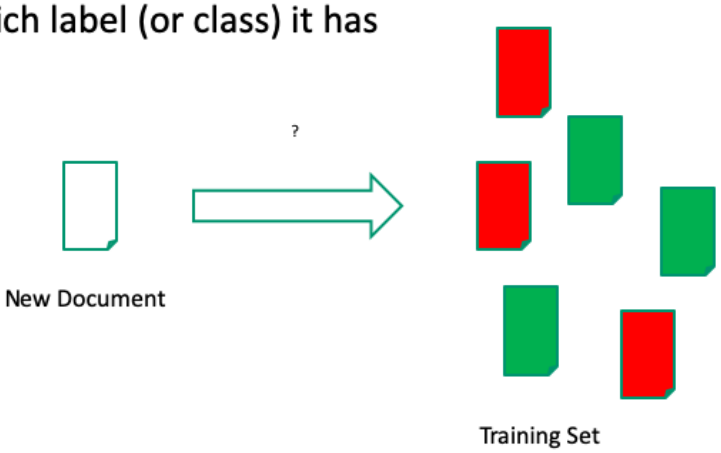
# 6. DOCUMENT CLASSIFICATION

## 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*:

More than 4000 visitors attended Scientastic the science festival of EPFL , organised in Valais for the first time

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

*Author*: @EPFL_SPS





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

## Challenge in Document Classification

The feature space is of very high dimension

- Vocabulary size
- All word bigrams
- All character trigrams
- etc.

Dealing with high dimensionality

- Feature selection, e.g., mutual information
- Dimensionality reduction, e.g., word embedding
- Classification algorithms that scale well




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.

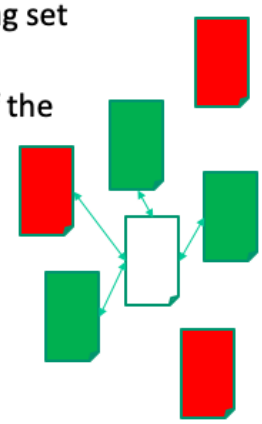
## Simple Method: k-Nearest-Neighbors (kNN)

**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: #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. For 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 the class is C for document D

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




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?

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

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.

## Fasttext

Classifier based on word embeddings

Extensions

- Using word n-grams (phrases)
- Subword information (character n-grams)

• Possible to build vectors for unseen words!

$$h_w = \sum_{g \in w} x_g$$




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 |

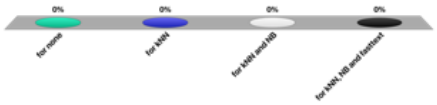
| Rank | Feature Selection Methods | # Papers |
|---|---|---|
| 1 | Chi-squared test(CHI) | 21 |
| 2 | Information Gain (IG) | 20 |
| 3 | Mutual Information(MI) | 16 |
| 4 | Latent Semantic Indexing (LSI), Singular Value Decomposition (SVD) | 10 |
| 5 | Document Frequency (DF) | 8 |
| 6 | Term Strength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



