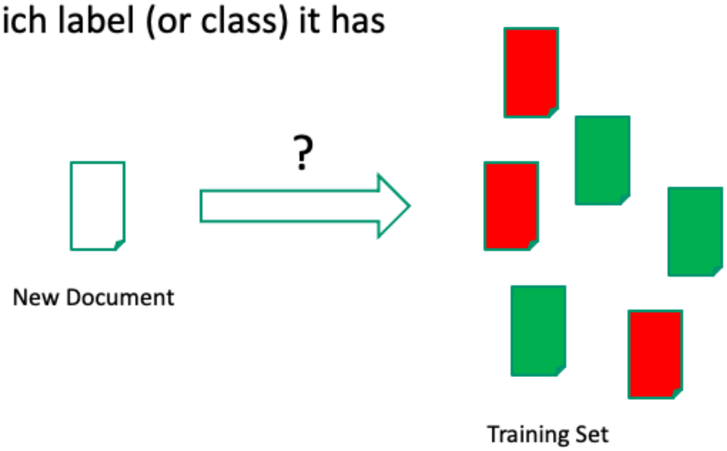
6. DOCUMENT CLASSIFICATION

Document Classification

Task: Given a training set of labelled documents, construct a **classifier** decide for an unlabeled document which label (or class) it has



Document classification is a special case of classification where the objects to be classified are (unstructured documents). This task has huge importance in many application areas, such as spam filtering, sentiment analysis, document filtering etc. and has therefore been studied as a specific classification problem in very much detail.

Document Classifiers

Features

- Words of the documents
  - bag of words
  - document vector
- More detailed information on words
  - Phrases
  - Word fragments
  - Grammatical features
- Any metadata known about the document and its author

An important question in document classification is the choice of features. A wide variety of possible features can be derived from text documents, many of which correspond to features used in document representation of information retrieval.

Example

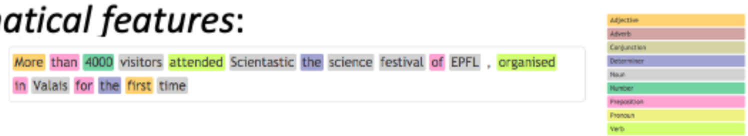
PromotionDesSciences @EPFL_SPS · May 1
More than 4000 visitors attended Scientastic the science festival of **EPFL**, organised in Valais for the first time. actu.epfl.ch/news/scientast…

Bag of words: {more, than, visitors, attend, …}

Phrases: {“science festival”, “first time”}

Word fragments: {mor, ore, tha, han, vis, isi, ..}

Grammatical features:

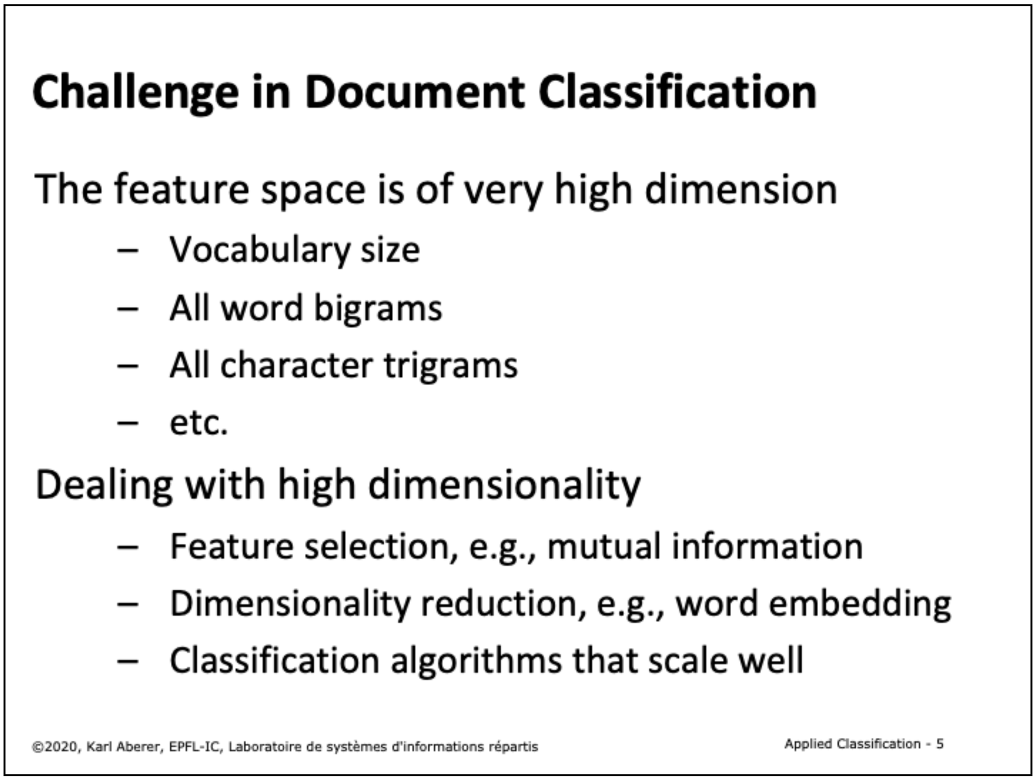


http://parts-of-speech.info/

Author: @EPFL_SPS

Applied Classification - 4

Here we give a few examples of features that could be extracted from a simple tweet, as shown above.



A problem that is characteristics for document classification is the fact that the feature space for documents can be huge. From information retrieval we know that the vocabulary size of a large document collection can grow to significant sizes, reaching millions of terms. When considering more features, such as word bigrams to capture phrases or n-grams in words, the size of the features space further explodes.

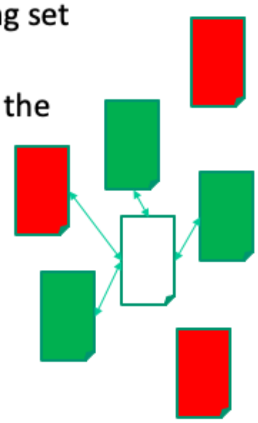
In order to deal with the high dimensionality of the feature space, different routes have been explored. A first is to perform feature selection, using e.g. mutual information, and retain only features that are specific for classification (with all the potential drawbacks discussed earlier). A second approach is to use dimensionality reduction methods, such as word embeddings or LSI, and represent then documents in the much lower dimensional concept space. Finally, one may also focus on using classification algorithms that scale well with dimensionality.

Simple Method: k-Nearest-Neighbors (kNN)

Approach: use vector space model

To classify a document D

- retrieve the k nearest neighbors in the training set according to the vector space model
- Choose the majority class label as the class of the document



If k large: $\frac{\#C}{k}$ estimates $P(C|D)$, the probability that the document has indeed class C

Actually, a very straightforward method for document classification, that scales well with dimensionality, is inspired by information retrieval. In the kNN method, the top k most similar documents according to the vector space model are retrieved, and then the majority label is used as the classification of the document. Strictly speaking with this method no classifier is learnt, but the classifier consists of the vector space representation of the whole document collection, so learning is trivial in that case. The method works in practice well. One critical choice is the parameter k, that determines how large the neighborhood is that is taken into account, and can be understood as the complexity of the model. In line with what we have seen on the dependency of bias and variance on model complexity. In the context of kNN a model is complex when k is small, since for every document very different sets of neighbors are chosen. Correspondingly it holds that for small k (complex model) we have low bias, but high variance and for large k we have high bias and low variance.

Naïve Bayes Classifier

Approach: apply Bayes law

Feature: Bag of words model: all words and their counts in the document

Using Bayes law the probability that document D has the class C is

$$P(C|D) \propto P(C) \prod_{w \in D} P(w|C)$$

Another frequently used method used in document classification is Naïve Bayes Classifier. Like kNN can be understood as the analogue of classification for vector space retrieval, Naïve Bayes can be understood as the analogue of classification for probabilistic retrieval. By applying Bayes law we can derive the probability of a document belonging to a class, from two estimators (probabilities) that have to be learnt from the training data. For this method to work, words not occurring in the training set, but in the test set, need to have non-zero probabilities.

Naïve Bayes Classifier Method

Training

How characteristic is word w for class C?

$$P(w|C) = \frac{|w \in D, D \in C| + 1}{\sum_{w'} |w' \in D, D \in C| + 1}$$ Laplacian smoothing

How frequent is class C?

$$P(C) = \frac{|D \in C|}{|\mathcal{D}|}$$

Classification: Thus the most probable class is the one with the largest probability

$$C_{NB} = \underset{C}{\operatorname{argmax}} \left(\log P(C) + \sum_{w \in D} \log P(w|C) \right)$$

For training the model we have to compute for each class how likely a word is to appear in that class and we have to compute the relative frequencies of the classes, i.e. the probability of a class label to occur. Note that in the estimation for P(w|C) we add +1 to the word counts. This is called Laplacian smoothing and is necessary to avoid to assign zero probability to terms that do not occur in the training set, to avoid that subsequently a document that does contain such a term is considered as impossible when using the classifier. The reasoning is the same why smoothing has been used in probabilistic information retrieval.

Classification is performed by selecting the class with the highest probability to occur. This is computed from the logarithm of the estimated probability, as the summation is numerically more stable than multiplication.

Classification Using Word Embeddings

Reminder

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Probability that word w occurs with context word c

$$P(D = 1 | w, c; \theta) = \frac{1}{1 + e^{-v_c \cdot v_w}}$$

For document classification word embeddings can be used in different ways. One obvious way would be to use word embedding vectors as word representation, and to represent then documents as an aggregate of those vectors to obtain a low-dimensional feature space. However, another possibility is to use the word embedding approach in a more direct way, which has resulted in a classification algorithm that has been recently developed at Facebook and is known as Fasttext.

Classification Using Word Embeddings

Consider instead of (word, context) pairs, (class, paragraph) pairs

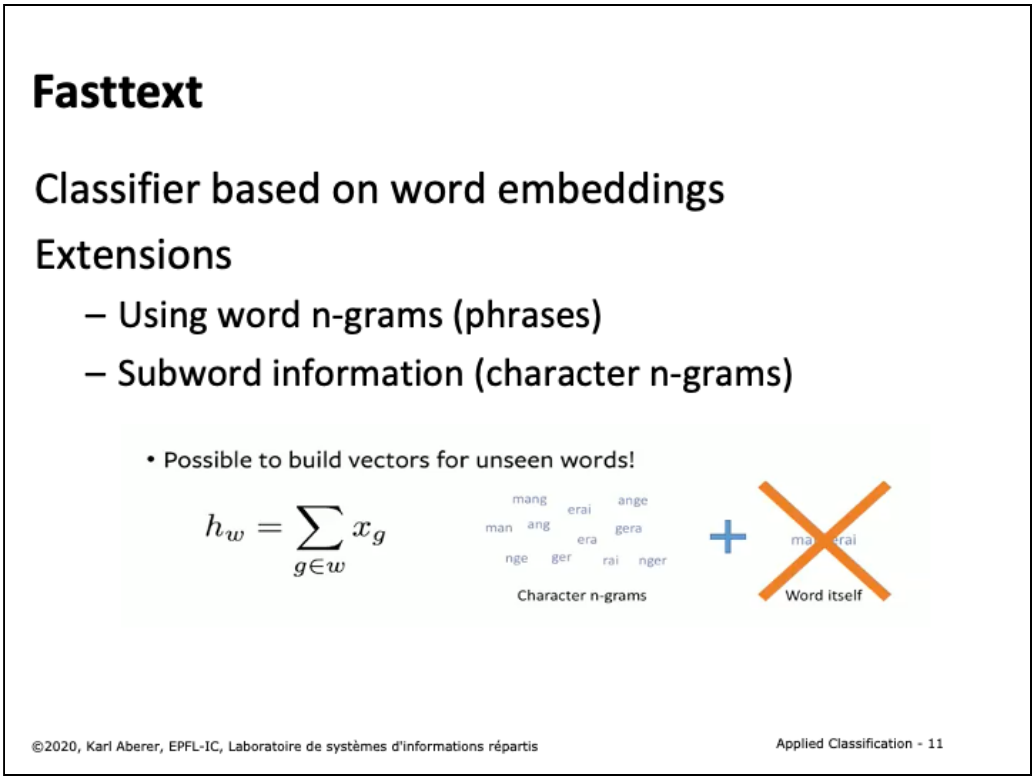
- Word w → Class Label C
- Context words c → Paragraph p

Learn $P(C|p) = \frac{e^{v_p \cdot v_C}}{\sum_{C'} e^{v_p \cdot v_{C'}}}$ (SGD)

Paragraph feature: $v_p = \sum_{w \in p} v_w$

Then predict label from paragraph features

In this approach to document classification the idea of creating word embeddings is slightly modified. Instead of trying to predict the a word from its context, for document classification one tries to predict the class label from a text fragment, e.g. a paragraph. Technically speaking, the two problems are equivalent, as in both cases we try to predict some word from a set of words. Thus the same method for learning the word embedding can be applied, using stochastic gradient descent. Once the model is learnt, as with Naïve Bayes the class label with the highest probability to occur is chosen as prediction for the document class. If there is not enough training data, instead of learning the word vectors v_w also pre-trained vectors from other document collections can be used.



