Recommendation Systems

Dott. Salvatore Alaimo

Studio 32 – Blocco 2

E-Mail: alaimos@dmi.unict.it

Overview

- Introduction
- Content-based recommendations
- Collaborative Filtering
  - Collaborative Filtering in practice: SVD
- Graph-based methods
- Hybrid methods
- Results evaluation
- The DT-Hybrid Algorithm
- Recommendation and R

Some bibliographical references

- Almost all the topics which will be discussed here are available in the 9th chapter of the Ullman's book.
- Graph-based methods:
  - Zhou, T., Ren, J., Medo, M., and Zhang, Y. (2007). **Bipartite network projection and personal recommendation**. Physical Review E, 76(4), 046115–22.
  - Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R., and Zhang, Y.-C. (2010). **Solving the apparent diversity-accuracy dilemma of recommender systems**. Proceedings of the National Academy of Sciences, 107(10), 4511–4515.
- The DT-Hybrid Algorithm:
  - S. Alaimo, A. Pulvirenti, R. Giugno, and A. Ferro. **Drug–target interaction prediction through domain-tuned network-based inference.** Bioinformatics 2013 29: 2004-2008.
    - **http://alpha.dmi.unict.it/dtweb/dthybrid.php**
- R examples:
  - **http://alpha.dmi.unict.it/~alaimos/2014recommendationExamples.R**

