Graph clustering

Anvar Kurmukov

2018

Использование открытых данных

«ВКонтакте» через суд запретила использовать данные пользователей соцсети для оценки в банках

Сервис по оценке кредитоспособности Double Data проиграл компании в суде.

47 комментариев В закладки Поделиться

Соцсеть «ВКонтакте» через суд запретила сервису Double Data использовать данные из открытых профилей пользователей соцсети в коммерческих целях. Об этом vc.ru рассказали в компании.

«ВКонтакте» подала иск против сервиса в январе 2017 года. По данным соцсети, Double Data собрал и проанализировал данные 407 млн профилей, чтобы оценить кредитоспособность пользователей, а затем предлагал эту информацию банкам.

STORIES

Dirty public secrets Yandex leaks internal Google Docs apparently shared by Russian banks, state officials, and Internet trolls

Meduza 21:55, 5 july 2018

Fitness tracking app Strava gives away location of secret US army bases

Data about exercise routes shared online by soldiers can be used to pinpoint overseas facilities

- **Latest: Strava suggests military users 'opt out' of heatmap as row deepens**

Взлом персональных страниц

Описание методов API

Ниже приводятся все методы для работы с данными **ВКонтакте**.

Account

account.ban	Добавляет пользователя или группу в черный список.
account.changePassword	Позволяет сменить пароль пользователя после успешного восстановления доступа к аккаунту через СМС, используя метод auth.restore.
account.getActiveOffers	Возвращает список активных рекламных предложений (офферов), выполнив которые пользователь сможет получить соответствующее количество голосов на свой счёт внутри приложения.
account.getAppPermissions	Получает настройки текущего пользователя в данном приложении.
account.getBanned	Возвращает список пользователей, находящихся в черном списке.
account.getCounters	Возвращает ненулевые значения счетчиков пользователя.
account.getInfo	Возвращает информацию о текущем аккаунте.
account.getProfileInfo	Возвращает информацию о текущем профиле.
account.getPushSettings	Позволяет получать настройки Push-уведомлений.
account.registerDevice	Подписывает устройство на базе iOS, Android, Windows Phone или Mac на получение Push-уведомлений.
account.saveProfileInfo	Редактирует информацию текущего профиля.
account.setInfo	Позволяет редактировать информацию о текущем аккаунте.
account.setNameInMenu	Устанавливает короткое название приложения (до 17 символов), которое выводится пользователю в левом меню.
account.setOffline	Помечает текущего пользователя как offline (только в текущем приложении).
account.setOnline	Помечает текущего пользователя как online на 5 минут.
account.setPushSettings	Изменяет настройку Push-уведомлений.

Messages

messages.addChatUser	Добавляет в мультидиалог нового пользователя.
messages.allowMessagesFromGroup	Позволяет разрешить отправку сообщений от сообщества текущему пользователю.
messages.createChat	Создаёт беседу с несколькими участниками.
messages.delete	Удаляет сообщение.
messages.deleteChatPhoto	Позволяет удалить фотографию мультидиалога.
messages.deleteConversation	Удаляет личные сообщения в беседе.
messages.denyMessagesFromGroup	Позволяет запретить отправку сообщений от сообщества текущему пользователю.
messages.edit	Редактирует сообщение.
messages.editChat	Изменяет название беседы.
messages.getByConversationMessageId	Возвращает сообщения по их идентификаторам в рамках беседы.
messages.getById	Возвращает сообщения по их идентификаторам.
messages.getChat	Возвращает информацию о беседе.
messages.getChatPreview	Получает данные для превью чата с приглашением по ссылке.
messages.getConversationMembers	Позволяет получить список участников беседы.
messages.getConversations	Возвращает список бесед пользователя.
messages.getConversationsById	Позволяет получить беседу по её идентификатору.
messages.getHistory	Возвращает историю сообщений для указанного диалога.

VK api

What is clustering?

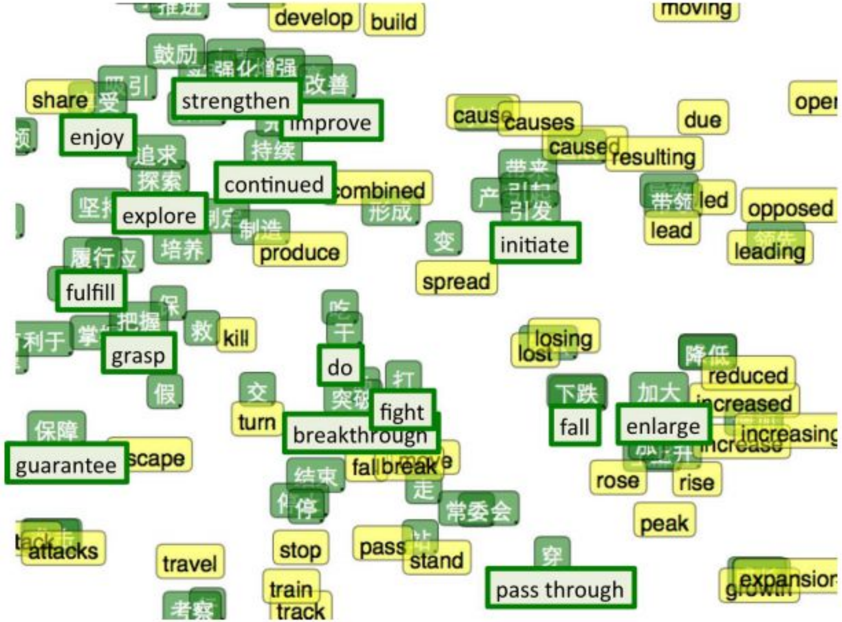
Cluster analysis or clustering is a problem of **dividing** a **set of objects** into groups such that objects from the same group are more **similar** (in some sense) then objects from different groups.

Questions:

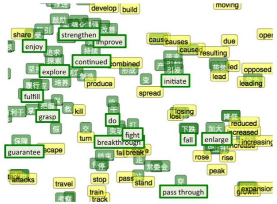
1. Set of objects → How do we represent different types of objects?
2. Similarity → Distance, color, semantics?
3. Dividing → What algorithms do we have?
4. How to evaluate a result?