In Fasttext different variations of the approach, using different feature sets have been explored, including the use of word phrases and character n-grams. One interesting implication of using character n-grams is that the classifier can even use feature vectors for unseen words, when they share a large number of n-grams in common with known words. The example illustrates that for example in French this can be indeed very useful.

Document Classification: Summary

Widely studied problem with many applications

- Spam filtering, Sentiment Analysis, Document Search
- Increasing interest
  - Powerful methods using very large corpuses
  - Information filtering on the Internet

Rank	Data Mining Methods	# Papers
1	SVM	55
2	KNN	31
3	NB	23
4	ANN	10
5	Rocchio Algorithm	9
6	Association Rule Mining	4

Rank	Feature Selection Methods	# Papers
1	Chi-squared test(CHI)	21
2	Information Gain (IG)	20
3	Mutual Information(MI)	16
4	Latent Semantic Indexing (LSI), Singular Value Decomposition (SVD)	10
5	Document Frequency (DF)	8
6	Term Strength (TS)	3
7	Odds Ratio(OR)	3
8	Linear Discriminant Analysis (LDA)	3

Document classification is an area of high, both because it is required in a growing number of application domains and because at the same time the classification methods are becoming increasingly powerful due to the availability of very large document corpuses and the growing computational power available. We have discussed only a few very well known methods. The tables show the findings of a literature review on different approaches to document classification, both in terms of algorithms used and of feature selection methods applied. This gives a sense of the diversity of this field.

For which document classifier the training cost is low and testing is expensive?

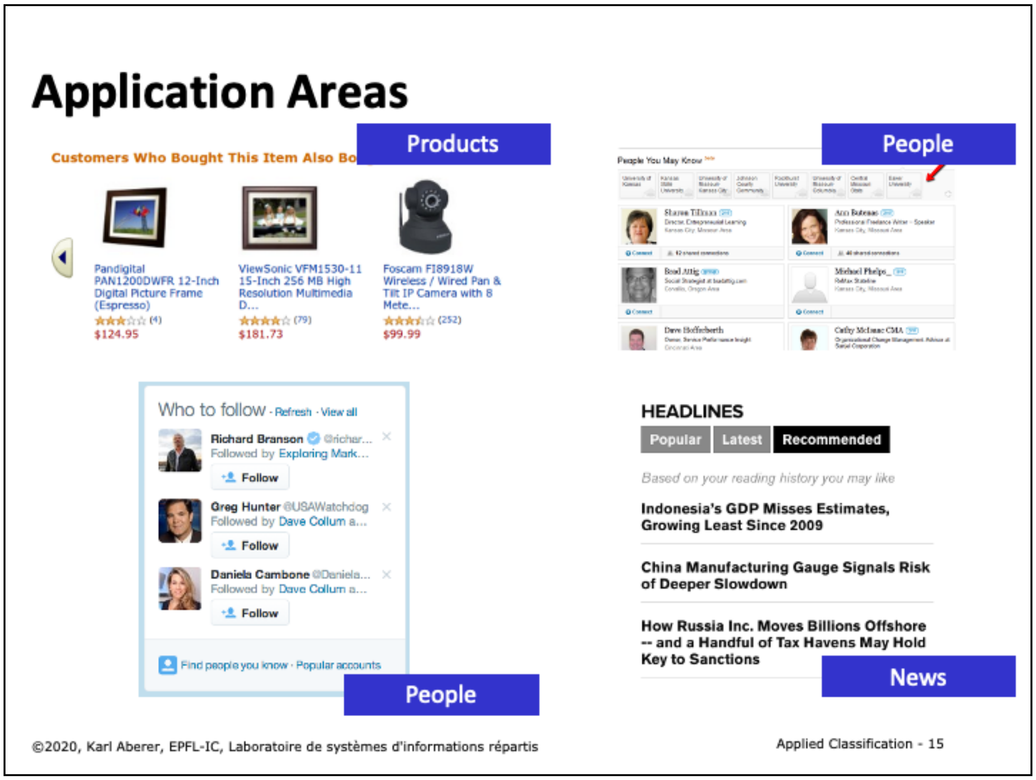
A. for none

B. for kNN Correct

C. for NB

D. for fasttext

7. RECOMMENDER SYSTEMS



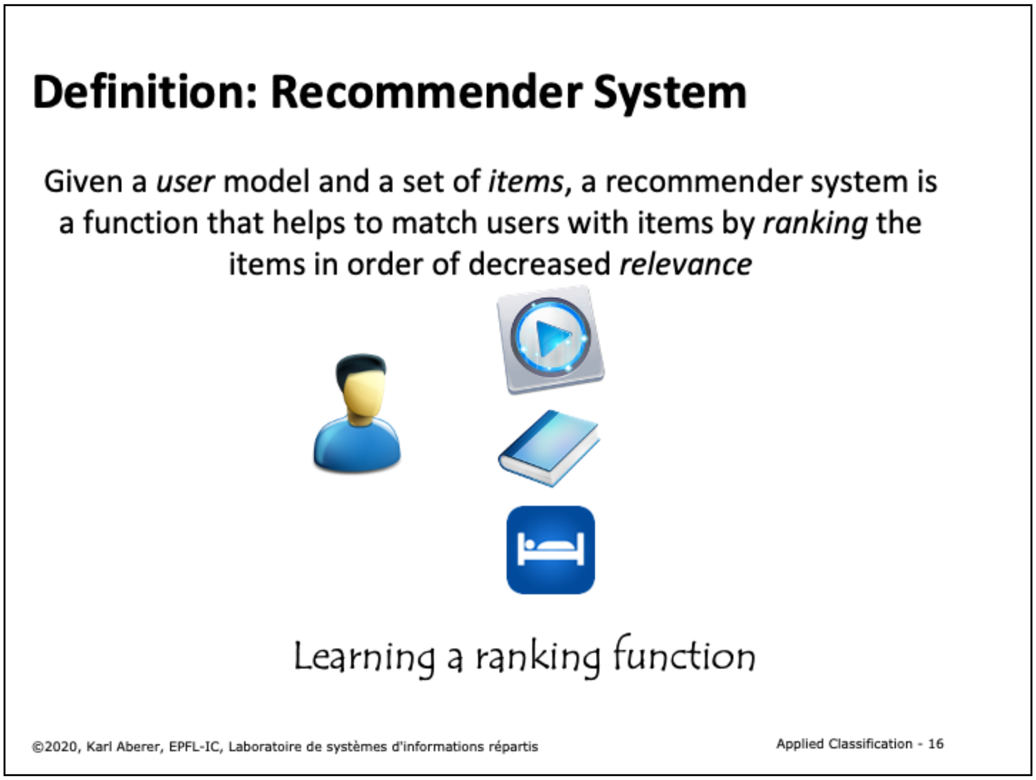
Recommender systems are widely used by many websites and applications to provide value for the user and the provider.

For users recommender systems help, e.g. in the context of a product search,

– find things that are interesting
– narrow down the set of choices
– help explore the space of options
– Discover new things

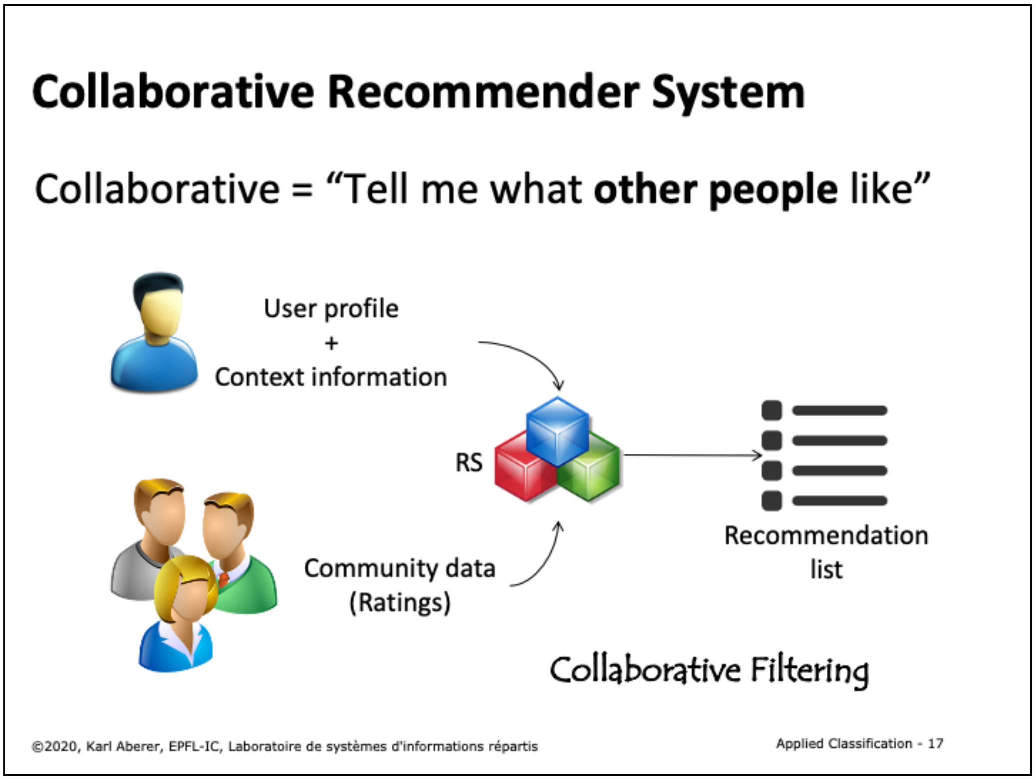
For providers recommender systems help to

– increase trust and customer loyalty with personalized services
– increase sales, click rates, page views etc.
– identify opportunities for promotion and persuasion
– obtain more knowledge about customers

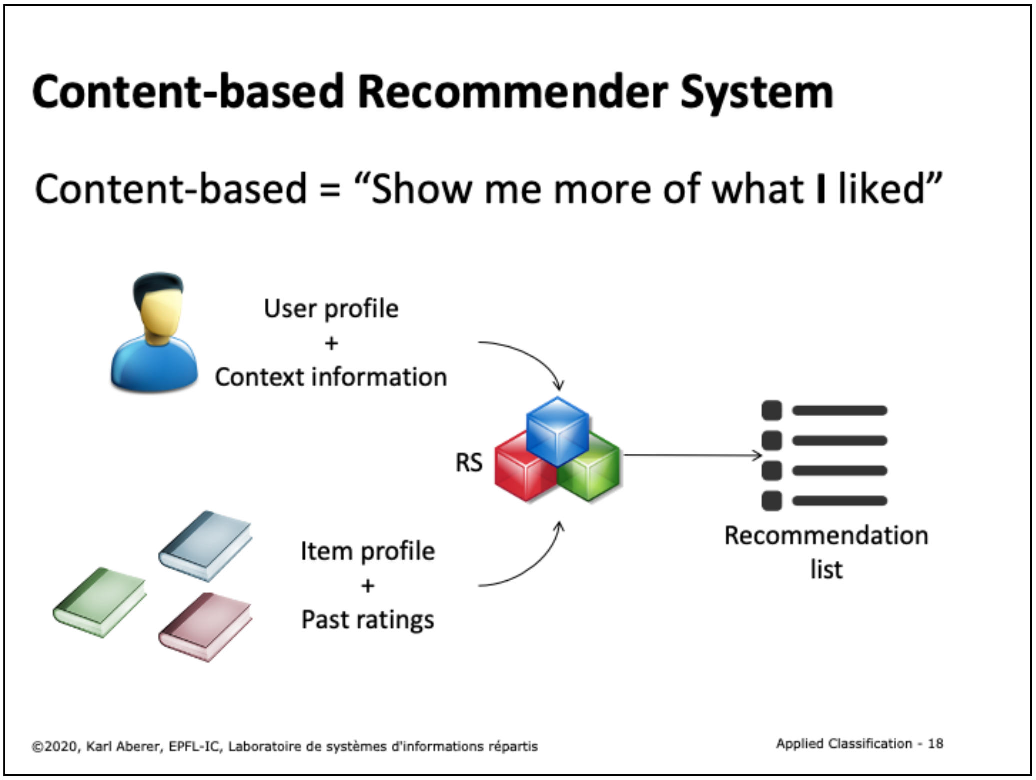


A recommender systems identifies for a given user a set of items that are relevant to her. This can result in a ranking of the items (modelling a relevance function) or a classification.