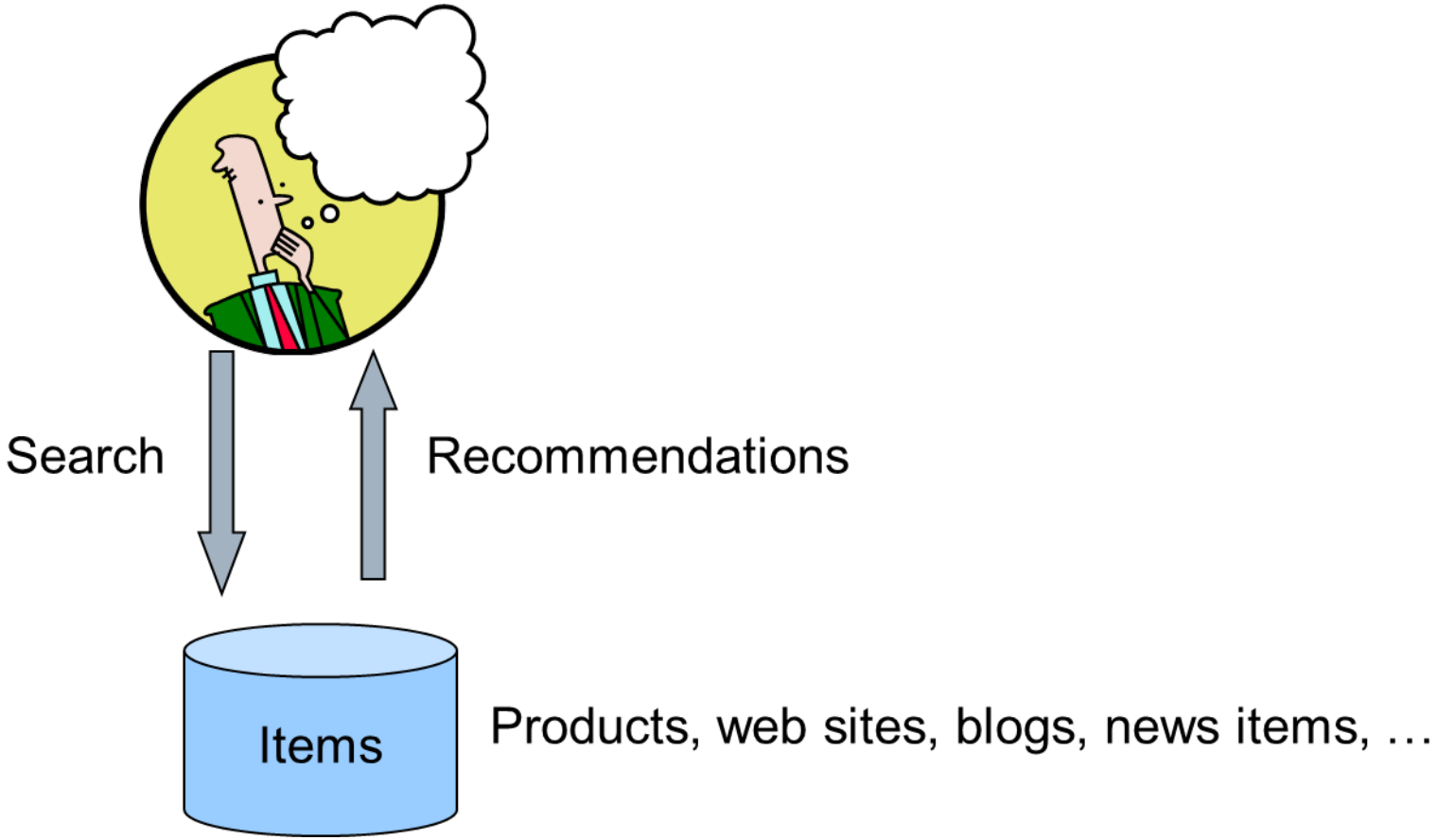
Part I

Introduction

Introduction

- A **recommendation system** is a class of applications (usually web-based) that involve predicting users responses based on their preferences.
- Two examples are:
  - Suggesting news articles to on-line newspaper readers (based on visited articles and their properties);
  - Offering customers of an e-commerce website suggestions about what they might like to buy (based on their purchase history and past searches).
- Many different technologies but two main groups:
  - **Content-based systems**: examine items' properties to recommend new items.
  - **Collaborative filtering systems**: use a similarity measure between users and/or items to recommend items. (Similar items or owned by similar users).

Introduction



Definition

- In a recommendation system there are **two classes of entities** which are grouped into two different sets:
  - A set of **Users** ($U=\{u_1,\ldots,u_n\}$)
  - A set of **Objects** ($O=\{o_1,\ldots, o_m\}$)
- Users have preferences that must be inferred from the data.
- These information are represented with an **utility matrix**.
- An **utility matrix** ($U_m=\{d_{uo}\}_{nxm}$) is a sparse matrix that for each user-object pair gives the degree of preference of that user for that object.
- Since the utility matrix is sparse, the **goal** of a recommendation system is to **predict the blanks in the utility matrix** to infer the preferences of an user.