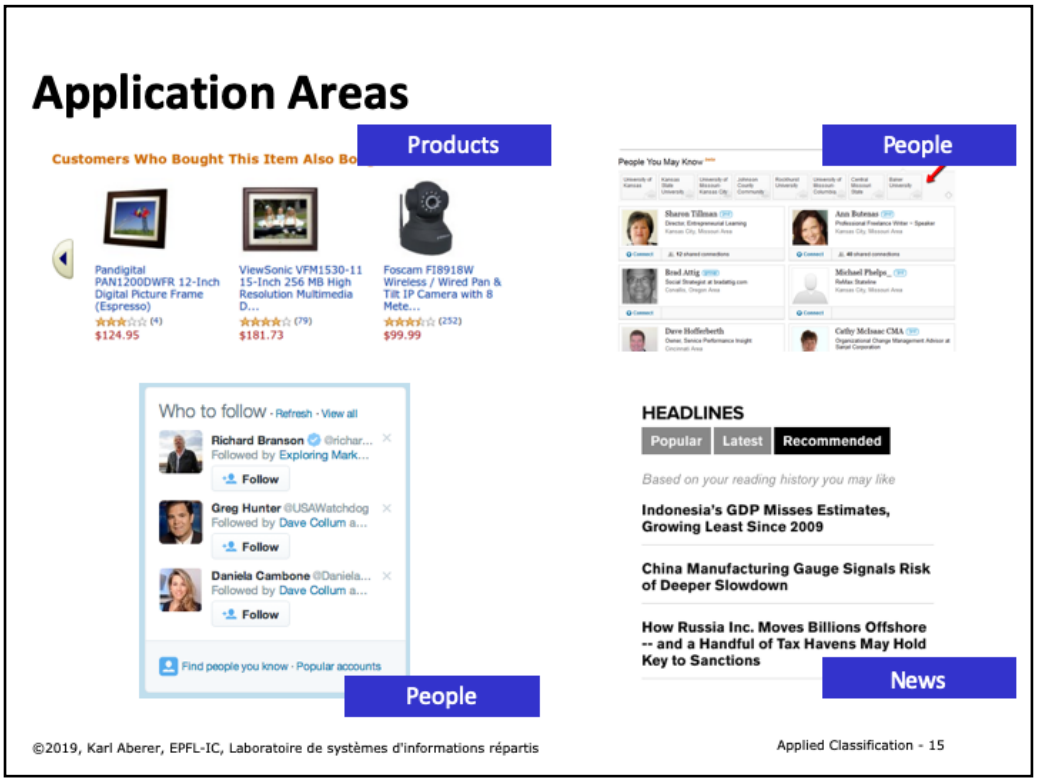

Document classification is an area of high, both because it is required in a growing number of application domains and because at the same time the classification methods are becoming increasingly powerful due to the availability of very large document corpuses and the growing computational power available. 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.

## The dimensionality of the feature space depends on the vocabulary size …

A. for none
B. for kNN
C. for kNN and NB
D. for kNN, NB and 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

## Definition: Recommender System

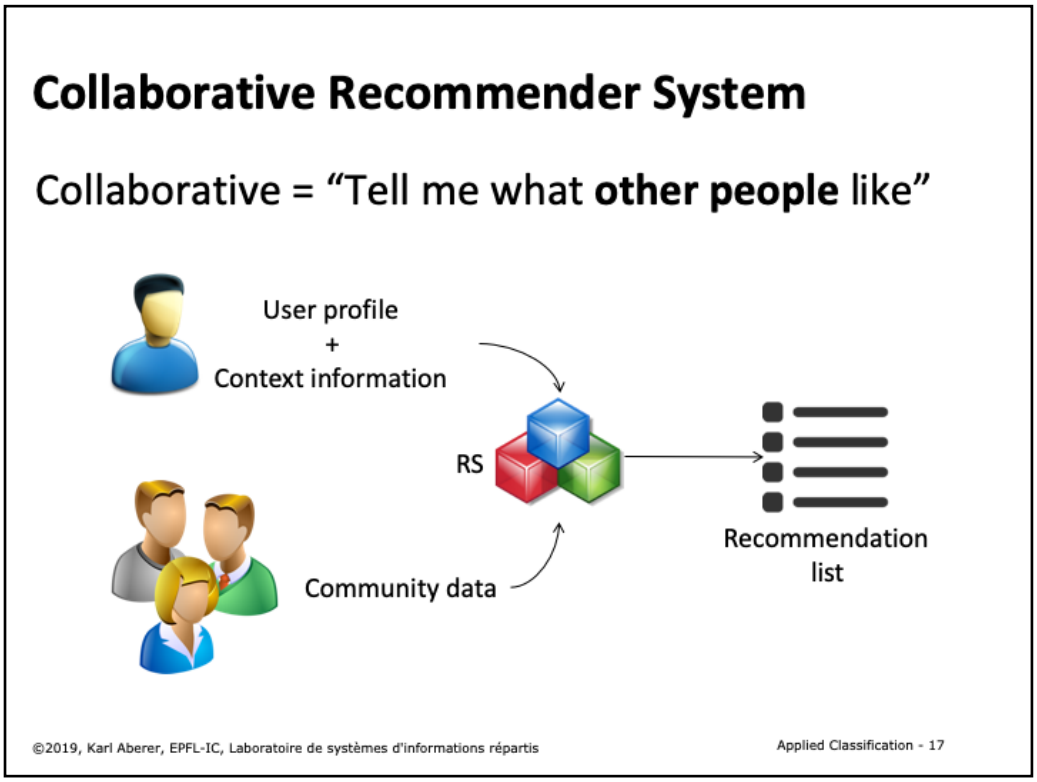
Given a *user* model and a set of *items*, a recommender system is a function that helps to match users with items by *ranking* the items in order of decreased *relevance*