The user model typically includes data such as ratings, preferences, demographics and attributes that characterize the user The items can also be enriched with attributes about their content and characteristics, although it is not always necessary.



In the collaborative paradigm, the recommender system uses the user profile (and possibly other information about the specific context in which the user needs a recommendation) and the data provided by other users to return a recommendation list with the most relevant items and their scores, from the highest (i.e., most relevant) to the lowest (i.e., least relevant). It is called collaborative because all the user participates to the recommendation by providing ratings to the items that they viewed/purchased etc.



In the content-based paradigm, the recommender system uses the profile of the items that the user rated in the past to return a recommendation list with the most relevant items and their scores, from the highest (i.e., most relevant) to the lowest (i.e., least relevant). It is called content-based because the items are not only objects, but they are defined by a set of attributes that summarize the content of the item.

There are also other approaches, such as hybrid ones, which combine collaborative and content-based, or knowledge-based ones, which require domain knowledge engineering to model how certain item features meet the user needs.

Collaborative Filtering

A widely used approach to generate recommendations
- Used by large e-commerce sites
- Applicable in many domains (e.g., books, movies ...)
- Well understood and studied

Approach (*Wisdom of the crowd*)
- Users give ratings to items
- Users with similar tastes in the past will have similar tastes in the future

Collaborative recommender systems are based on collaborative filtering. In collaborative filtering, given a user and an item not yet rated by the user, the goal is to estimate the user rating for this item by looking at the ratings for the same item that were given in the past by similar users. Collaborative filtering requires a community of users that provide ratings and a way to assess user similarity.

User-based Collaborative Filtering

Basic technique:

Given a user x and an item i not rated by x, estimate the rating $r_x(i)$ by

1. Finding a set of users $N_U(x)$ who rated the same items as x (= neighbours of x) in the past and who have rated i
2. Aggregate the ratings of i provided by $N_U(x)$

Compute the ratings for all the items not rated by x and recommend the best-rated ones

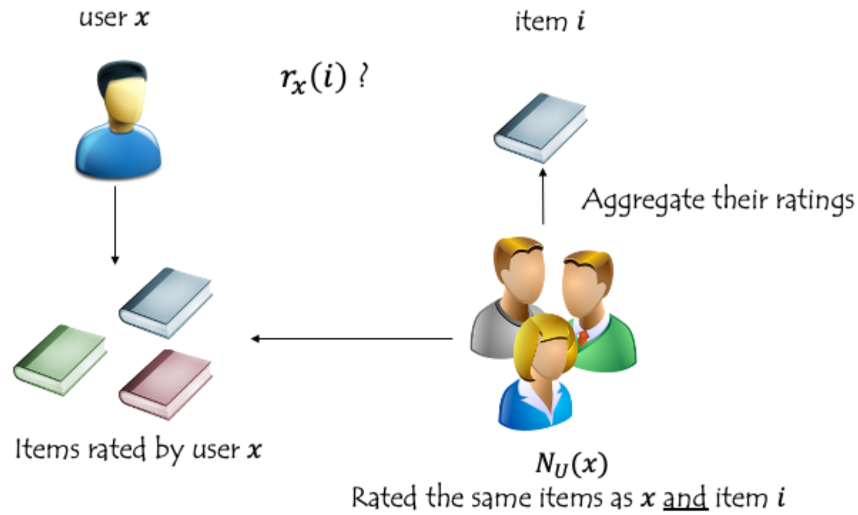
The basic technique for user-based collaborative filtering needs to define

1 A metric to compute similarity between users

2 The number of neighbours considered for $N_U(x)$

3 A way to aggregate the ratings of I provided by $N_U(x)$

User-based Collaborative Filtering illustrated



Are all users in $N_U(x)$ the same or are some more relevant than others?
➔ similarity between users

Similarity Between Users

Pearson correlation coefficient

$$sim(x,y) = \frac{\sum_{i=1}^{N}\left(r_x(i) - \overline{r_x}\right)\left(r_y(i) - \overline{r_y}\right)}{\sqrt{\sum_{i=1}^{N}\left(r_x(i) - \overline{r_x}\right)^2}\sqrt{\sum_{i=1}^{N}\left(r_y(i) - \overline{r_y}\right)^2}}$$

Cosine Similarity

$$sim(x,y) = \cos(\vartheta) = \frac{\sum_{i=1}^{N} r_x(i) \cdot r_y(i)}{\sqrt{\sum_{i=1}^{N} r_x(i)^2}\sqrt{\sum_{i=1}^{N} r_y(i)^2}}$$

x, y: users

N : number of items i rated by <u>both</u> x and y

$r_x(i)$: rating of user x of item i

r_x : average ratings of user x

A widely used similarity measure is the Pearson correlation coefficient, which returns a value between -1 and 1. The cosine similarity considers the users x and y as two N-dimensional vectors. Computing similarity follows exactly the same principles as for document similarity in vector space retrieval. If the angle between the two vectors is 0, thus the users are identical, and the cosine similarity returns 1. The greater the angle, the less similar the two users are. Assuming that all ratings are positive, the maximum angle is π/2, thus the minimum similarity is cos(π/2)=0.

Example

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

$sim_{corr}(U, User1) = 0.85$

$sim_{corr}(U, User2) = 0.71$

$sim_{corr}(U, User3) = 0$

$sim_{corr}(U, User4) = -0.79$

$sim_{cos}(U, User1) = 0.97$

$sim_{cos}(U, User2) = 0.99$

$sim_{cos}(U, User3) = 0.89$

$sim_{cos}(U, User4) = 0.79$

This is an example of similarity computed using the two metrics. It is possible to see how the users are not ranked in the same way. For example, the most similar user to user U is User 1, if the Pearson correlation coefficient is used, or User 2, if the cosine similarity is used. Also User 4 is not that different from U using the cosine similarity, although he seems to have quite opposite tastes.

One drawback of Pearson correlation coefficient is that it is not computable, if the variance of one of the user ratings is 0 (e.g., a user with ratings 1 1 1 1). However, in general, the correlation coefficient works well in general domains, particularly compared to the cosine similarity. One problem of cosine similarity is that it does not consider the absolute values of the ratings but only their relative distribution. This is appropriate when assessing the similarity of documents in information retrieval, since the absolute length of a document should not affect the similarity. However this does not work well with ratings. For example, two users with ratings 5 5 5 5 and 1 1 1 1 for item 1 to 4 have a cosine similarity of 1, since the vectors are parallel (albeit they have different magnitude). However, the users are not really similar, as the first likes everything while the second likes nothing.

Aggregate the Ratings

A common aggregation function is

$$r_x(a) = \bar{r}_x + \frac{\sum_{y \in N_U(x)} sim(x,y)(r_y(a) - \bar{r}_y)}{\sum_{y \in N_U(x)} |sim(x,y)|}$$

$N_U(x)$: neighbours of user x

a : item not rated by x

The aggregation function calculates whether the neighbours' ratings for the unseen item *a* are higher or lower than their average. We can consider this term as the bias of the user *y* w.r.t. item *a*. Then the function combines the rating bias using the similarity as a weight, so that the most similar neighbours will have more importance. Then the aggregated neighbours' bias is added/subtracted from user's *x* average rating.

Example

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

Closest users to U: $sim_{corr}(U, User1) = 0.85$, $sim_{corr}(U, User2) = 0.71$

$\bar{r}_U = 4$

$(r_{U1}(I5) - \bar{r}_{U1}) = 3 - \frac{12}{5} = \frac{3}{5}, (r_{U2}(I5) - \bar{r}_{U2}) = 5 - \frac{19}{5} = \frac{6}{5}$

$$r_U(I5) = 4 + \frac{0.85 * \frac{3}{5} + 0.71 * \frac{6}{5}}{0.85 + 0.71} = 4.87 \approx 5$$

User-based Collaborative Filtering

Problems

- Cold start: users or items without ratings
- Scalability: large numbers of users
- Data dispersion: highly variable ratings, difficult to find similar users

Possible solution

- Item-based collaborative filtering

A typical problem of user-based collaborative filtering is the cold-start problem, that is, the recommendation for a new user that did not rate anything yet or for an item that was never rated. Also, user-based collaborative filtering is problematic when there are millions of users and/or items (Amazon, Netflix). For example, for each user the most similar users have to be found (nearest neighbours) from a large set of users. With large numbers of users, it becomes hard to find common ratings between users, due to the high dispersion of data.

Item-based Collaborative Filtering

The basic technique: use the similarity between items (and not users) to make predictions

- Given a user x and an item i not rated by x, estimate the rating $r_x(i)$ by

  1. Find a set of items $N_I(i)$ which are similar to i and that have been rated by x
  2. Aggregate the ratings of the items in $N_I(i)$ and use the aggregation as prediction of $r_x(i)$

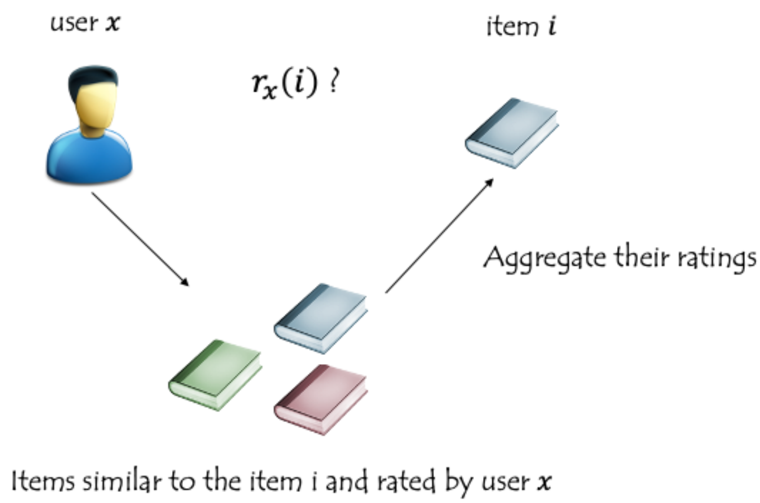
The basic technique for item-based collaborative filtering needs to define

1 A metric to compute similarity between items

2 The number of neighbours considered in N_I

3 A way to aggregate the ratings of the items in N_I

Item-based Collaborative Filtering illustrated



Similarity Between Items

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

$sim_{corr}(item5, item1) = 0.97$

$sim_{corr}(item5, item2) = -0.48$

$sim_{corr}(item5, item3) = -0.43$

$sim_{corr}(item5, item4) = 0.58$

The attributes that define an item are the ratings of the users different than U. The similarity can be computed with the same functions explained before, correlation coefficient or cosine similarity. In this example, similarity is computed with the correlation coefficient, and it is possible to see how item 1 and 4 are the most similar to item 5. Thus, aggregating the ratings of these items given by user U is an estimate of the rating of user U for item 5.