Mathematical representation

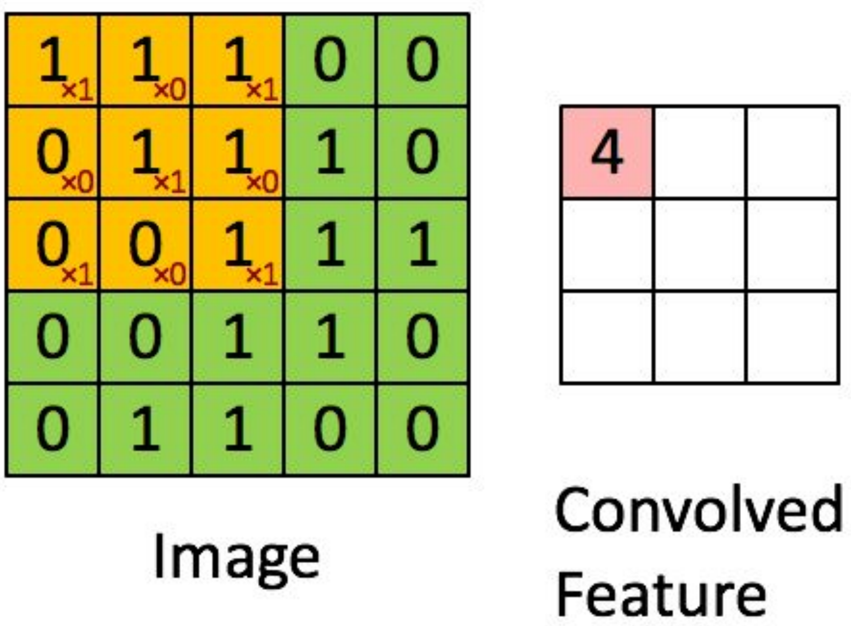
1. Text → Bag of words, word2vec



Mathematical representation

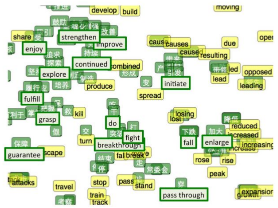


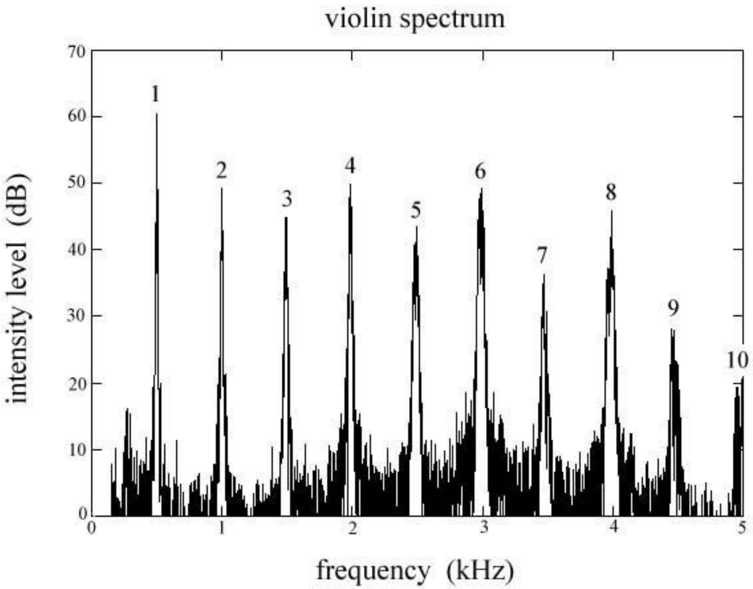
1. Text → Bag of words, word2vec
2. Images → Convolutional filters, pixelwise vectors

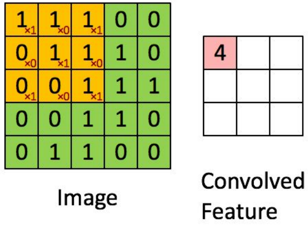


Mathematical representation

1. Text → Bag of words, word2vec
2. Images → Convolutional filters, pixelwise vectors
3. Sound → Spectrum, Fourier transform

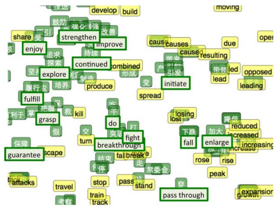


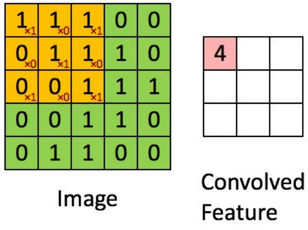




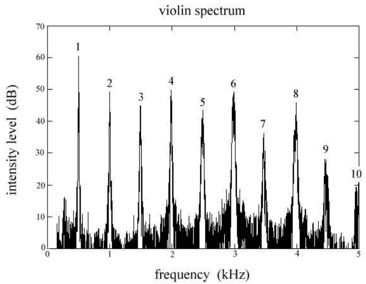
Mathematical representation

1. Text → Bag of words, word2vec
2. Images → Convolutional filters, pixelwise vectors
3. Sound → Spectrum, Fourier transform
4. etc.



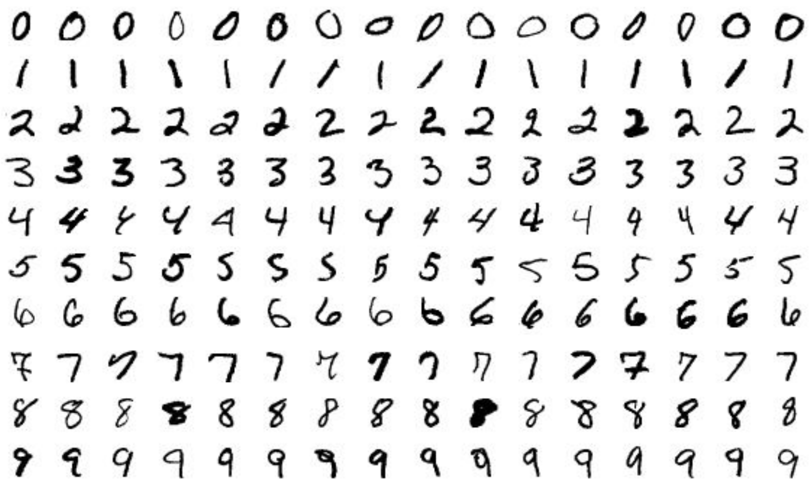


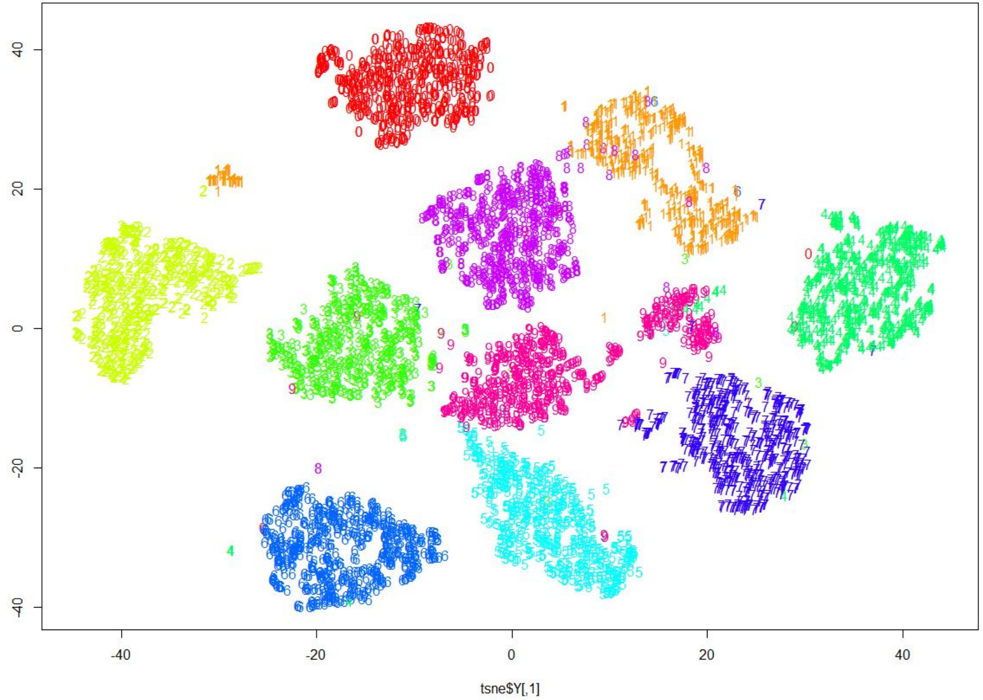
Mathematical representation should be **good**



Similarity

Similar objects should have similar mathematical representations.