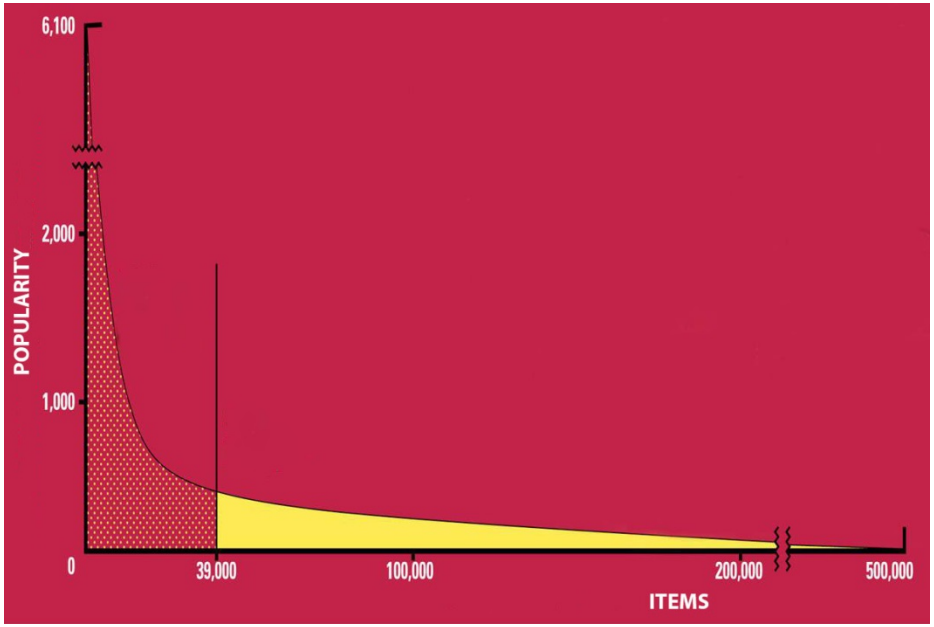
Example

	HP1	HP2	HP3	TW	SW 1	SW 2	SW 3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- This matrix represents users' ratings of movies on a **1-5** scale, with **5** the highest rating.
- **Blanks** represent the situation where the user has **not rated the movie**.
- We might design a system to take into account the properties of movies.
- For example user "A" does not like "SW1" so we might infer that he will not like movies "SW2" and "SW3".

Long tail phenomenon

- It explains why we need recommendation systems.
- Physical stores have limited shelf space so they can show only a small fraction of all the possible items that exists (the most popular ones).
- The web enables near-zero-cost dissemination of information about products so anything can be available to customers.
- Users have more choice but they might not know all available items.
- Recommendation engines solve this problem.



Types and applications

- The fields of application of these systems are many:
  - Product recommendations: Amazon or similar online vendors;
  - Movie recommendations: Netflix or YouTube;
  - News Articles: online newspapers or blogs.
- There are many types of recommender systems:
  - Editorial
  - Simple aggregates
    - Top 10, Most Popular, Recent Uploads
  - Tailored to individual users

Key Problems

- Gathering “known” ratings for matrix
- Extrapolate unknown ratings from known ratings
  - Mainly interested in high unknown ratings
- Evaluating extrapolation methods
  - We will not go into details about this right now.

Gathering Ratings

- Building an utility matrix is an hard task, but there are **two general approaches** to discover the values users place on items:
  - **Explicit** approach:
    - We can ask people to rate items (like some online stores or YouTube).
    - It is limited in its effectiveness, since users are generally unwilling to provide responses, and the information, gained from those who do, may be biased.
  - **Implicit** approach:
    - We can learn ratings from user actions.
    - e.g., purchasing an object or watching a movie should imply high rating
    - What about low ratings? we have only **two values** usually "**1**" when an user likes an object, "**0**" when we don't know anything about it.
    - We can not define an order for the ratings because they only indicate the presence (or absence) of information.

Extrapolating Utilities

- The main problem is that the **utility matrix is sparse**.
  - This is evident if we take into account what we said about the long tail phenomenon:
    - There are a few items evaluated by a large number of users, while there are many objects evaluated by a small group of users.
- We will describe **four main approaches**:
  - **Content-based**
  - **Collaborative Filtering**
  - **Graph-based**
  - **Hybrid Approach**
- At the end of our lesson we will describe an hybrid algorithm developed by us, which has found practical applications in areas other than those described before.