Aggregate the Ratings

A common aggregation function is

$$r_x(a) = \frac{\sum_{b \in N_I(a)} sim(a,b) r_x(b)}{\sum_{b \in N_I(a)} |sim(a,b)|}$$

$N_I(a)$: neighbours of item a

b : item rated by x

The aggregation function computes the prediction of an item a for a user x by computing the sum of the ratings given by the user on the items similar to a. Each rating is weighted by the corresponding similarity sim(a,b) between items a and b.

Example

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

Closest items: sim_{corr}(item5, item1) = 0.97, sim_{corr}(item5, item4) = 0.58

$r_U(item5) = (0.97*5 + 0.58*4)/(0.97 + 0.58) = 4.62 \approx 5$

Item-based Collaborative Filtering

Item-based collaborative filtering does not solve the cold-start problem but has better scalability

- Calculate all the pair-wise item similarities in advance
- Items similarities are more stable
- The neighbourhood $N_I(a)$ to be considered at runtime is small

Although item-based collaborative filtering does not solve the cold-start problem, it is usually more suited to tackle big datasets. In fact, the item similarities can be computed in advance, given that the items are more stable than the users (in general, new items appear at a slower pace than new users). Furthermore, the size of the neighbourhood of a given item a is in general smaller than the neighbourhood of a user x, because $N_I(a)$ is composed of the other items rated by user x (e.g., tens of DVDs bought on Amazon), while $N_U(x)$ is composed by the other users that rated the same items as x (e.g., millions of users that watched The Godfather on Netflix).

Given 3 users with ratings...

u1:	1	3
u2:	2	4
u3:	1	4

A. $Sim_{corr}(u1, u2) > Sim_{corr}(u1, u3)$

B. $Sim_{corr}(u1, u2) = Sim_{corr}(u1, u3)$ **Correct**

C. $Sim_{corr}(u1, u2) < Sim_{corr}(u1, u3)$

Item-based collaborative filtering addresses better the cold-start problem because …

A. usually there are fewer items than users

B. it uses ratings from items with similar content

C. item similarities can be pre-computed

D. none of the above Correct

Content-based Recommendation

Collaborative filtering uses only ratings, it does not exploit any other information about the items

However, the *content* of the item might provide some useful information

- E.g., recommend sci-fi novels to users who liked Asimov books in the past

Both recommendation techniques we saw before need only the ratings of the user, and no information about the item is needed. However, the content of the item might provide useful information for the recommendation. Content-based recommendation takes into consideration only the items that have been rated by the user and the content of all existing items, significantly reducing the scalability problems since a community of users is not necessary any more.

Content-based Recommendation

The basic technique

- Given the items that have been rated by user x in the past

  1. Find the items that are similar to the past items
  2. Aggregate the ratings of the most similar items

The basic technique for content-based recommendation needs to define

1 A way to formalize the item content

2 A similarity measure between items

3 An aggregation function for the ratings

Content-based Recommendation

Most methods originate from information retrieval to extract the content of an item

Given an item and a textual description (e.g. movie synopsis, book review), compute the tf-idf weights

$$w(t,a) = tf(t,a) \cdot idf(t) = \frac{freq(t,a)}{\max_{s \in T} freq(s,a)} \cdot \log\left(\frac{N}{n(t)}\right)$$

N : number of items to recommend

$n(t)$: number of items where term t appears

The first step for content-based recommendation is the characterisation of the items in terms of TF-IDF. Each item can be therefore described by the set of terms that appear in their description, with their associated weight (the TF-IDF measure)

To compute the TF-IDF measure, all the steps typical of information retrieval (removing stopwords, stemming, only top M terms etc.) are applied.

Similarity between Items

Cosine similarity

$$sim(a,b) = \cos(\vartheta) = \frac{\sum_{t=1}^{T} w(t,a) \cdot w(t,b)}{\sqrt{\sum_{t=1}^{T} w(t,a)^2} \sqrt{\sum_{t=1}^{T} w(t,b)^2}}$$

a, b: items

T : number of terms t that appear in all items

$w(t, a)$: tf-idf of term t in item a

The cosine similarity in content-based recommendation considers the items a and b as two T-dimensional vectors, where each dimension is the TF-IDF measure of a specific term t.

Making Predictions

Given a set of items D that have been rated by the user, find the nearest neighbours $N_I(a)$ in D of a new item a

Use the ratings of the nearest neighbours to predict the rating of a with the aggregation function seen before

$$r_x(a) = \frac{\sum_{b \in N_I(a)} sim(a,b) r_x(b)}{\sum_{b \in N_I(a)} |sim(a,b)|}$$

Once a way to compute the similarity between items has been established, using the vector space model (TF-IDF + cosine similarity), the simplest approach to estimate the rating of item a not yet seen/rated by user x is to find the nearest neighbours of a in the subset of items that have been already rated by the user x and aggregate their ratings. Note that this approach to predict the rating is analogous to the use of the kNN nearest neighbor classification for document classification. In both cases, the nearest documents are used as predictors for the most likely label respectively rating.

Content-based Recommendation

Problems

- Cold start problem for users with no ratings
- Content can be limited or impossible to extract (multimedia)
- Tends to recommend “more of the same”

Scalable: tf-idf of items can be computed offline

Content-based recommendation also suffers from the cold start problem as collaborative filtering, but only for the users that did not rate anything in the past. Another problem is that sometimes the information available to extract the content is limited (e.g., movie synopsis versus full plot) or impossible to process (e.g. a sound or a video). Finally, content-based recommendation tends to recommend more of the same, it does not surprise the user with new interesting items.

On the other hand, it is in general more scalable, because it does not rely on a community of users to provide recommendations. Furthermore, the TF-IDF measure can be computed once a new item enters the system, and then update the pair-wise similarities with all the existing items.

For a user that has not done any ratings, which method can make a prediction?

A. User-based collaborative RS

B. Item-based collaborative RS

C. Content-based RS

D. None of the above Correct

For an item that has not received any ratings, which method can make a prediction?

A. User-based collaborative RS

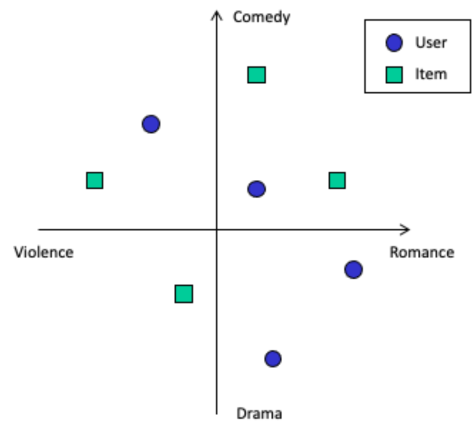
B. Item-based collaborative RS

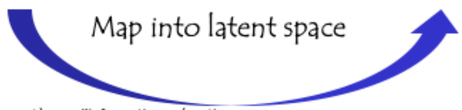
C. Content-based RS Correct

D. None of the above

Advanced Methods: Matrix Factorization

	I1	I2	I3	I4
U1	3	1	-	-
U2	-	3	4	3
U3	3	-	1	-
U4	-	5	5	2





Matrix factorization transforms the user-item rating matrix into a latent factor model, where each user and item is described as a vector in a k-dimensional space. It can be understood as a combination of user-based and item-based collaborative filtering.

For the items, the factors describe evident as well as non interpretable characteristics. For the users, each factor measures how much the user likes items that score high on the corresponding factor. The estimate of the rating of item a that user u would given can be computed as the dot product between the two low-dimensional matrices: $q_i^T \cdot p_u$ where q_i is the item vectors and p_u is the user vectors. The major challenge of course is computing the mapping of each item and user to their corresponding vectors in q_i and p_u. After that, the rating can be estimated using the dot product.

This approach is analogous to singular value decomposition for document retrieval. However, a direct application of SVD is not possible because the user-item matrix is not complete (missing ratings).

Derive Latent Factors

Rating Matrix R with dimension $|U|\times|D|$

- Users U and Items D

Goal: Decompose R (approximatively)

$$R \approx P\times Q^t = \hat{R}$$

K latent factors

- P is a $|U|\times K$ matrix: user features
- Q is a $|D|\times K$ matrix: item features

Problem: R has many undefined values (missing ratings)!

Here we formulate the factorization problem.

Approximation Error

For a given rating r_{ij}

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj} \right)^2$$

Minimizing error: compute gradient

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2 \left(r_{ij} - \hat{r}_{ij} \right) q_{kj} = -2 e_{ij} q_{kj}$$

$$\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2 \left(r_{ij} - \hat{r}_{ij} \right) p_{ik} = -2 e_{ij} p_{ik}$$

Thus, to learn the factor vectors q_i and p_u the system minimises the regularised square error on the set of known ratings (u,i) ∈ M. The minimization problem can be solved using stochastic gradient descent. To that end we first compute the gradients.

Stochastic Gradient Descent

Update Rule: for some random pair i, j

$$p_{ik} := p_{ik} - \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$

$$q_{kj} := q_{kj} - \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij}\, p_{ik}$$

Repeat until error us small

Needs only to be computed for ratings r_{ij} that exist!

With the gradients we can then define the update rules. Since we perform SGD, we need only to update error values for which a corresponding rating exists. As a side effect, once the decomposition is computed, we obtain also estimates for the ratings for the other user-item pairs, for which not rating exists in the training data.

Regularization

In order to avoid overfitting

$$e_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj} \right)^2 + \lambda(\|P\|^2 + \|Q\|^2)$$

And then

$$p_{ik} := p_{ik} - \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha\left(e_{ij} q_{kj} - \lambda p_{ik}\right)$$

$$q_{kj} := q_{kj} - \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha\left(e_{ij}\, p_{ik} - \lambda q_{kj}\right)$$

In order to avoid over-fitting, usually a regularization term is added and the update rule is modified correspondingly.

8. MINING SOCIAL GRAPHS

Mining Graphs

Up to now

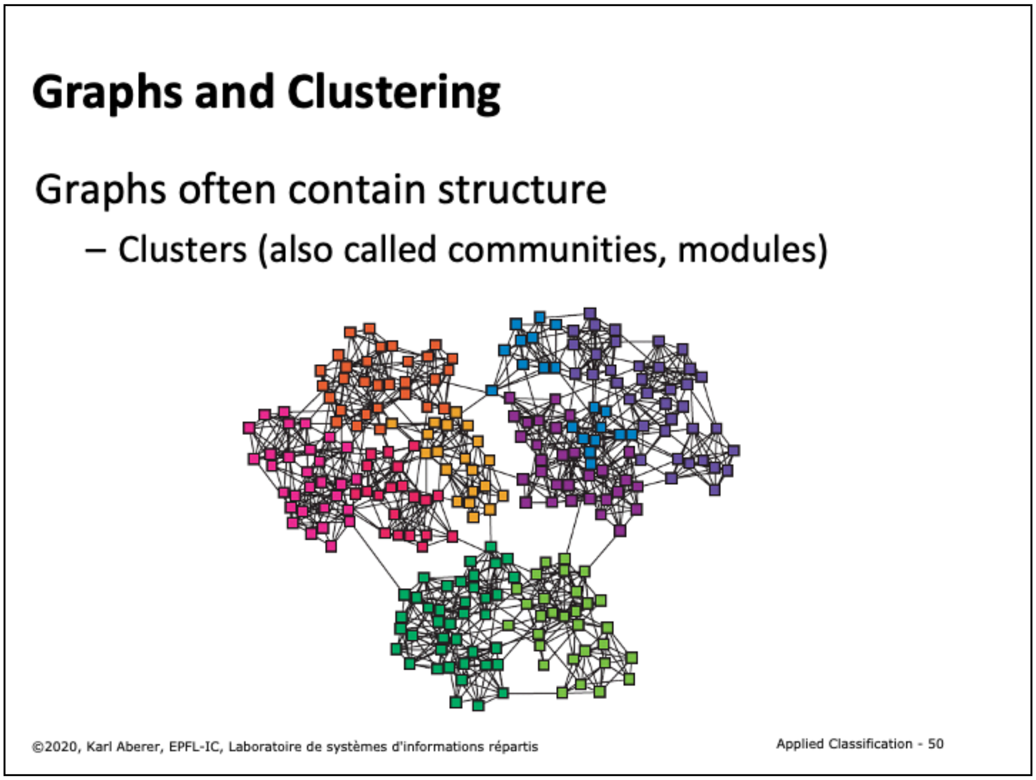
- Objects have been described by attributes
- Relationships among objects inferred from distance measure on attributes

In many cases explicit relationships are given

- Explicit relationships = Graphs
- Prime example: **social network graphs**
- Graph structure can also be inferred from distance measure (e.g. for documents)

So far we have assumed that relationships among different objects in clustering are implicitly defined, using a distance measure that is based on the objects' attributes. In many applications such relationships are not implicitly given, but explicitly provided, notably in social network graphs. There the relationships among users are given by different friend relationships or interactions such as likes and retweets. The resulting graphs can be weighted or unweighted. In the following we will consider the case of unweighted graphs.