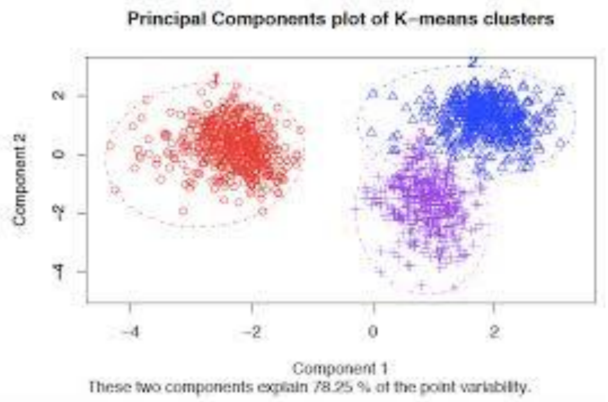


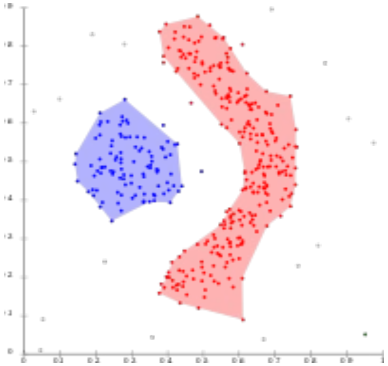


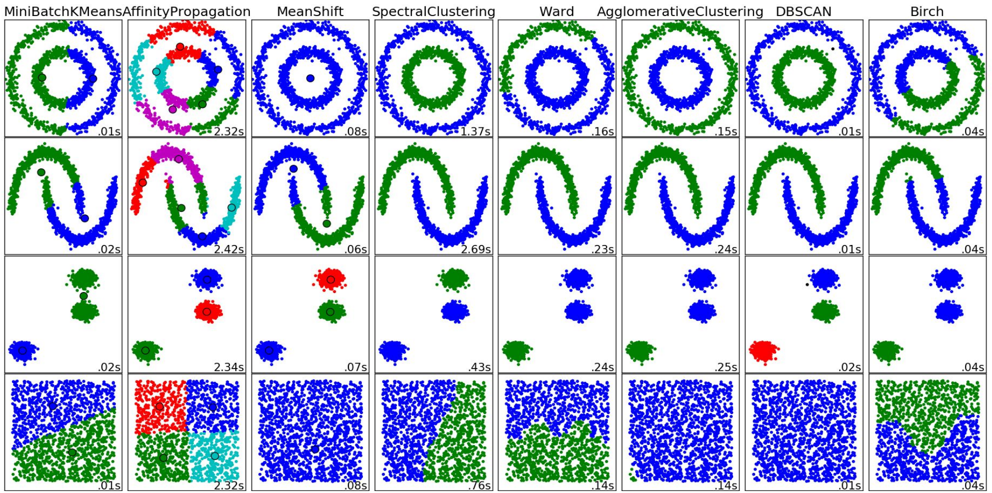
Object similarity $\rightarrow$ Distance between representations

Algorithms

1. Centroid-based methods
2. Hierarchical division
3. Deep Learning
4. Spectral partitioning
5. Density-based models
6. many others

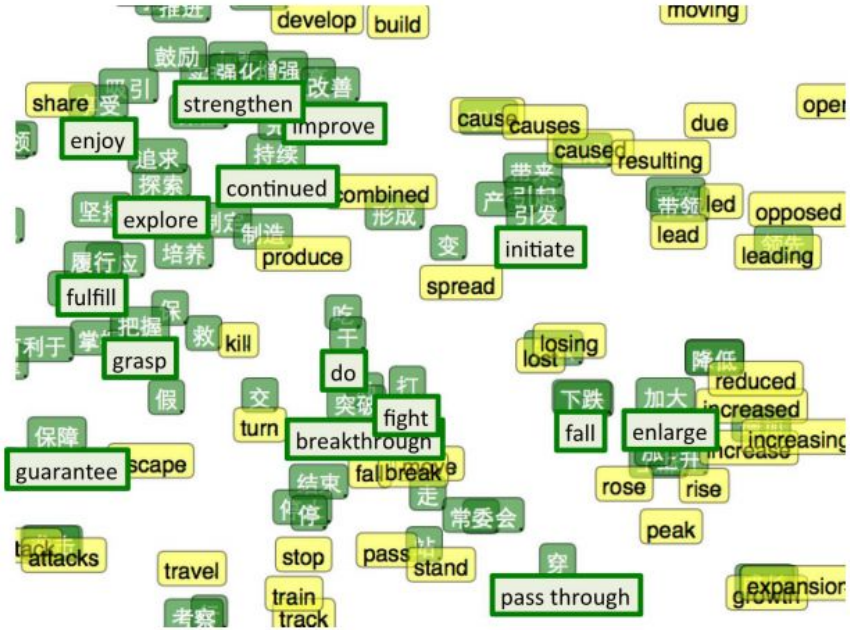






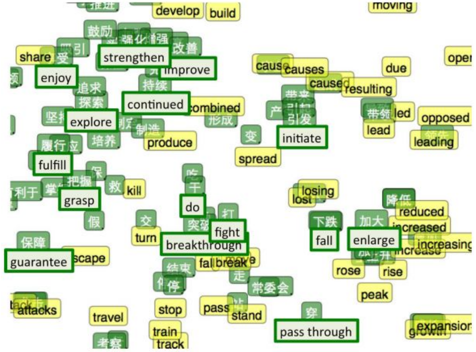
Results evaluation

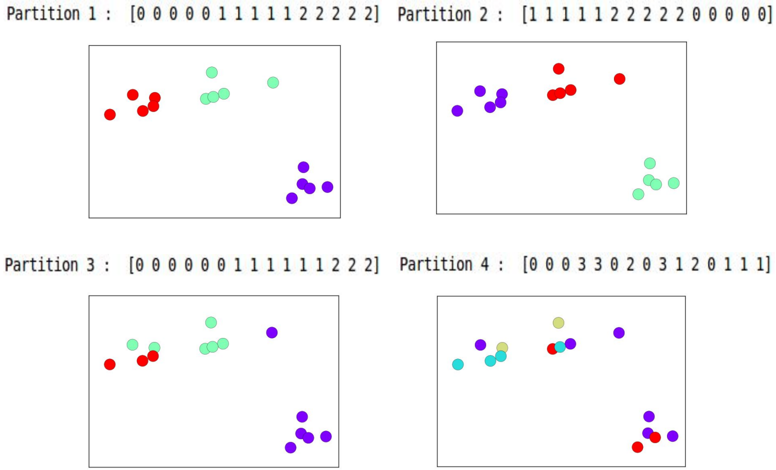
1. Human evaluation (expert opinion)
2. Gravity metrics
3. Using external data (compare with benchmark)



Results evaluation

1. Human evaluation (expert opinion)
2. Gravity metrics
3. Using external data (compare with benchmark)





MI $(P_1, P_1) = 1.0$

MI $(P_1, P_2) = 1.0$

MI $(P_1, P_3) = 0.529$

MI $(P_1, P_4) = 0.049$

Mutual Information is a measure of similarity, thus value 1 indicates completely identical partitions, and values close to 0 stands for very dissimilar.