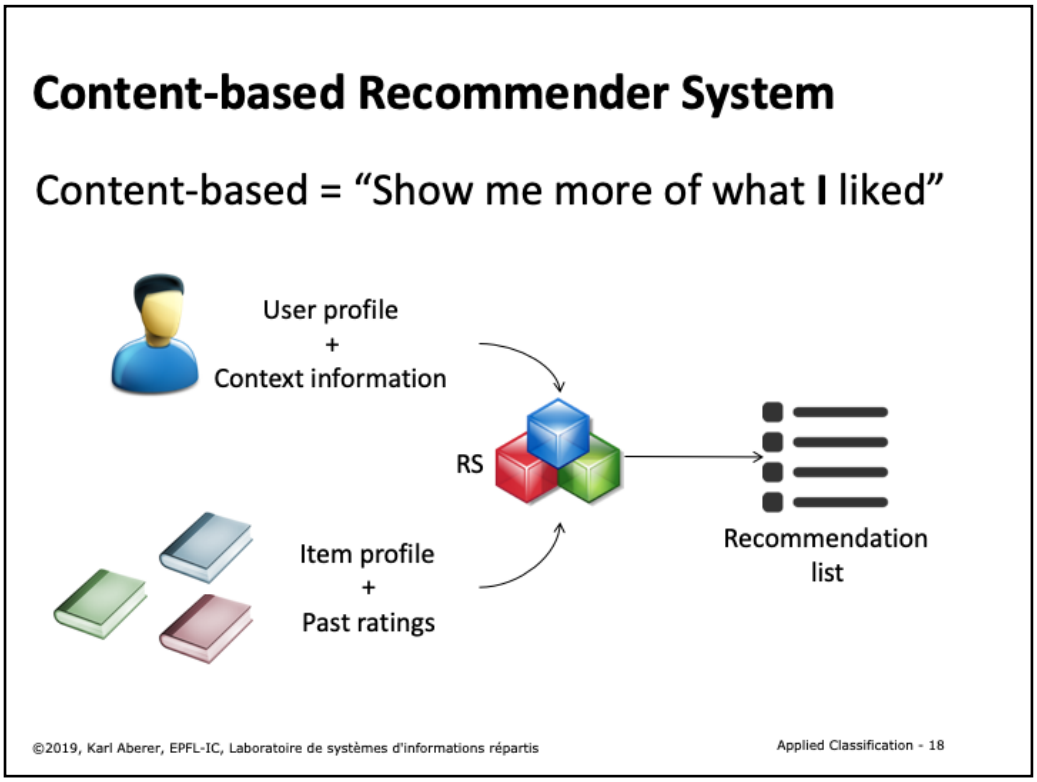


A recommender systems identifies for a given user a set of items that are relevant to her. This can result in a ranking of the items (modelling a relevance function) or a classification.

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. Find 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 $N_U(x)$

## Similarity Between Users

Pearson correlation coefficient

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

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 both $x$ 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

$$\bar{r}_U = 4$$

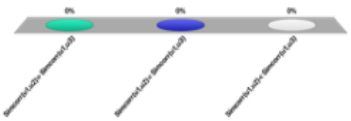
$$(r_{U1}(I5) - \bar{r}_{U1}) = 3 - \frac{12}{5} = \frac{3}{5}, (r_{U2}(I5) - \bar{r}_{U2}) = 5 - \frac{19}{5} = \frac{6}{5}$$

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

# Given 3 users with ratings…

U1: 1 3
U2: 2 4
U3: 1 4

A. $Sim_{corr}(u1,u2) > Sim_{corr}(u1,u3)$
B. $Sim_{corr}(u1,u2) = Sim_{corr}(u1,u3)$
C. $Sim_{corr}(u1,u2) < Sim_{corr}(u1,u3)$



## 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 $N_I$

## 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

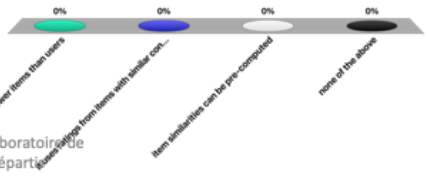
- Calculate all the pair-wise item similarities in advance
- Items similarities are more stable
- The neighbourhood $N_I(a)$ to be considered at runtime is small




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).