Of course, it would also be possible to derive a relationship graph by using a distance measure based on attribute values and using the distance as as an edge weight, respectively consider only edges above a distance threshold. In this way the following methods of clustering objects based on graphs could also be applied in this more traditional context.

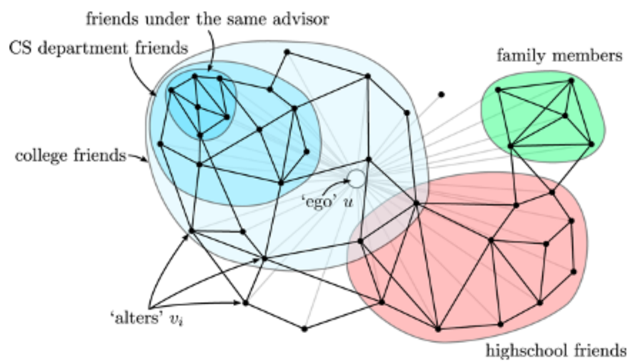


It is widely observed that natural networks contain structure. This is true for social networks (e.g. on social media platforms, citation networks), as well as for many natural networks as we find them in biology. Graph-based clustering aims at uncovering such hidden structures.

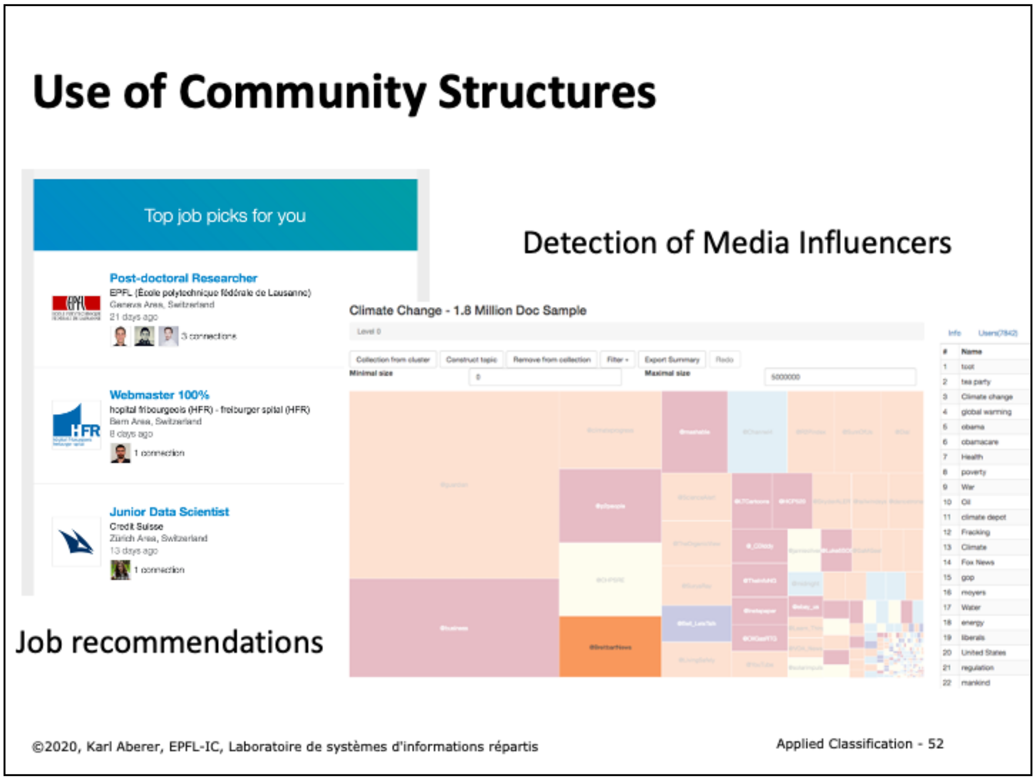
Social Network Analysis

Clusters (communities) in social networks (Twitter, Facebook) related to

- Interests
- Level of trust



In social networks mining community structures is particularly popular. Consider the social network neighborhood of a particular social network user, i.e. all other social network accounts that are connected to the user, either through explicit relationships, such as a follower graph, or through interactions such as likes or retweets. Typically the social neighborhood would decompose into different groups, depending on interests and social contexts. For example, the friends from high school would have a much higher propensity to connect to each other, than with members of the family of the user. Thus they are likely to form a cluster. Similarly, other groups with shared interest or high mutual level of trust would form communities.



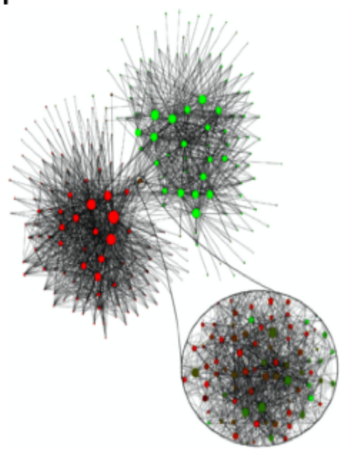
Many tasks can benefit from a reliable community detection algorithm. Online Social Networks rely on their underlying graph to recommend content, for example relevant jobs. Knowing to which community a user belongs can improve dramatically the quality of such recommendations.

Another typical use of community detection is to identify media influencers. Community detection can first be used to detect communities that share in social networks common interests or beliefs (e.g. for climate change we might easily distinguish communities that are climate deniers and climate change believers), and then the main influencers of such communities could be identified.

Use of Community Structures: Social Science

Call patterns in Belgium mobile phone network

- Two almost separate communities



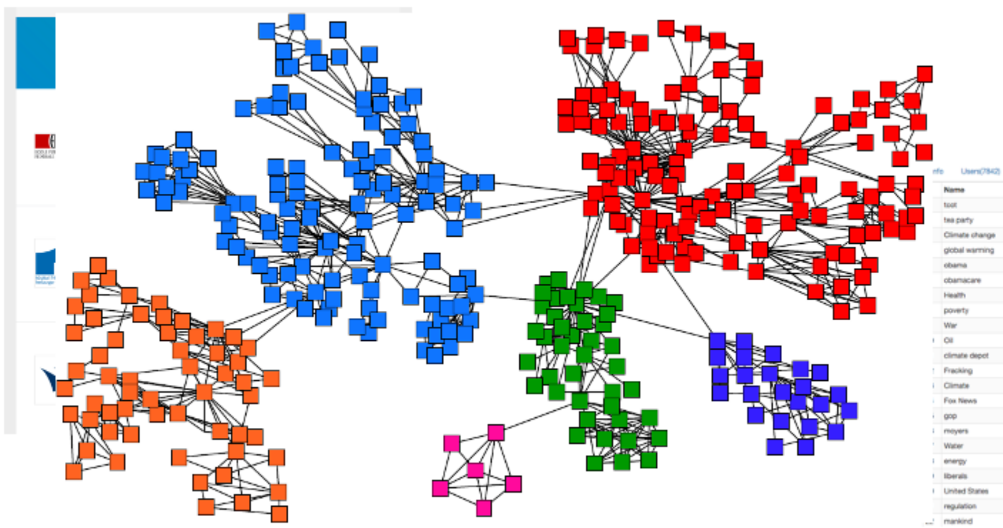
V.D. Blondel et al, *J. Stat. Mech.* P10008 (2008).

In 2008 Vincent Blondel and his students have started applying a new community detection algorithm to the call patterns of one of the largest mobile phone operators in Belgium. It was designed to identify groups of individuals who regularly talk with *each other* on the phone, breaking the whole country into numerous small and not so small communities by placing individuals next to their friends, family members, colleagues, neighbors, everyone whom they regularly called on their mobile phone. The result was somewhat unexpected: it indicated that Belgium is broken into two huge communities, each consisting of many smaller circles of friends. Within each of these two groups the communities had multiple links to each other. Yet, these communities never talked with the communities in the other group (guess why?). Between these two mega-groups was sandwiched in a third, much smaller group of communities, apparently mediating between the two parts of Belgium.

Which of the following graph analysis techniques do you believe would be most appropriate to identify communities on a social graph?

A. Cliques **Correct**
B. Random Walks
C. Shortest Paths
D. Association rules

Use of Community Structures



©2020, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Applied Classification - 52

The intuition behind community detection is that the heavily linked components of the graph belong to the same community.

Similarly as earlier for clustering in multidimensional spaces, therefore, the goal of a community detection algorithm is to identify communities that are heavily intra-linked (high intra-cluster similarity) and scarcely inter-linked (low inter-cluster similarity).

Types of Community Detection Algorithms

Hierarchical clustering

- iteratively identifies groups of nodes with high similarity

Two strategies

- **Agglomerative algorithms** merge nodes and communities with high similarity
- **Divisive algorithms** split communities by removing links that connect nodes with low similarity

In general, community detection algorithms are based on a hierarchical approach. The idea is that within communities sub-communities can be identified, till the network decomposes into individual nodes. In order to produce such a hierarchical clustering, two approaches are possible: either by starting from individual nodes, by merging them into communities, and recursively merge communities into larger communities till no new communities can be formed (agglomerative algorithms), or by decomposing the network into communities, and recursively decompose communities till only individual nodes are left (divisive algorithms).

In the following we will present one representative of each of the two categories of algorihtms:

1. The Louvain Algorithm, an agglomerative algorithm
2. The Girvan-newman algorithm, a divisive algorithm

Louvain Modularity Algorithm

Agglomorative Community Detection

- Based on a measure for community quality (**Modularity**)
- greedy optimization of modularity

Overall algorithm

- **first** small communities are found by optimizing modularity **locally** on all nodes
- then each small community is **grouped** into one new community node
- **Repeat** till no more new communities are formed

The Louvain algorithm is essentially based on a the use of a measure, modularity, that allows to assess the quality of a community clustering. The algorithm performs greedy optimization of this measure. It is fairly straightforward: initially every node is considered as a community. The communities are traversed, and for each community it is tested whether by joining it to a neighboring community, the modularity of the clustering can be improved. This process is repeated till no new communities form anymore.