K-means

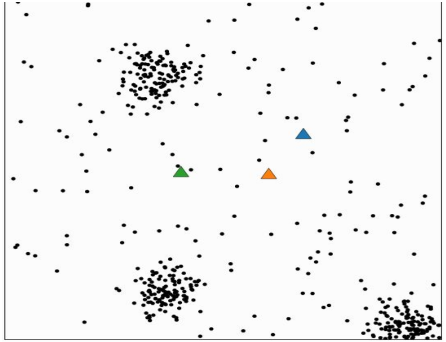
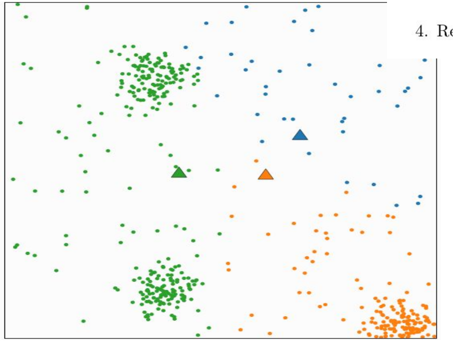
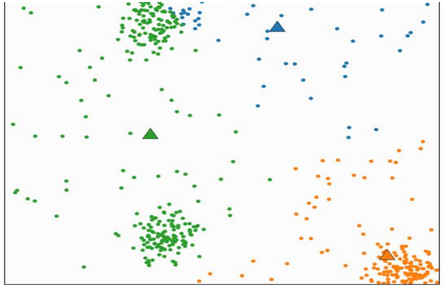
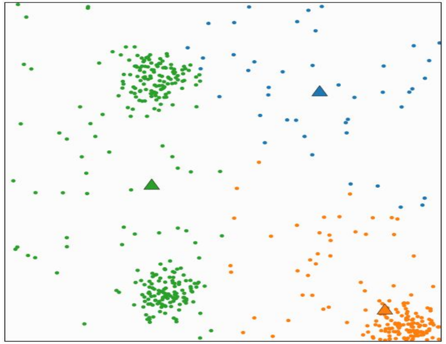
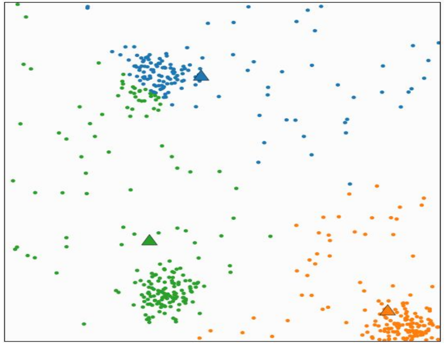
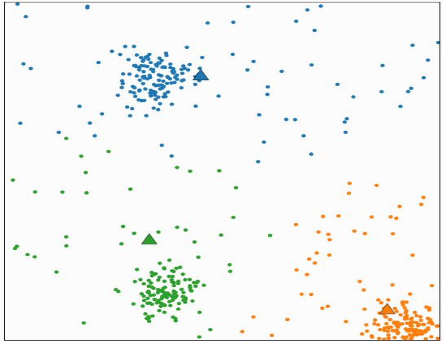
Given set of points $X = \{x_1, x_2, \ldots, x_n\}, X \subseteq \mathbb{R}^d$ and an integer number k:

1. Arbitrarily choose k initial centers $C = \{c_1, c_2, \ldots, c_k\}$.
2. For each $i \in \{1, \ldots, k\}$, set cluster C_i to be the set of points X that are closer to c_i than they are to c_j, for all $j \neq i$. Ties might be broken uniformly at random.
3. For each $i \in \{1, \ldots, k\}$, set c_i to be the center of mass of all points in $C_i : c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$.
4. Repeat Steps 2 and 3 until C no longer changes.

K-means example

Given set of points $X = \{x_1, x_2, \dots, x_n\}, X \subseteq \mathbb{R}^d$ and an integer number k:

1. Arbitrarily choose k initial centers $C = \{c_1, c_2, \dots, c_k\}$.
2. For each $i \in \{1, \dots, k\}$, set cluster C_i to be the set of points X that are closer to c_i than they are to c_j, for all $j \neq i$. Ties might be broken uniformly at random.
3. For each $i \in \{1, \dots, k\}$, set c_i to be the center of mass of all points in C_i : $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$.
4. Repeat Steps 2 and 3 until C no longer changes.

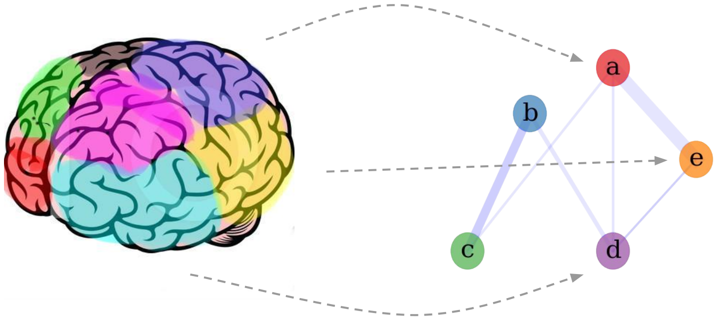







Applications

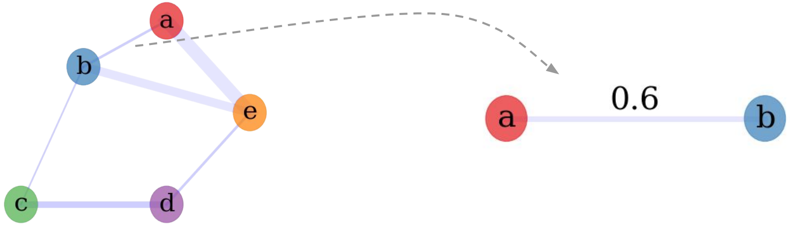
1. Biology (Gene analysis)
2. Medicine (Image segmentation, patients diagnosis)
3. Social network analysis (Community detection)
4. Computer science (Recommender systems, anomaly detection)
5. others

Brain network

Brain regions become nodes



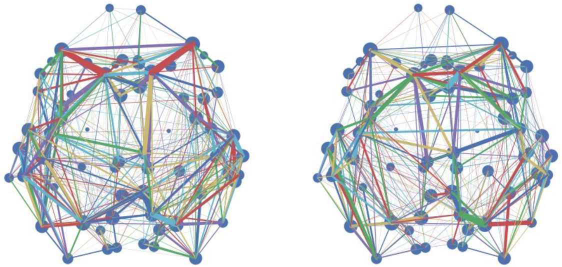
Neural connections between regions become edges



Graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{l}, \boldsymbol{w})$, where

- $\boldsymbol{V}$ is the set of nodes
- $\boldsymbol{E}$ is the set of edges
- $\boldsymbol{l}$ is node's labeling mapping
- $\boldsymbol{w}$ is edge's weighting mapping

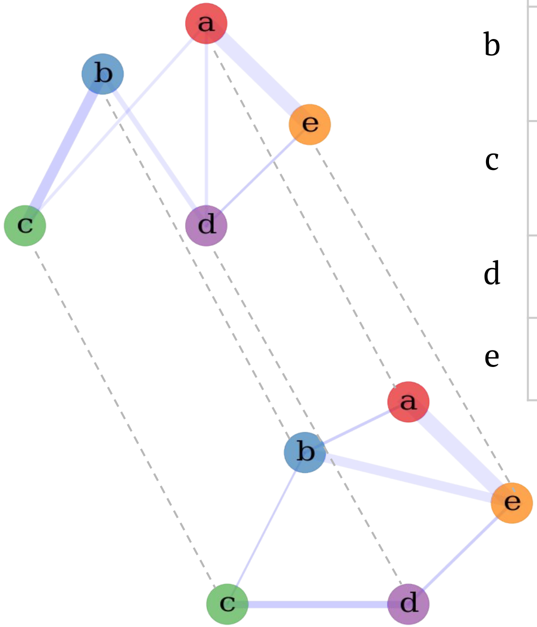
is called a brain network or a **connectome**