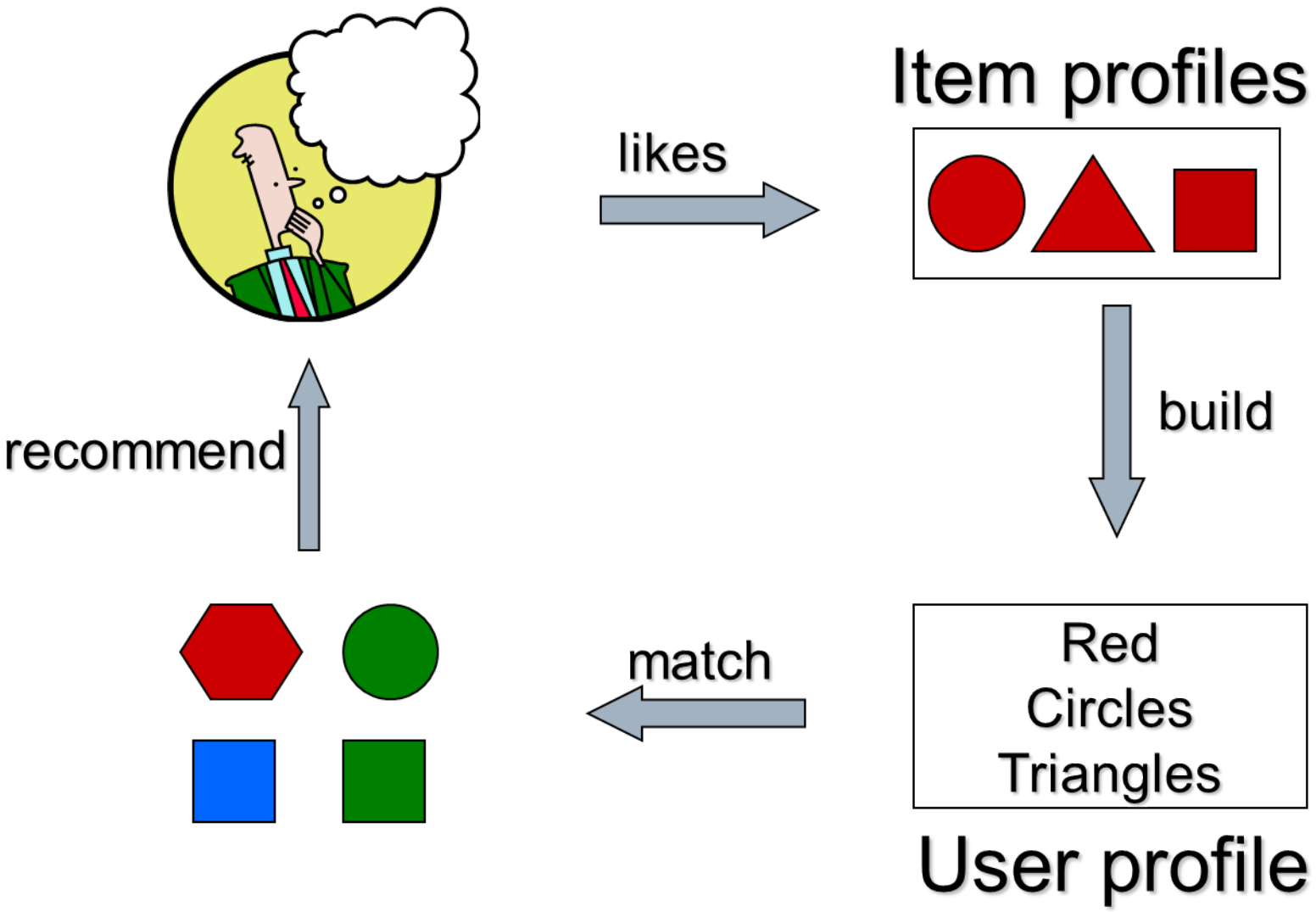
Part II

Content-based recommendations

Content-based recommendations

- **Main idea**: recommend items to user U similar to previous items rated highly by U
- The system focuses on **properties** of items.
  - The **similarity** between items is determined by measuring the similarity between their properties.
- Movie recommendations
  - recommend movies with same actor(s), director, genre, …
- Websites, blogs, news
  - recommend other sites with “similar” content

Plan of action



Item Profiles

- For each item, create an **item profile**.
- A **Profile** is a set of features (records representing important characteristics of an item).
  - movies: author, title, actor, director,…
  - text: set of "important" words in document
  - some features are easily discovered, while others are not.
- **Documents** are classes of objects for which it is not easy to define what features should be.
  - We may use the most important words of a document (words that make it possible to distinguish the arguments).
  - But, how to pick important words?
    - Usual heuristic is **TF.IDF** (Term Frequency times Inverse Doc Frequency)

TF.IDF

- Let "$\mathbf{f_{ij}}$" be the frequency of term "$\mathbf{t_i}$" in document "$\mathbf{d_j}$"
- We can define the frequency of a term "$\mathbf{TF_{ij}}$" as:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

- Let "$\mathbf{n_i}$" be the number of docs that mention term "**i**", and
- **N** the total number of docs, we can define the inverse doc frequency ($\mathbf{IDF_i}$) of term "**i**" as:

$$IDF_i = \log \frac{N}{n_i}$$

- The **TF.IDF score** is: "$\mathbf{w_{ij} = TF_{ij} \cdot IDF_i}$"
- After removing all **stop words** (the most common words that do not say much on the topic of a document) and calculated the scores, the **profile** of a document can be defined as the set of words that have the highest **TF.IDF score**, along with their score.