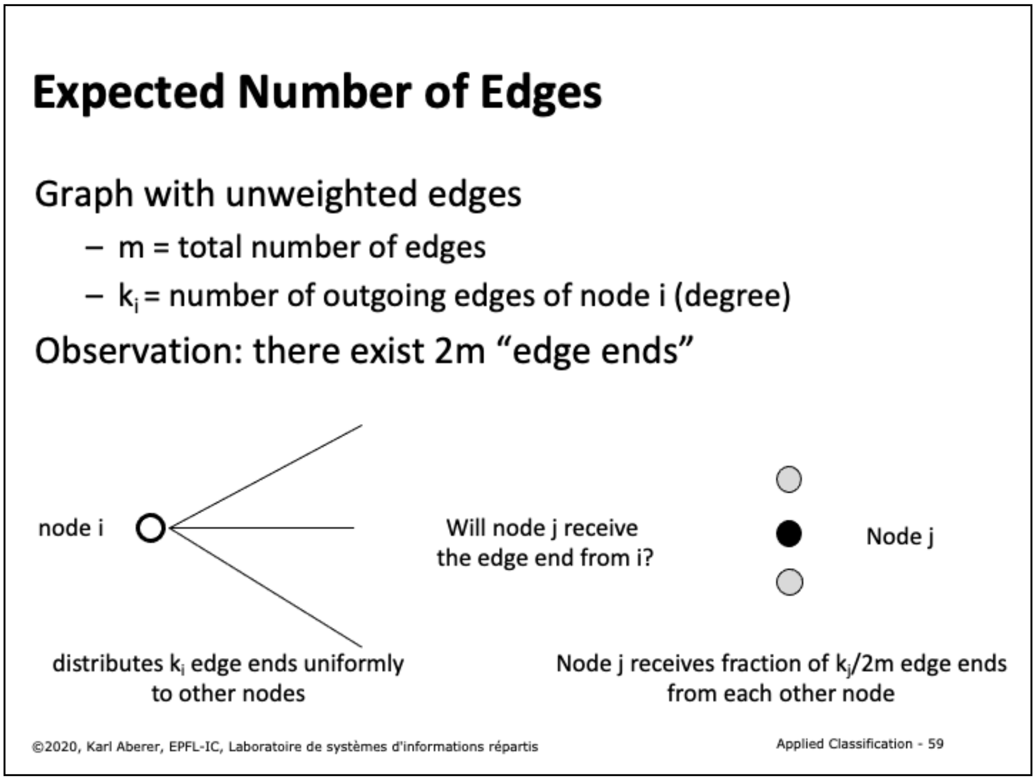
Measuring Community Quality

Communities are sets of nodes with many mutual connections, and much fewer connections to the outside

Modularity measures this quality: the higher the better

$$\sum_{C \in \text{Communities}} (\#\text{edges within } C - \text{expected } \#\text{edges within } C)$$

In order to measure the quality of a community clustering, modularity compares simply the difference between the number of edges that occur within a community with the number of edges that would be expected if the edges in the graph would occur randomly. The random graph model is used as what is called the null model, a graph that has the same distribution of node degrees, but randomly assigned connections.



The null model requires to determining the number of edges that we would expect among a set of nodes, of which we know the degrees, but not the connectivity in detail. Assume that for all nodes i in a community we would now their degree ki (number of edges leaving the node). How many edges would we then observe if the connections on the graph were generated randomly? To answer this question we reason as follows: select one of the nodes i with degree ki. What is now the probability (or fraction of weight) an arbitrary other node j with weight kj would receive? If there are a total of m edges in the network, there are 2m edge ends (since each edge ends in two nodes). If the edge ends of ki are uniformly distributed, node j would thus receive ki/2m edge ends. Thus the number of expected edges connecting nodes i and j would be ki times kj/2m.

Modularity

Modularity measure Q

- A_{ij} = effective number of edges between nodes i and j
- C_i, C_j = communities of nodes i and j

$$Q = \frac{1}{2m} \sum_{i,j} \Big(\underbrace{A_{ij}}_{\text{Effective number of edges}} - \overbrace{\frac{k_i k_j}{2m}}^{\text{Expected number of edges}} \Big) \delta(C_i, C_j)$$

Properties

- Q in [-1,1]
- 0.3-0.7 < Q means significant community structure

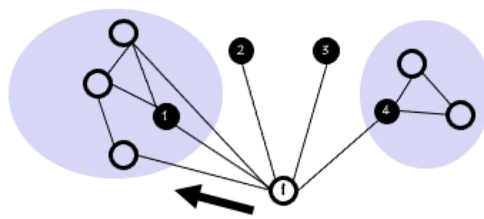
Given the expected number of edges (the null model) we can now formulate the modularity measure as the total difference of number of expected and effective edges for all pairs of nodes from the same cluster. The delta function assures that only nodes belonging to the same community are considered, since it returns 1 when ci = cj.

Due to normalization the measure returns calues between -1 and 1. In general, if modularity exceeds a certain threshold (0.3 to 0.7) the clustering of the network is considered to exhibit a good community structure.

Locally Optimizing Communities

What is the modularity gain by moving node i to the communities of any of its neighbors?

– Test all possibilities and choose the best

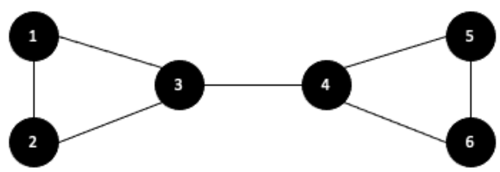


Given the modularity measure the next question is now how to use it to infer the communities. This is performed through local optimization. The algorithm sequentially traverses all nodes of the network, and for each node checks how the modularity can be increased maximally by having the node joining the node of a neighboring community. It then decides to join that node to the best community.

Example

Initial modularity Q = 0

Start processing nodes in order



We illustrate the algorithm for a simple example. Initially, the modularity is zero since all nodes belong to different communities. We start now to process the nodes in some given order, e.g. size of identifier.

Example: Processing Node 1

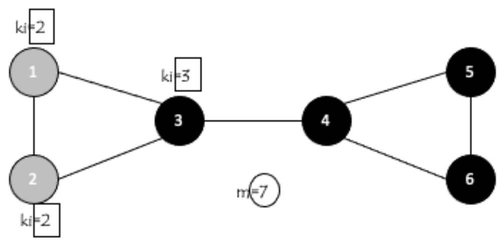
Joining node 1 to node 2

- Q = 1/2*7 (1 – 2*2/2*7) = 1/14 * 10 /14 > 0

Joining node 1 to node 3

- Q = 1/2*7 (1 – 2*3/2*7) = 1/14 * 8/14   > 0

New modularity: 1/14 * 10 /14



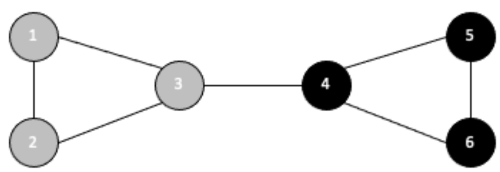
For node 1 we can join it either to node 2 or 3, it’s two neighbors. To decide which is better, we have to compute modularity after the join. We see that it is better to join node 2, as the resulting modularity will be higher. We can think of this as follows: since node 3 has more connections, it is more likely to be randomly connected to node 1, this connecting to node 2 is more “surprising”.

Example: Processing Nodes 2 and 3

Joining node 2 to node 3 (leaving community of node 1)
- no improvement

Joining node 3 to community {1,2} (via node 1 or 2)
- Q = 1/14 (3 – 4/14 – 6/14 – 6/14) = 1/14 * 26 / 14

Joining node 3 to node 4
- Q = 1/14 10/14 + 1/14 (1 – 9/14) = 1/14 * 15/14



Applied Classification - 64

For node 2 there will be no change, as the only alternative would be node 3. But for the same reasons node 1 did not join node 3, also node 2 does not. Next is node 3: here we have two choices, either to join community {1,2}, or to join node 4. In the first case we obtain a larger community, and in the second case two smaller communities. Computation of modularity reveals that joining 1 and 2 gives a much better community structure.

Example: Processing Nodes 4, 5 and 6

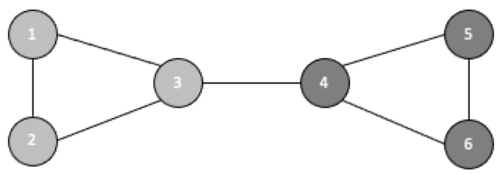
Joining node 4 to community {1,2,3} via 3

- Q = 1/14 (4 - 4/14 - 6/14 - 6/14 – 6/14 – 6/14 - 9/14) = 1/14 * 19/14

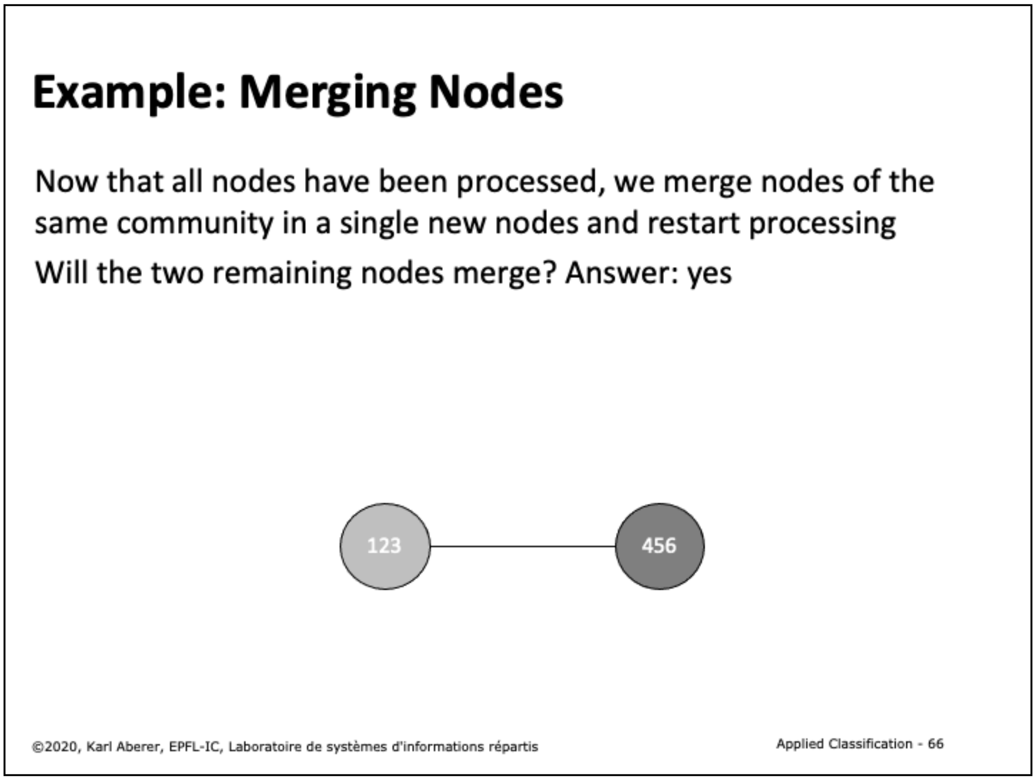
Joining node 4 to node 5

- Q = 1/14 * 26/14 + 1/14 * (1 – 6/14) = 1/14 * 34/14

Finally, also node 6 will join the second community



Similar arguments as before will then join node 4 to 5 and finally node 6 to the community consisting of nodes 4 and 5. Then the first round of processing is over.



For the next round of processing the resulting community nodes are collapsed in new nodes for the complete community, and the algorithm is re-run with the new resulting graph. Obviously, now the two remaining nodes will merge into a community, as the modularity will move from zero to positive. Then the algorithm terminates.

Modularity clustering will end up always with a single community at the top level?

A. true

B. Only for dense graphs

C. Only for connected graphs Correct

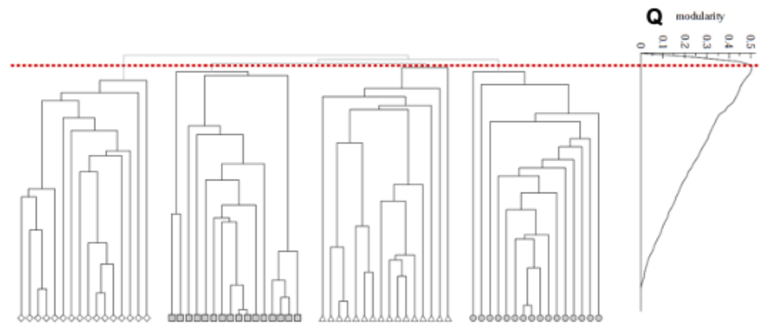
D. never

Modularity clustering will end up always with the same community structure?

A. true
B. Only for connected graphs
C. Only for cliques
D. false Correct

Modularity to Evaluate Community Quality

Modularity can also be used to evaluate the best level to cutoff of a hierarchical clustering



Applied Classification - 69

Apart from constructing the communities, the modularity measure can also be used to evaluate the quality of communities in hierarchical clustering. This can be done independently of how the clustering has been constructed. In fact, there is an optimal level of clustering when moving from one level of the hierarchy to the next. Initially increasing the number of communities increases their quality, by separating distinct communities. At a certain point, splitting of communities in smaller communities worsens the quality of the community structure. Thus there exists an optimum point of clustering that can be selected using modularity.

Louvain Modularity - Discussion

Widely used in social network analysis and beyond

- Method to extract communities from very large networks very fast

Complexity: O(n log n)

Louvain modularity clustering is today the method of choice for social network clustering, mainly because of its good computational efficiency. It runs in n log n, which makes it applicable for very large networks, as they occur today, in particular for social networks resulting from large platforms, as social network sites, messaging services or telephony.