Connectomes: properties

- connectomes are relatively **small** graphs, usually with at most few hundreds of nodes
- the graphs are **simple** (no loops), **undirected**, i.e. the adjacency matrices are symmetric
- edges are **weighted**
- graphs are **connected**
- each node is **uniquely labeled** (according to the brain region), and the **set of labels is the same** across connectomes

Thus, graphs are fully described by their adjacency matrices (A)



	a	b	c	d	e
a	0	**6**	**0**	**0**	**30**
b	6	0	**4**	**0**	**20**
c	0	4	0	**15**	**0**
d	0	0	15	0	**6**
e	30	20	0	6	0

Adjacency matrix. Upper triangle colored

Classification task

$$\text{Dataset} = \{(G_1, y_1), (G_2, y_2), \ldots, (G_n, y_n)\}$$

G_i is simple, weighted, undirected graph with unique labels on nodes

y_i is a class label {Type I, Type II}, e.g:

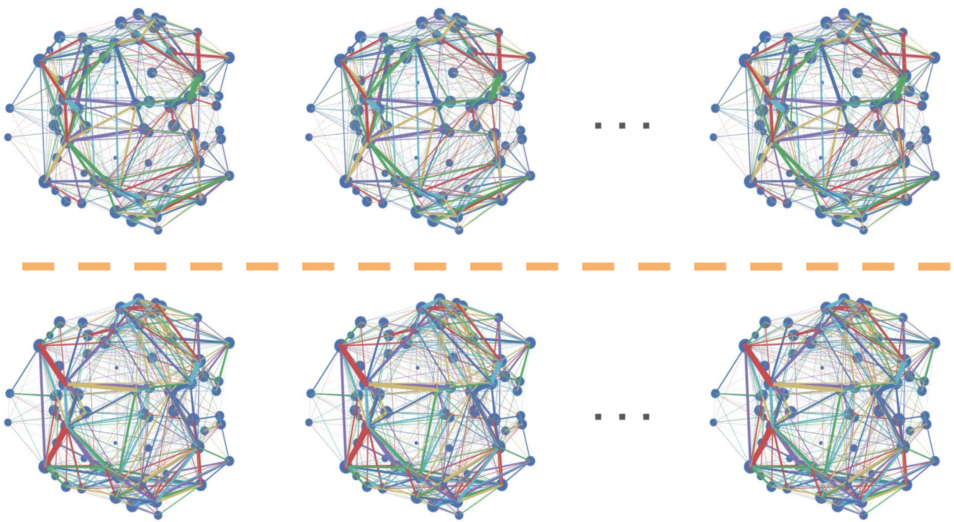
{disease, control},

{male, female},

etc.

Type I

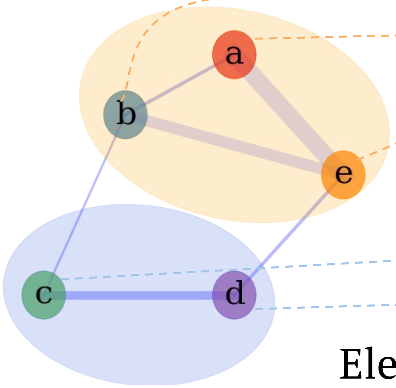
Type II



Do **Type I** graphs substantially differ from **Type II** graphs?

Clustering based kernels

G_1 :



P_1 :

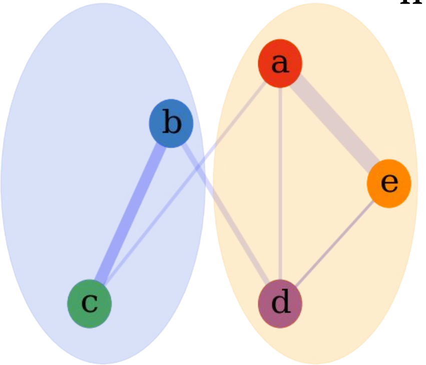
a	b	c	d	e
1	1	0	0	1

Idea: use similarity between graph clusterings as a measure of similarity between graphs

Elements of **belonging vector** P_i are **belonging markers** (or indicators), vertices from the same cluster have same belonging marker

$$sim(G_1, G_2) \sim sim(P_1, P_2)$$

G_2 :

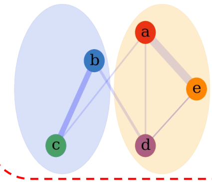


P_2 :

a	b	c	d	e
1	0	0	1	1

Note: in order to measure similarity between graph clusterings we must use appropriate metrics e.g. **Rand Index** or **Mutual Information** or their variants

Non-overlapping partition



P_2 :

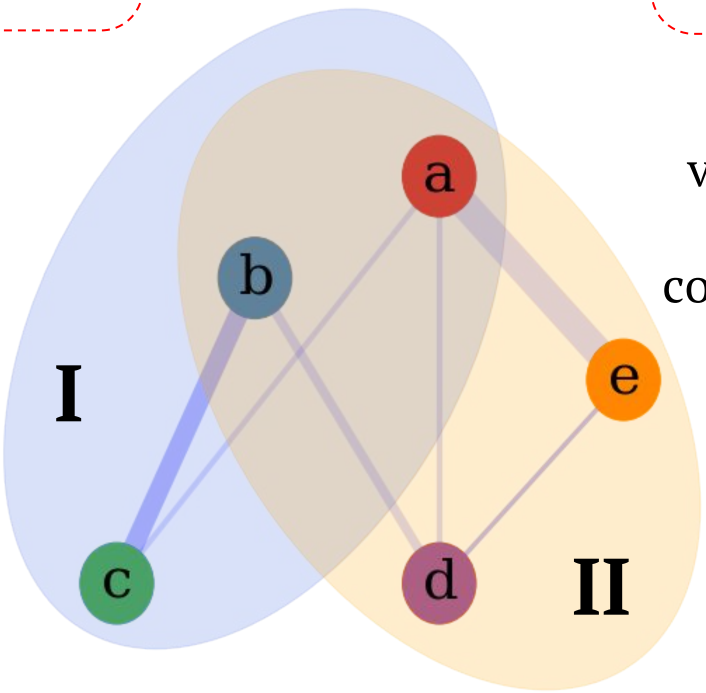
a	b	c	d	e
1	0	0	1	1

Overlapping clustering

Idea: what if instead of non-overlapping partitions we will try overlapping ones?

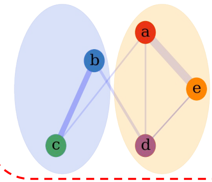
H_2 :

	a	b	c	d	e
I	.7	.2	1	0	0
II	.3	.8	0	1	1



Instead of belonging vector(P_2), we now have a **belonging matrix** (H_2), columns stands for vertices, and rows stands for clusters.

