure to aggregate the local sim values.

```
>>> from sklearn.metrics.pairwise import euclidean_distances
```

```
>>> X = [[0, 1], [1, 1]]
>>> # distance between rows of X
>>> euclidean_distances(X, X)
      array([[0., 1.],  [1., 0.]])
```

Manhattan

Description

The taxicab metric is also known as rectilinear distance, L1 distance, L1 distance, snake distance, city block distance, Manhattan distance or Manhattan length, with corresponding variations in the name of the geometry. The latter names allude to the grid layout of most streets on the island of Manhattan, which causes the shortest path a car could take between two intersections in the borough to have length equal to the intersection distance in taxicab geometry. It measures distance following only axis-aligned directions.

Implementation:

We have used the implementation from the scikit-learn library

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.pairwise
sklearn.metrics.pairwise.manhattan_distances(X, Y=None, *, sum_over_features=True)
Compute the L1 distances between the vectors in X and Y. With sum_over_features equal to False it returns the componentwise distances.

Manhattan similarity = 1-Manhattan distance

Example:

```
sklearn.metrics.pairwise import manhattan_distances
manhattan_distances([[3]], [[3]])
array([[0.]])
manhattan_distances([[3]], [[2]])
array([[1.]])
manhattan_distances([[1, 2], [3, 4]],         [[1, 2], [0, 3]])
array([[0., 2.], [4., 4.]])
```

Jaccard

Jaccard similarity is a statistic used for gauging the similarity and diversity of sample sets. It was developed by Paul Jaccard and independently formulated again by T. Tanimoto. Thus, the Tanimoto index or Tanimoto coefficient are also used in some fields.

It is computed by computing the ratio of Intersection over Union. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Note that by design 0<= J(A,B) <=1. If A and B are both empty, define J(A,B) = 1. Jaccard similarity also applies to bags, i.e., Multisets. This has a similar formula,[4] but the symbols mean bag intersection and bag sum (not union). The maximum value is 1/2.

$$J(A,B) = \frac{|A \cap B|}{|A \uplus B|} = \frac{|A \cap B|}{|A| + |B|}.$$

Implementation

sklearn.metrics.jaccard_score

Example

```
sklearn.metrics.jaccard_score(y_true, y_pred, *, labels=None, pos_label=1,
average='binary', sample_weight=None, zero_division='warn')
```

```
>>> import numpy as np
>>> from sklearn.metrics import jaccard_score
>>> y_true = np.array([[0, 1, 1], [1, 1, 0]])
>>> y_pred = np.array([[1, 1, 1], [1, 0, 0]])
>>> jaccard_score(y_true[0], y_pred[0])
0.6666...
```

Cosine Similarity

cosine_similarity computes the L2-normalized dot product of vectors. That is, if x and y are row vectors,

$$k(x, y) = \frac{xy^\top}{\|x\|\|y\|}$$

their cosine similarity k is defined as:

This is called cosine similarity, because Euclidean (L2) normalization projects the vectors onto the unit sphere, and their dot product is then the cosine of the angle between the points denoted by the vectors. On L2-normalized data, this function is equivalent to linear_kernel. This kernel is a popular choice for computing the similarity of documents represented as tf-idf vectors. cosine_similarity accepts scipy.sparse matrices. (Note that the tf-idf functionality in sklearn.feature_extraction.text can produce normalized vectors, in which case cosine_similarity is equivalent to linear_kernel, only slower.)

Implementation

```
sklearn.metrics.pairwise.cosine_similarity(X, Y=None, dense_output=True)
```

Simple Position similarity

Description

It computes the similarity between two instances (or concepts) in a taxonomy using their position.

SIM (A,B) = depth (LCS (A,B)) / MAX Depth of taxonomy.

LCS stands for Least Common Subsummer between A and B.

Implementation

Color similarity

Description

Calculates the dominant color similarity between two artworks. It uses the weighted euclidean distance between the dominant colour of each painting in HSV space.