Girvan-Newman Algorithm

Divisive Community Detection

- Based on a **betweenness measure** for edges, measuring how well they separate communities
- Decomposition of network by splitting along edges with highest separation capacity

Overall algorithm

- Repeat until no edges are left
  - Calculate betweenness of edges
  - Remove edges with highest betweenness
  - Resulting connected components are communities
- Results in hierarchical decomposition of network

We now introduce a second algorithm fro community detection, that belongs to the class of divisive algorithms. Also this algorithm is based on a measure, this time on a measure on edges. The betweenness measure gives an indication which edges are likely to connect different communities, and thus are good splitting points, to partition larger parts of the network in to communities. The algorithm recursively decomposes the network, by removing edges with the highest betweenness measure, till no edges are left. Also this algorithm results in a hierarchical clustering.

Edge Betweenness

Edge betweenness: fraction of number of shortest paths passing over the edge

$$betweenness(v) = \sum_{x,y} \frac{\sigma_{xy}(v)}{\sigma_{xy}}$$

where

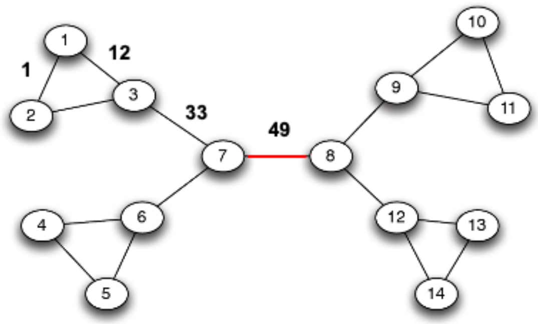
σ_{xy}: number of shortest paths from x to y

$\sigma_{xy}(v)$: number of shortest paths from x to y passing through v

Betweenness centrality is an indicator of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node. A node with high betweenness centrality has a large influence on the transfer of items through the network, under the assumption that item transfer follows the shortest paths. The concept finds wide application, including computer and social networks, biology, transport and scientific cooperation.

Alternatively, there exist also the concept of *random-walk betweenness*. A pair of nodes m and n are chosen at random. A walker starts at m, following each adjacent link with equal probability until it reaches n. Random walk betweenness x_{ij} is the probability that the link $i \rightarrow j$ was crossed by the walker after averaging over all possible choices for the starting nodes m and n

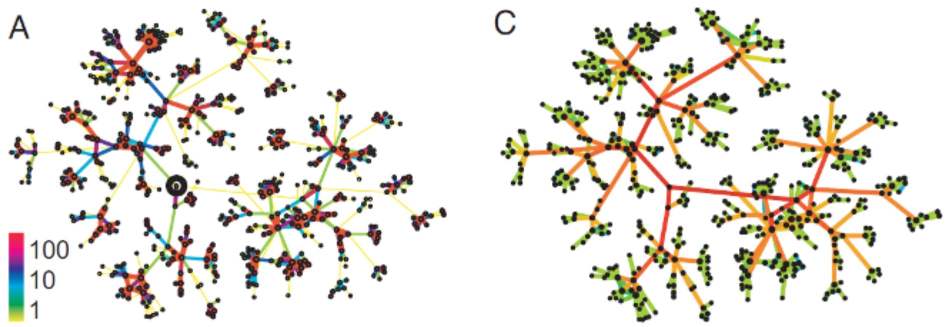
Example



$$\sigma_{xy}(v)$$

In this example network we compute betweenness for some selected edges. For example, for the edge 1-2 there is one shortest path between 1 and 2 that traverses the edge 1-2, thus the value is 1. For 1-3 there are shortest paths from the remaining 12 nodes in the network (except node 2) to node 1 that have to pass through this edge, thus the betweenness value is 12. For edge 3-4 we have paths going to both nodes 1 and 2, thus the betweenness value of that edge is significantly higher than for 1-3.

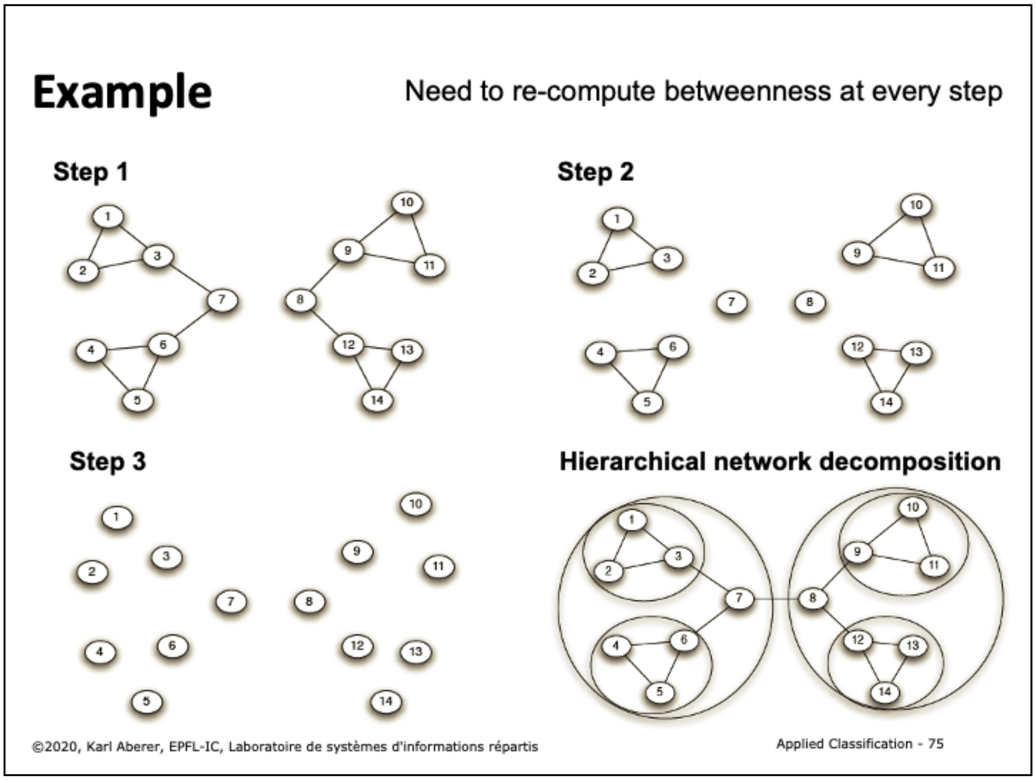
Underlying Intuition



Edge strengths (call volume) in a real network

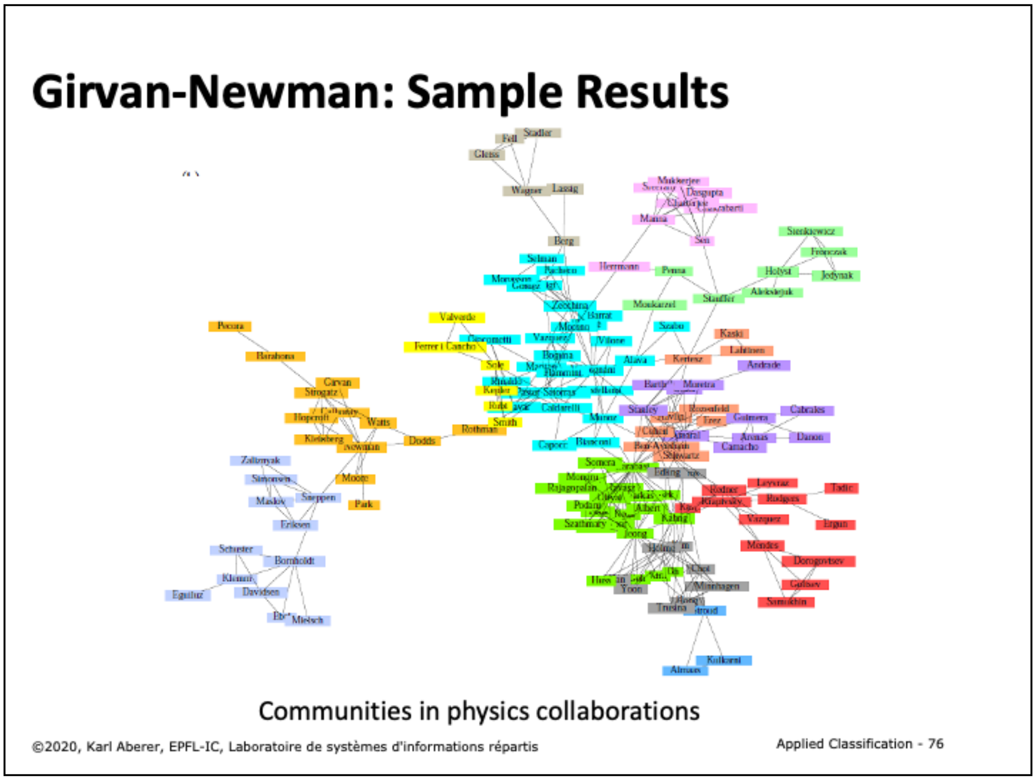
Edge betweenness in a real network

In a sense betweenness is the dual concept to connectivity. This is illustrated by the two graphs that result from real phone call networks. On the left hand side we see the indication of the strengths of connections among the nodes. Communities are tightly connected by such links. On the right hand side we see the betweenness measure. Now the links that are connecting communities have a high strength, since, intuitively speaking, the traffic from one community to another has to traverse over these sparsely available links.

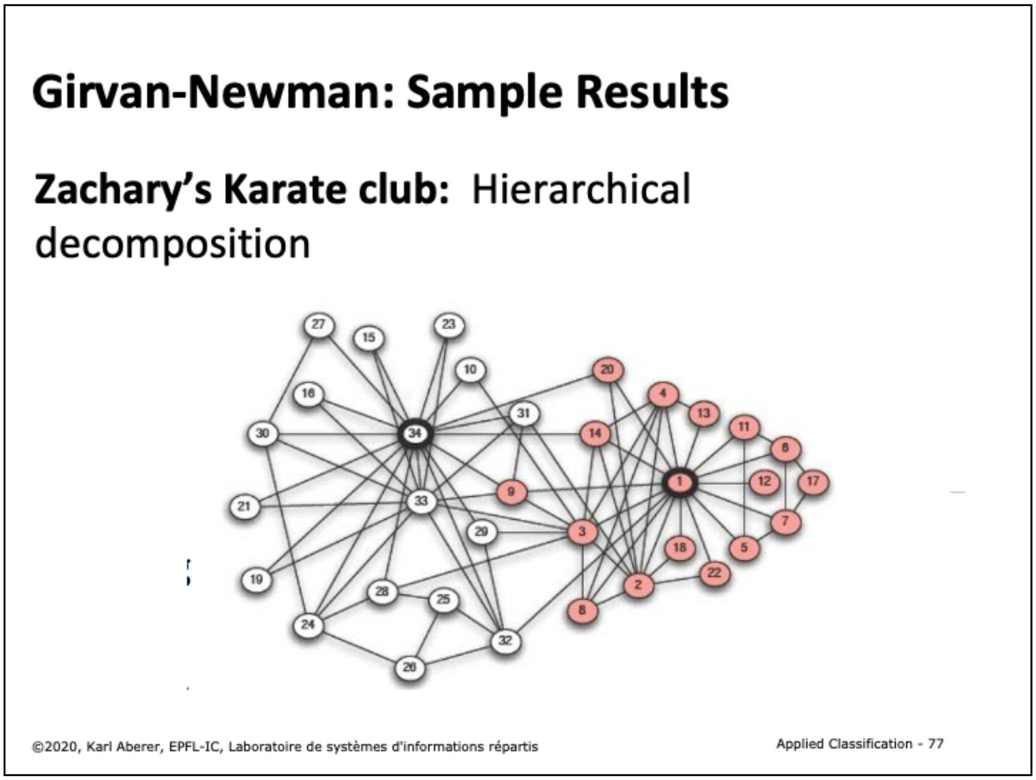


Here we illustrate the execution of the Girvan-Newman algorithm. In Step 1 we remove one edge (in the middle) that had the highest betweenness value, resulting in two communities. Next (by symmetry) the edges connected to nodes 7 and 8 are removed, and in the third and fourth step the network decomposes completely. By overlaying the communities that have resulted from each step we obtain the final hierarchical clustering.