Non-overlapping partition



P_2 :

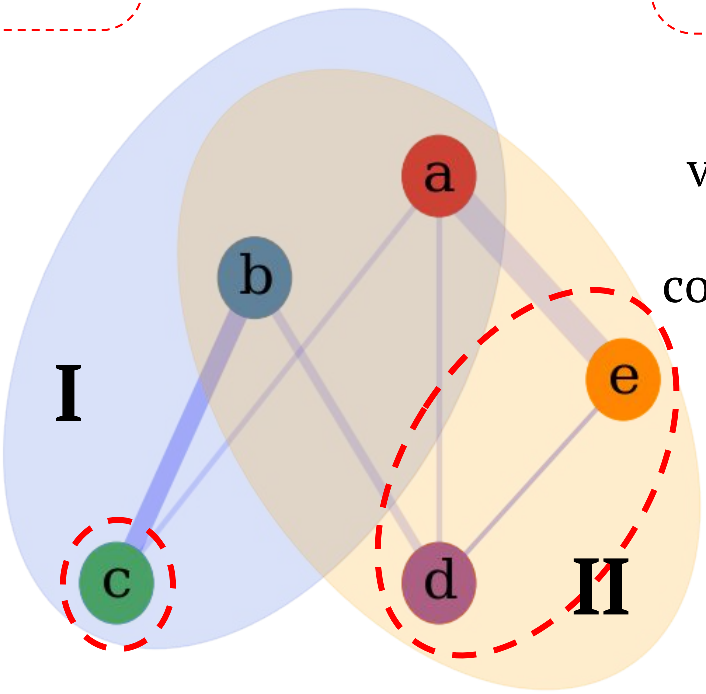
a	b	c	d	e
1	0	0	1	1

Overlapping clustering

Idea: what if instead of non-overlapping partitions we will try overlapping ones?

H_2 :

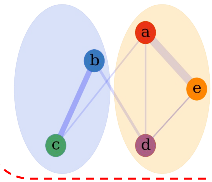
	a	b	c	d	e
I	.3	.8	1	0	0
II	.7	.2	0	1	1



Instead of belonging vector(P_2), we now have a **belonging matrix** (H_2), columns stands for vertices, and rows stands for clusters.

Nodes ***c, d, e,*** belong to one cluster (***c*** to **I** and ***d, e*** to **II**)

Non-overlapping partition



P_2 :

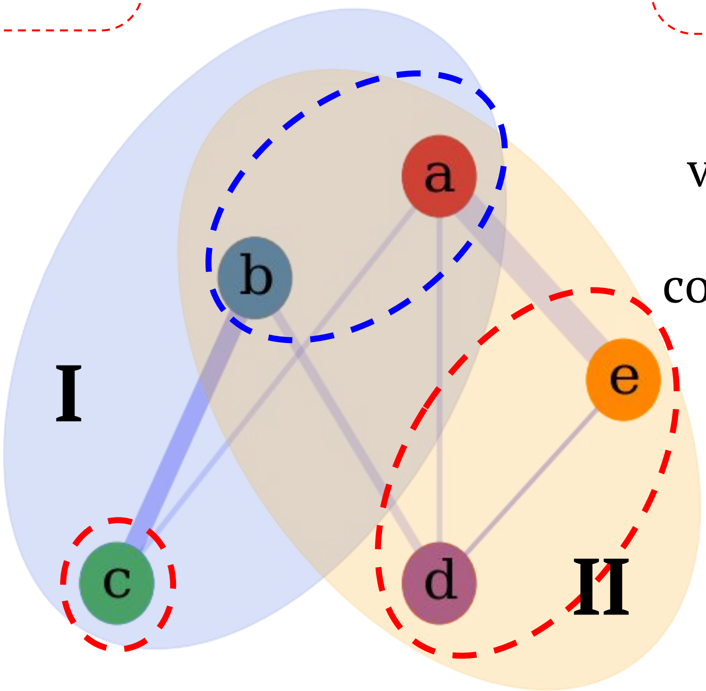
a	b	c	d	e
1	0	0	1	1

Overlapping clustering

Idea: what if instead of non-overlapping partitions we will try overlapping ones?

H_2 :

	a	b	c	d	e
I	.3	.8	1	0	0
II	.7	.2	0	1	1



Instead of belonging vector(P_2), we now have a **belonging matrix** (H_2), columns stands for vertices, and rows stands for clusters.

Nodes ***a*** and ***b*** belong to both clusters with **different strength**

Nodes ***c, d, e,*** belong to one cluster (***c*** to **I** and ***d, e*** to **II**)

Similarity of graph clusterings

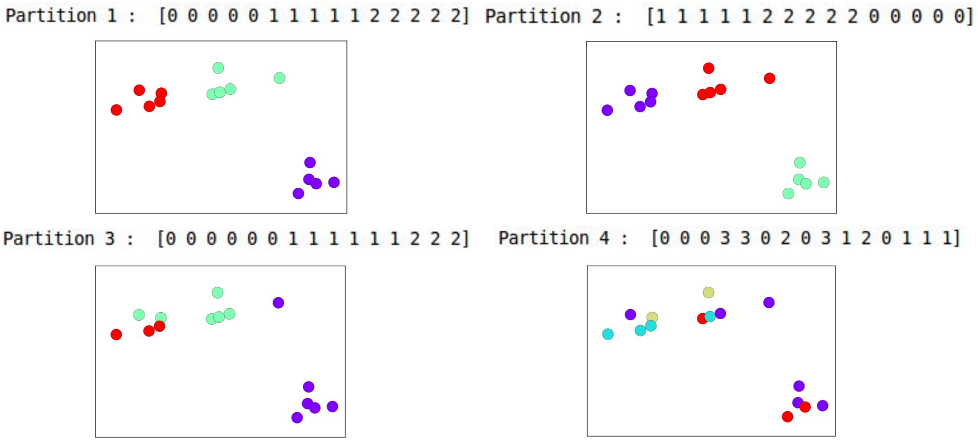
AMI (P_1, P_1) = 1.0

AMI (P_1, P_2) = 1.0

AMI (P_1, P_3) = 0.529

AMI (P_1, P_4) = 0.049

Mutual Information is a measure of similarity, thus value 1 indicates completely identical partitions, and values close to 0 stands for very dissimilar. Which is also true for both AMI and NMI



Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. Journal of Machine Learning Research, pp. 2837–2854 (2010)

In order to compare non-overlapping partitions we used so-called **Adjusted Mutual Information**

$$\omega_{AMI}(G_i, G_j) = 1 - AMI(P_i, P_j)$$

All **soft** overlappings were thresholded down to **hard** overlappings

Normalized Mutual Information were used to compare hard overlappings

$$\omega_{NMI}(G_i, G_j) = 1 - NMI(\overline{H_i}, \overline{H_j})$$

where overlined H's are hard overlappings, produced from soft ones

$$K(G_i, G_j) = e^{-\alpha\omega(G_i, G_j)}$$

Soft and Hard overlapping

SOFT belonging matrix

H_2 :

	a	b	c	d	e
I	.7	.2	1	0	0
II	.3	.8	0	1	1

HARD belonging matrix

$\overline{H_2}$:

Threshold = 0.15

	a	b	c	d	e
I	1	1	1	0	0
II	1	1	0	1	1

Threshold = 0.4

	a	b	c	d	e
I	1	0	1	0	0
II	0	1	0	1	1

Soft and Hard overlapping

SOFT belonging matrix

H_2 :

	a	b	c	d	e
I	.7	.2	1	0	0
II	.3	.8	0	1	1

Strength / weights
(e.g. **a** **belongs** to **I** with strength .7
and to **II** with strength .3)

From strength to
cluster indicators

HARD belonging matrix

$\overline{H_2}$:

Threshold = 0.15

	a	b	c	d	e
I	1	1	1	0	0
II	1	1	0	1	1

Threshold = 0.4

	a	b	c	d	e
I	1	0	1	0	0
II	0	1	0	1	1

Cluster
indicators
(e.g. **a** **lies** in **I**
and **II**)

Soft and Hard overlapping

SOFT belonging matrix

H_2 :

	a	b	c	d	e
I	.7	.2	1	0	0
II	.3	.8	0	1	1

Strength / weights
(e.g. **a** **belongs** to **I** with strength .7 and to **II** with strength .3)