User Profiles

- With the information about the item profiles, we can create vectors (with the same components as the items' one) that describe user preferences.
  - We can use the utility matrix to find the connections between users and objects.
- To do so, we must distinguish two cases:
  - **Utility matrix with binary values**: average of the profile vectors' components for the objects that the user likes;
  - **Utility matrix with real values**: weight the profile vectors by the utility value normalizing them by subtracting the average value for a user.

Prediction

- After building profiles for users and objects, we must use a heuristic to predict profiles that may be recommended to the user.
- To do so, we can use any metric. A simple method is to use the cosine distance.
  - Given a user profile "**C**" and an item profile "**S**", we can estimate the distance as: $\mathbf{d(C,S) = cos(C, S) = (C \cdot S)/(||C|| \cdot ||S||)}$
- To complete the definition of the recommendation system, we now need a method which allows to find objects that have a maximum similarity (or minimal distance) with those of the user.

Model-based approaches

- Another approach is to treat the problem as one of machine learning.
- For each user, we learn a classifier that classifies items into two classes:
  - **liked by user** and **not liked by user**
  - e.g., Bayesian, regression, SVM, decision tree
- We now can apply this classifier to each item to find recommendation candidates.
- The main weakness of this approach is scalability:
  - Classifiers tend to take a long time to construct.
  - To build a classifier for each user, we need to look at all the item profiles.
  - Thus, this approach is used only for small problem sizes.
- We will not investigate this approach further.

Limitations of content-based approach

- Finding the appropriate features
  - e.g., images, movies, music
- Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
- Recommendations for new users
  - How to build a profile?