As the graph structure changes in every step, the betweenness values have to be recomputed in every step. This constitutes the main cost of the algorithm.

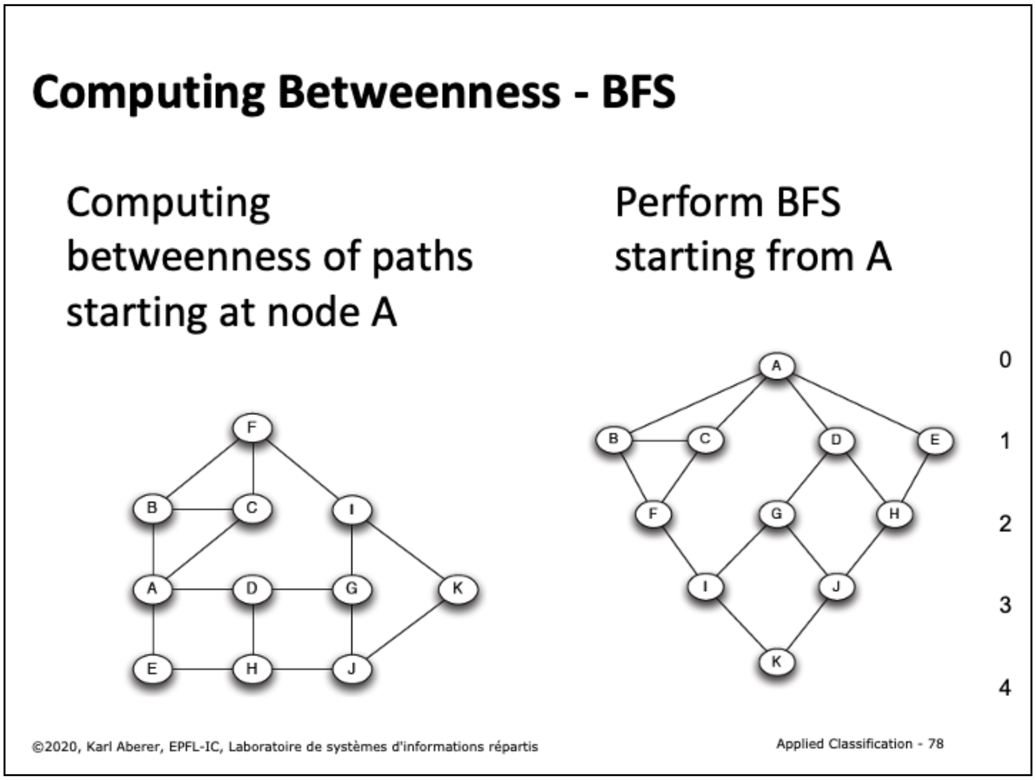


The algorithm has been applied in many contexts, in particular on smaller graphs resulting from social science studies.



The origins of the algorithm come from studying a network of the 34 karate club members studied by the sociologist Wayne Zachary. Links capture interactions between the club members outside the club. The circles and the squares denote the two fractions that emerged after the club split into two following a feud between the group's president and the coach. The split between the members closely follows the boundaries of these communities. This karate club has been historically used as a benchmark to test community finding algorithms.

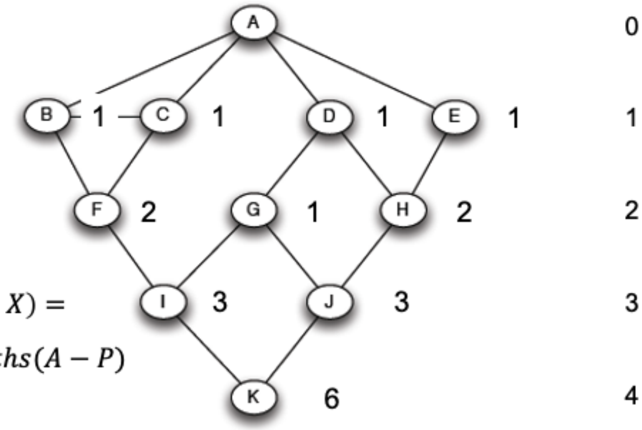
There was one outlier, node number 3, where the algorithm wrongly assigns the member. at the time of conflict, node 9 was completing a four-year quest to obtain a black belt, which he could only do with the instructor (node 34)



We describe now the process of computing the betweenness values. The approach is to perform a breadth-first search (BFS) for every node in the graph. The nodes are arranged in increasing levels of distance of the starting node, e.g. node A.

Computing Betweenness – Path Counting

Count the number of shortest paths from A to all other nodes of the network



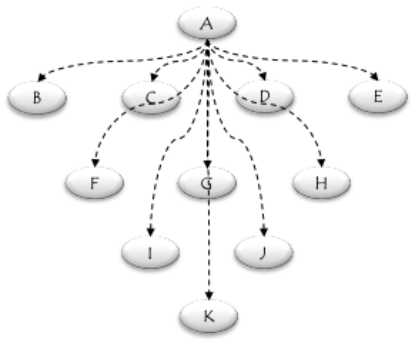
$$\#shortestpaths(A-X) = \sum_{P \; parent \; of \; X} \#shortestpaths(A-P)$$

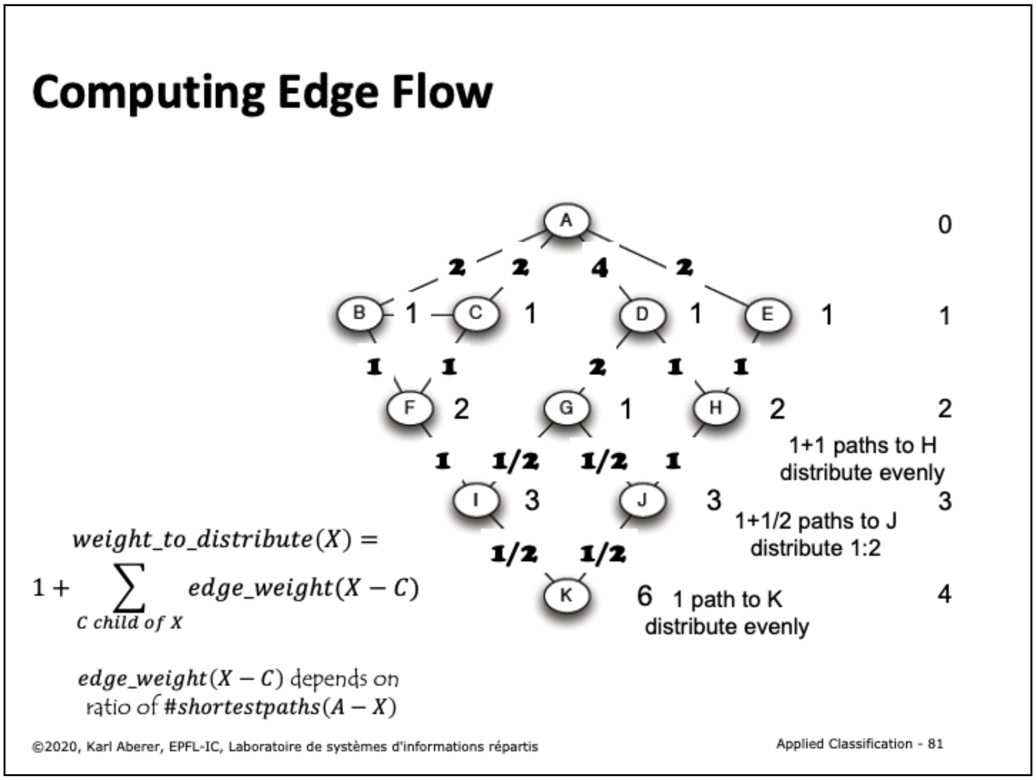
In a first phase we count the number of shortest paths that are leading to each node, starting from node A. To do so we can simply reuse the data that has been computed at the previous level, summing up the number of paths that have been leading to each parent of a given node.

Computing Betweenness – Edge Flow

Edge Flow

- 1 unit of flow from A to each node
- Flow to be distributed evenly over all paths
- Sum of the flows from all nodes equals the betweenness value





In a second phase we compute edge flow values (that are the betweenness values related to node A). We proceed in a bottom-up approach. The flow that arrives at every node is 1. In addition, the node receives also all the flows that are passing on to the children. Thus, the total flow weight a node receives (or has to pass through) is 1 plus the flows to all of its children. This weight is distributed over the parents, proportionally to the number of paths that are leading to those parents (what has been computed in phase 1).

This is a description of the initial steps in detail:

Node K: it receives a total flow of 1 from A. Since it has 2 incoming edges this flow is evenly distributed over the two incoming edges.

Node J: it receives a total flow of 1 from A plus transfers a flow of ½ to K. Thus the flow to distribute is: $1 + \sum_{C\ child\ of\ X} edgeweight(X - C)$ = 1.5. It's parent nodes G and H have a ratio of incoming shortest paths of 1:2, thus the flow is accordingly distributed over the incomgin edges, ½ and 1

Node H: it receives a total flow of 1 from A pls transfers a flow of 1 to J. This the flow do distributed is 2. The two parent nodes D and E have the same number of incoming shortest paths, this the flow is equally distributed.

The remaining steps proceed analogously.

Algorithm for Computing Betweenness

1. Build one BFS structure for each node
2. Determine edge flow values for each edge using the previous procedure
3. Sum up the flow values of each edge in all BFS structures to obtain betweenness value
   - Flows are computed between each pairs of nodes → final values divided by 2

Once the flows specific to a node have been computed for every node, the last step is to aggregate for each edge all the flow values that have been computed for all the nodes. In this way we compute each flow twice (flow into both direction), therefore, the final betweenness value corresponds to the aggregate flow value divided by 2.

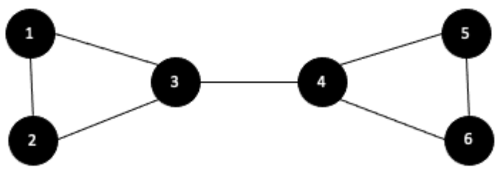
$\sigma_{xy}(v)$ of edge 3-4 is …

A. 16

B. 12

C. 9 Correct

D. 4



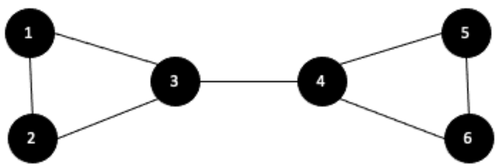
When computing path counts for node 1 with BFS, the count at 6 is …

A. 1 Correct

B. 2

C. 3

D. 4



Girvan-Newman Discussion

Classical method

- Works for smaller networks

Complexity

- Computation of betweenness for one link: $O(n^2)$
- Computation of betweenness for all links: $O(L\ n^2)$
- Sparse matrix: $O(n^3)$

The Girvan-Newman algorithm is the classical algorithm for community detection. It's major drawback is its scalability. The flow computation for one link has quadratic cost in the number of nodes (it has to be computed for each pair of nodes). If we assume sparse networks, where the number of links is of the same order as the number of nodes, the total cost is cubic. This was also one of the motivations the inspired the development of the modularity based community detection algorithm

References

Document classification

- Aggarwal, Charu C., and Cheng Xiang Zhai. "A survey of text classification algorithms." *Mining text data*. Springer US, 2012. 163-222.
- Jindal, Rajni, Ruchika Malhotra, and Abha Jain. "Techniques for text classification: Literature review and current trends." *Webology* 12.2 (2015): 1
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008 (http://www-nlp.stanford.edu/IR-book/) Chapter 13
- Fasttext: https://www.youtube.com/watch?v=CHcExDsDeHU

Recommender Systems

- Chapter 9 in mmds.org
- Recommender Systems Handbook, Springer 2015

References

Community detection

The slides are loosely based also on:

- http://barabasilab.neu.edu/courses/phys5116/

Papers

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

– Girvan, Michelle, and Mark EJ Newman. "Community structure in social and biological networks." *Proceedings of the national academy of sciences* 99.12 (2002): 7821-7826.