From strength to cluster indicators

HARD belonging matrix

Threshold = 0.15 ⟶ Threshold = 0.4

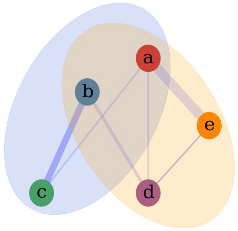
$\overline{H_2}$:

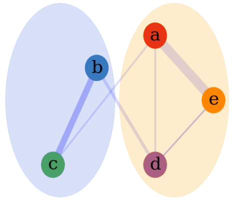
	a	b	c	d	e
I	1	1	1	0	0
II	1	1	0	1	1

	a	b	c	d	e
I	1	0	1	0	0
II	0	1	0	1	1

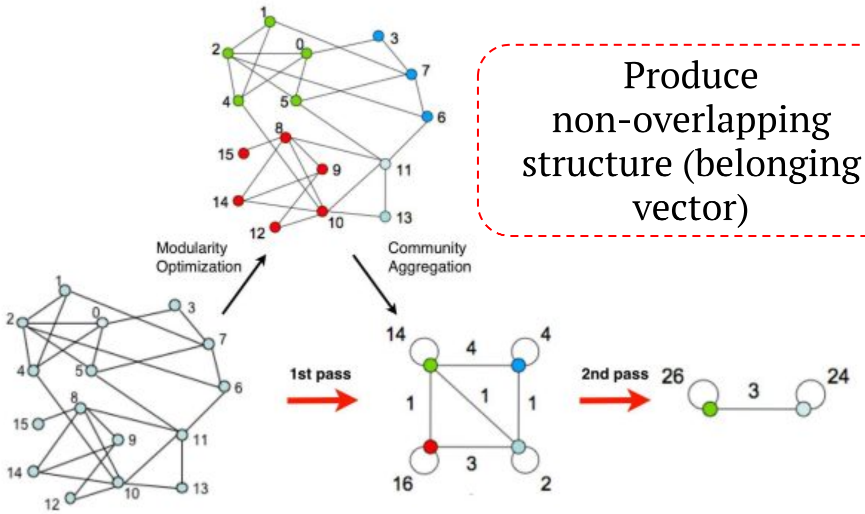
Cluster indicators
(e.g. **a** **lies** in **I** and **II**)

Increasing threshold leads to non-overlapping structure





Louvain Modularity



Produce non-overlapping structure (belonging vector)

Modularity is given by:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(i,j)$$

Blondel, Vincent D., et al. "Fast unfolding of communities in large networks." *Journal of statistical mechanics: theory and experiment* 2008.10 (2008): P10008.

Non-negative Matrix Factorization (NMF)

$$\min_{W,H \geq 0} ||A - WH||_F^2$$

$$A \in \mathbb{R}_+^{n \times n}, W \in \mathbb{R}_+^{n \times k}, H \in \mathbb{R}_+^{k \times n}$$

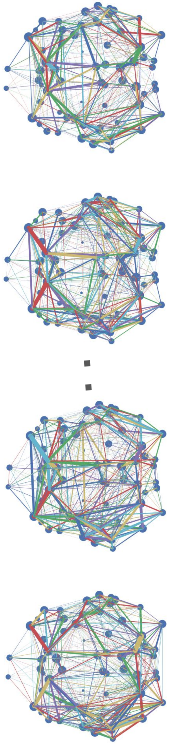
H is interpreted as a **belonging matrix**.

Produce overlapping community structure (belonging matrix)

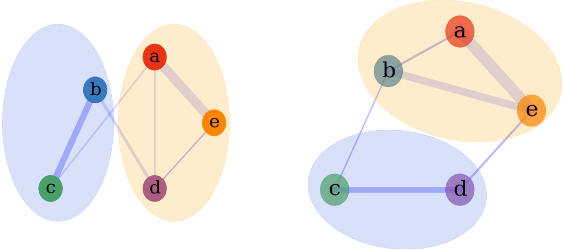
Kuang, Da, Chris Ding, and Haesun Park. "Symmetric nonnegative matrix factorization for graph clustering." *Proceedings of the 2012 SIAM international conference on data mining*. Society for Industrial and Applied Mathematics, 2012.

Summary

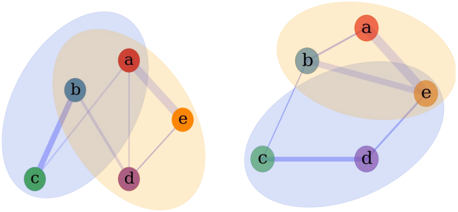
Connectomes



Non-overlapping partitions

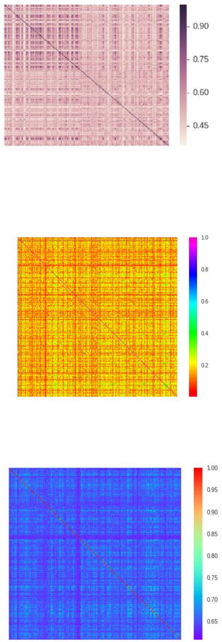


Overlapping clusters

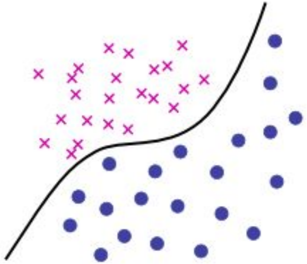


Frobenius norm between adjacency matrices (all pairwise comparisons)

Kernel matrix



SVM with precomputed Kernel matrix



Unauthorized figures

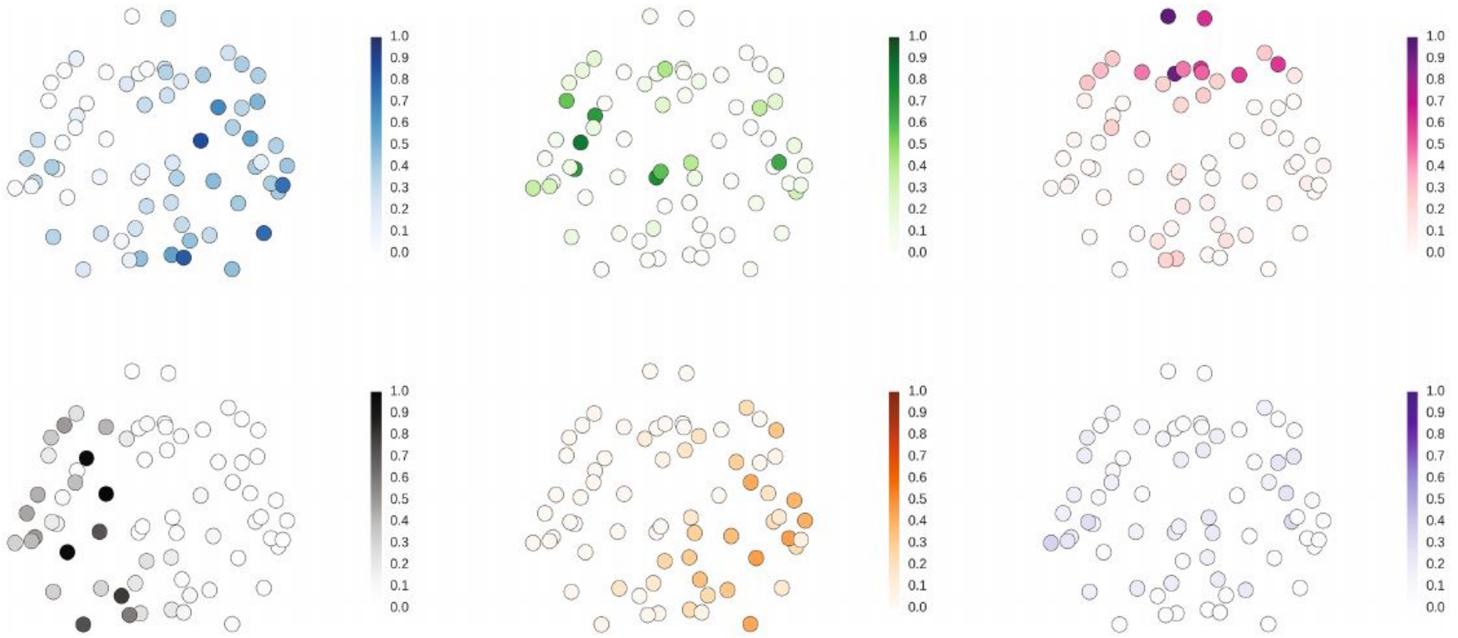


Fig. 2. Six overlapping communities: an example of a single network (healthy subject) with the nodes shown in their original 3D coordinates (axial view); color intensity is proportional to the strength of belonging to the respective community