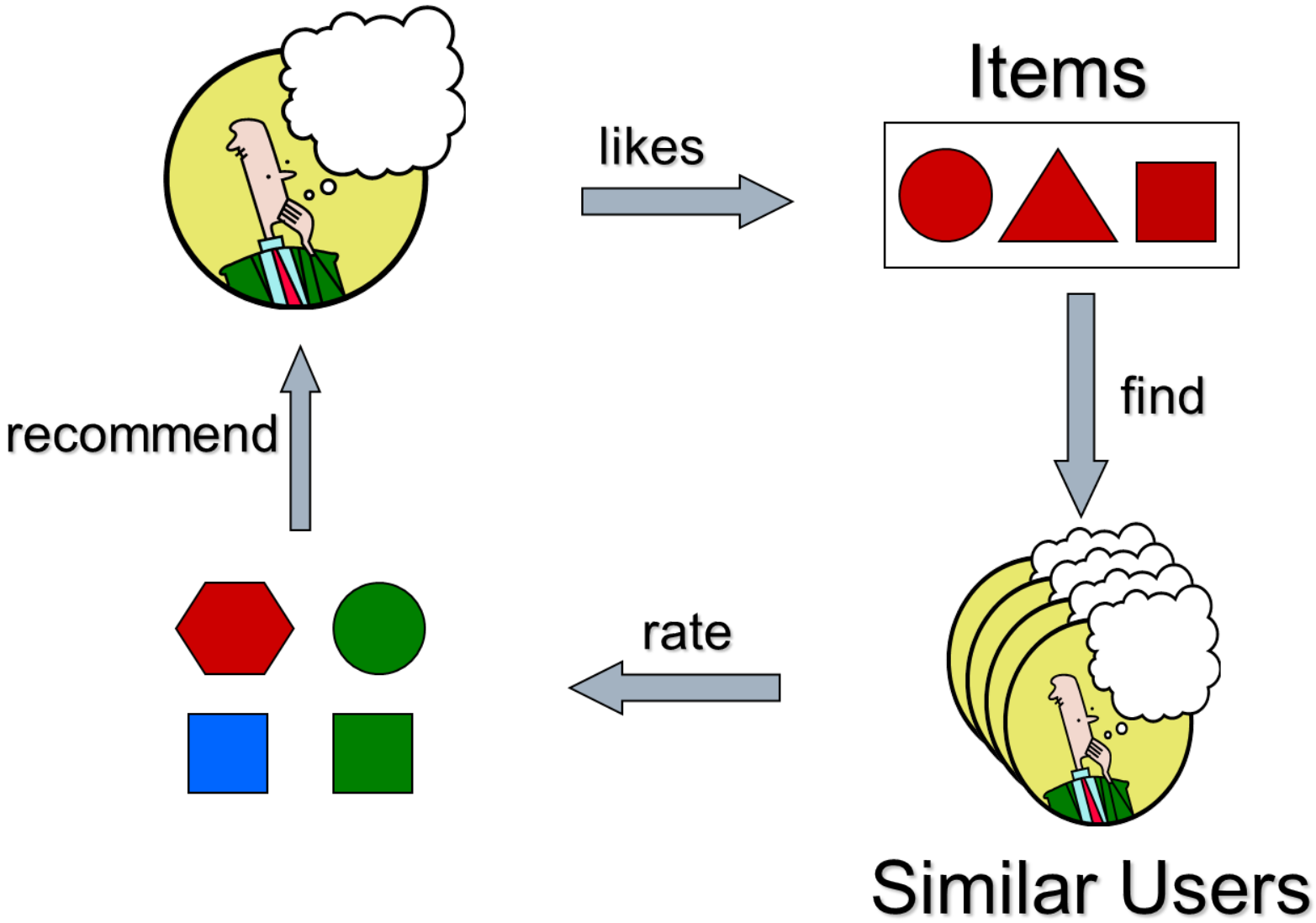
Part III

Collaborative Filtering

Collaborative Filtering

- It is a **significantly different approach** to recommendation.
- Instead of using items' features to determine their similarity, we focus on the similarity between user ratings for two items.
  - Column of the utility matrix instead of item-profile vector;
  - Row of the utility matrix instead of user-profile vector.
- **Users are similar if their vectors are close** according to some distance measure (eg. Jaccard or cosine distance).
- A recommendation for a user "**U**" is made of all the items that an **user similar to** "**U**" likes.

Plan of action



Plan of action

- Consider an user "**C**"
- Find set "**D**" of other users whose ratings are "**similar**" to C's ratings.
- Estimate new C's ratings based on ratings of users in "**D**"
- Build a list of recommended items based on the ratings that we have calculated.

Measuring Similarity

- Let "$\mathbf{r_x}$" be the vector of ratings of user/item "**x**" (the "**x-th**" row/column in the utility matrix).
- We can measure the similarity with another user/object "**y**" using any similarity measure. The most common are:
  - Jaccard similarity: $\mathbf{sim(x,y) = |r_x \cap r_y| / |r_x \cup r_y|}$
  - Cosine similarity: $\mathbf{sim(x,y) = cos(r_x, r_y)}$
  - Pearson correlation coefficient:
    - Let "$\mathbf{S_{xy}}$" be the set of the position of non-blank elements shared between "$\mathbf{r_x}$" and "$\mathbf{r_y}$"

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_x[s] - \overline{r_x})(r_y[s] - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_x[s] - \overline{r_x})^2 (r_y[s] - \overline{r_y})^2}}$$

Measuring Similarity

- Sometimes it is useful to normalize the utility matrix to obtain more accurate results.
- A simple and effective way, calculates the average rating for a user and subtracts it to all the user ratings.
  - This method turns low ratings into negative number, and high ratings into positive numbers.