References:

Dasari, Haritha & Bhagvati, Chakravarthy & Jain, R.. (2005). Distance measures in RGB and HSV color spaces. 333-338. (PDF) Distance measures in RGB and HSV color spaces
Belén Díaz-Agudo, Guillermo Jiménez-Díaz, Jose Luis Jorro-Aragoneses: User Evaluation to Measure the Perception of Similarity Measures in Artworks. ICCBR 2021: 48-63 User Evaluation to Measure the Perception of Similarity Measures in Artworks | SpringerLink

The dominant colour is the center of the biggest cluster when applying k-means on the artwork image pixels in RGB space.

$$SimAttr_{col}(I_i, I_j) = 1 - dist(hsv_i, hsv_j)$$

$$dist(hsv_i, hsv_j) = \sqrt{(v_i - v_j)^2 + {s_i}^2 + {s_j}^2 + 2s_i s_j(h_i - h_j)}$$

where hsv_i is the dominant colour or artwork I_i in HSV space.

Implementation:

```
Class DominantColorSimilarity(Similarity):
```

https://github.com/spice-h2020/prototype-clustering/blob/main/community_module/similarity/localsimilarity/colorSimilarity.py

```
def computeSimilarity(self, A, B):
        """Method to calculate the dominant color similarity between artwork A and artwork
B.
        Parameters
        ----------
        A : str            The first artwork to calculate the dominant color similarity.
        B : str            The second artwork to calculate the dominant color similarity.
        Returns
        double
            Value of the dominant color similarity between artwork A and artwork B.
        """
        a = self._dominantColor(A)
        b = self._dominantColor(B)
        dh = min(abs(a[0]-b[0]), 360-abs(a[0]-b[0])) / 180.0
        ds = abs(a[1] - b[1])
        dv = abs(a[2] - b[2]) / 255.
        minS = min(a[1],b[1])
        #distance = math.sqrt(dh * dh + ds * ds + dv * dv) # Euclidean
        distance = math.sqrt(dv * dv + a[1]*a[1] + b[1]*b[1] - 2*a[1]*b[1]*dh) # Weighted
euclidean
        #distance = math.sqrt(minS*minS*dh*dh + ds*ds + dv*dv) # Geodesic distance in HSV
        #distance = minS*dh + ds + dv # Weighted L1
        return 1. - distance
```

EmotionSimilarity

https://github.com/spice-h2020/prototype-clustering/blob/main/community_module/similarity/emotionSimilarity.py

This similarity uses the annotations in Wikiart Emotion Dataset about the emotions evoked by the artworks in different users. It is computed using the 3 most popular emotions and calculating the distance between emotions according to the Plutchik wheel of emotions (Figure below) –that places similar emotions close together and opposites 180 degrees apart, like complementary.

$$SimAttr_{emo}(I_i, I_j) = 1 - \frac{1}{3}\sum_{k=1}^{3} dist(e_{ik}, e_{jk})$$

$$dist(e_i, e_j) = min_{dist}(e_i, e_j)/4$$

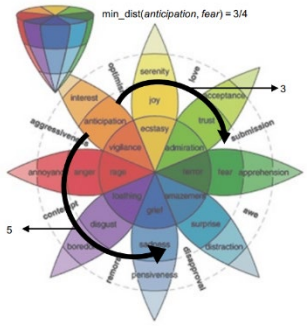

Fig. 1. Emotion wheel conceptualised by Plutchik [19]

where e_{ik} is the k-th most popular emotion in artwork a_i

Wikidata content similarity

Description

This measure employs the knowledge about the elements depicted in an artwork stored in Wikidata. For each artwork, we created a list of contents collecting the values for the "depict" property in Wikidata. A first test over these lists highlighted those common contents were not frequent so we enlarged the list of contents using the concept hierarchy defined in Wikidata with the properties "instance of" and "subclass". The final list is computed traversing these hierarchies up to 2 levels. Finally, the similarity measure is computed using Jaccard over the list of contents.

$$SimAttr_{con}(I_i, I_j) = Jaccard(C_i, C_j) = \frac{C_i \cap C_j}{C_i \cup C_j}$$

where C_i is the list of contents in artwork a_i .

Taxonomy similarity

Description

Similarity is computed for a given pair of objects where an individual can represent either a user itself or, in general, the value of an attribute. Any individual is defined in terms of the concepts of which that individual is an instance and the attributes asserted for it (if any). In graph representations these attributes can be relations connecting the individual to other individuals or primitive values (like scalars, numbers, or symbols). This similarity function is computed using the following equation:

$$SIM\ (K_3, K_4)\frac{depth < K_3, K_4 >}{MAX(depth)}$$

Implementation

https://github.com/spice-h2020/prototype-clustering/blob/main/community_module/similarity/taxonomySimilarity.py