Unauthorized figures

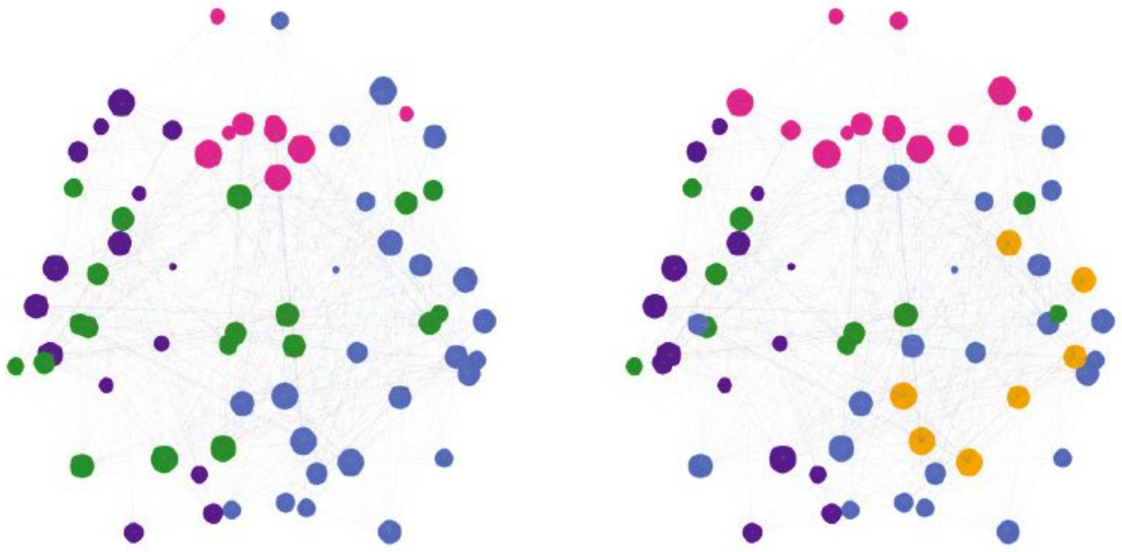


Fig. 3. Comparison of the non-overlapping (left) and overlapping (right) community structures obtained for the same example graph as in Fig. 3; node size is proportional to its degree (the number of edges coming from the respective node). Right plot is produced by selecting a single community for each node based on the maximal membership probability.

igraph, networkx

Installation

igraph is on the Python Package Index so it can be installed using `pip` or `easy_install`:

```
$ pip install python-igraph
```

networkx 2.2

Latest version

```
pip install networkx==2.2
```

Last released: Sep 19, 2018

[networkx docs]

[igraph]

community_label_propagation(*weights*=None, *initial*=None, *fixed*=None)

Finds the community structure of the graph according to the label propagation method of Raghavan et al. Initially, each vertex is assigned a different label. After that, each vertex chooses the dominant label in its neighbourhood in each iteration. Ties are broken randomly and the order in which the vertices are updated is randomized before every iteration. The algorithm ends when vertices reach a consensus. Note that since ties are broken randomly, there is no guarantee that the algorithm returns the same community structure after each run. In fact, they frequently differ. See the paper of Raghavan et al on how to come up with an aggregated community structure.

Parameters

- `weights`: name of an edge attribute or a list containing edge weights
- `initial`: name of a vertex attribute or a list containing the initial vertex labels. Labels are identified by integers from zero to *n-1* where *n* is the number of vertices. Negative numbers may also be present in this vector, they represent unlabeled vertices.
- `fixed`: a list of booleans for each vertex. `True` corresponds to vertices whose labeling should not change during the algorithm. It only makes sense if initial labels are also given. Unlabeled vertices cannot be fixed.

Return Value

an appropriate `VertexClustering` object.

Overrides: igraph.GraphBase.community_label_propagation

Reference: Raghavan, U.N. and Albert, R. and Kumara, S. Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76:036106, 2007. *http://arxiv.org/abs/0709.2938*.

community_leading_eigenvector(*clusters*=None, *weights*=None, *arpack_options*=None)

community_multilevel(*self*, *weights*=None, *return_levels*=False)

community_leading_eigenvector_naive(*clusters*=None, *return_merges*=False)

community_infomap(*self*, *edge_weights*=None, *vertex_weights*=None, *trials*=10)

community_fastgreedy(*self*, *weights*=None)

community_edge_betweenness(*self*, *clusters*=None, *directed*=True, *weights*=None)

ctrl+F “community”

[igraph manual]

[Networkx Louvain clustering]

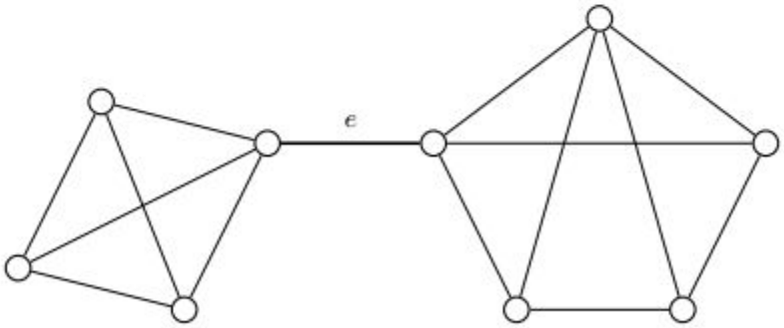
Examples

2.4 Divisive clustering and betweenness centrality

Newman and Girvan have proposed a divisive graph clustering algorithm based on the betweenness centrality measure (see Algorithm 3.4.1) [13]. The betweenness centrality measure was initially introduced by Freeman to identify the vertices with high potential to control communication in social networks [14].

The main idea of the betweenness centrality measure is the assumption that a vertex $v \in V$ is important for the communication in a social network if there are many vertices $a, b \in V$ whose shortest path include v. Higher importance for the communication in a social network gives higher betweenness centrality. If v has high betweenness centrality and v is removed from the graph, then the communication between many pairs of vertices becomes affected.

The graph clustering algorithm proposed by Newman and Girvan uses a generalization of the betweenness centrality for edges instead of vertices (see equation 3.4.1). Edges with high betweenness are potentionally a connection between two dense subgraphs (see figure 2.4.1), which is utilized by the algorithm by connecting clusters with edges of high betweenness centrality.



link

Figure 2.4.1: The edge e has the highest betweenness centrality in this graph. Therefore, e is likely to be an edge between two clusters by the means of the graph clustering algorithm from Newman and Girvan.