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



## Content-based Recommendation

Collaborative filtering needs only the ratings, it does not require any 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
- Content can be limited or impossible to extract (multimedia)
- Tends to recommend “more of the same”

More scalable: tf-idf of items can be computed offline

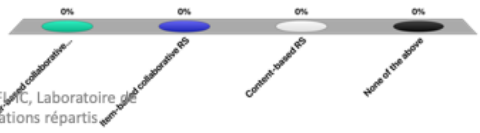



Content-based recommendation also suffers from the cold start problem as collaborative filtering, but only for the users that did not rate anything in the past. Another problem is that sometimes the information available to extract the content is limited (e.g., movie synopsis versus full plot) or impossible 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.

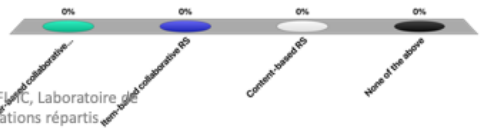
For a user that has not done any ratings, which method can make a prediction?

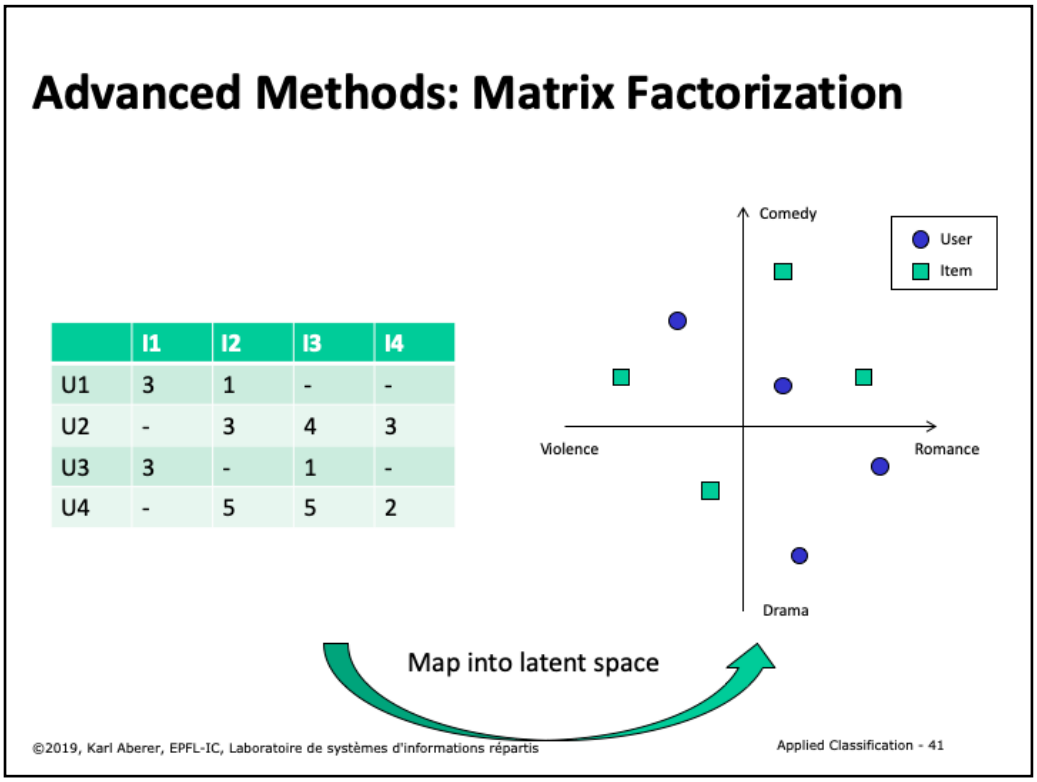
A. User-based collaborative RS
B. Item-based collaborative RS
C. Content-based RS
D. None of the above



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
D. None of the above






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

Users U and Items D

Matrix R of ratings: |U| x |D|

Goal: Decompose R (approximatively)

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

P is a |U| x K matrix: user features

Q is |D| x K matrix: item features

K latent factors

Problem: R has many undefined values!





Here we formulate the factorization problem.

## Approximation Error

For a given rating $r_{ij}$

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

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 $(u,i) \in M$. The minimization problem can be solved using stochastic gradient descent. To that end we first compute the gradients.

## Stochastic Gradient Descent

Update Rule

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

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

Repeat till error small

Needs only to be applied 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

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

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




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

# References

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.

# 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