Rating predictions

- Let
  - "D" be the set of "k" users most similar to "C" who have rated an item "s";
  - $\bar{r}_C$ be the average rating given to items of user "C".
- A **prediction function** could be:

$$r_{Cs} = \bar{r}_C + \frac{1}{|D|}\sum_{d \in D}(r_{ds} - \bar{r}_d)$$

- This prediction function, which uses the utility matrix normalization as shown previously, adjusts the estimate in the case that an user tends to give very high or very low ratings.

Item-Item Collaborative Filtering

- The procedure illustrated so far is called "**User-user collaborative filtering**".
- The procedure we describe now takes into account items similarity to compute new ratings instead of user ones.
  - For an item "**s**", we find "**k**" other similar items;
  - We can now estimate a rating for the item, based on the ratings shown in similar items using the same similarity measure and prediction function as in the user-user model.
- It has been shown that the Item-Item approach often works better than the User-User one because it is easier to find similar objects that similar users.

Pros and cons of collaborative filtering

- It works for any kind of item
  - No feature selection needed
- New user problem
- New item problem
- Sparsity of rating matrix

Part III-bis

Collaborative Filtering in practice:

SVD

Challenges

- **Sparsity**: No correlation between users can be found. Reduced coverage occurs.
- **Scalability**: Nearest neighbor algorithms computation time grows with the number of products and users.
- **Synonymy**

Dimensionality Reduction

- A possible solution to the problems listed above?
  - …Dimensionality Reduction
- Latent Semantic Indexing (LSI)
  - An efficient and fast algorithm was developed in the late 80s and early 90s
  - Addresses the problems of synonymy, sparsity, and scalability for large datasets.
  - Reduces dimensionality of a dataset and captures the latent relationships.
- Easily mapped to Collaborative Filtering!

LSI

- …is an indexing and retrieval method that uses SVD to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text.
- …is based on the principle that words that are used in the same contexts tend to have similar meanings.
- Key points:
  - Term-Document Matrix
  - Concepts Space
  - Mapping between:
    - Terms Concepts
    - Documents Concepts

LSI to CF

- Term-Document Matrix
- Concepts Space
- Mapping between:
  - Terms ↔ Concepts
  - Documents ↔ Concepts

- User-Item Matrix
- Categories Space
- Mapping between:
  - Items ↔ Categories
  - Users ↔ Categories

The Netflix Challenge

- …was an open competition for the best CF algorithm to predict user ratings for films, based on previous ratings without any other information.
- The grand prize: **US$1,000,000**
- The competition began in **2006**
- A team won the prize in **2009** (bested Netflix's own algorithm by 10.06%)
- The dataset:
  - ~**480,000** users and ~**18,000** movies; only ~**100** million ratings
  - the user-item matrix is **99%** sparse! (About **8.5 billion** potential ratings)

SVD – The math behind LSI

- For any **M x N** matrix **A** of rank **r**, it can be decomposed as:
  - $A_K = U_K \cdot D_K \cdot V_K^T$
- **U** is a **M x K** orthogonal matrix.
- **V** is a **N x K** orthogonal matrix.
- **D** is a **K x K** diagonal matrix whose entries are the first **k** non-zero singular values of **A**.
  - $\mathbf{d_1 \geq d_2 \ldots \geq d_K > 0}$
- $\mathbf{A_k}$ is the closest approximation to **A**
  - $\mathbf{A_k}$ minimizes the Frobenius norm over all **rank-k** matrices

$$\|A - A_K\|_F$$

Making Recommendations

- **Cosine Similarity**: common way to find neighborhood.
- Thanks to SVD, we can also make predictions of products with a simple dot product.
  - Given a user and the vector of its objects (**p**) on which to make predictions:
    - We can map from the original space to that of the categories with:

$$\bar{p} = p \cdot V_K$$

    - We can compute the recommendation for the user with:

$$R = \bar{p} \cdot V_K^T$$

Challenges with SVD

- **Scalability:** Once again, compute time grows with the number of users and products. $O(m^3)$
  - Offline stage.
  - Online stage.
- Even doing the SVD computation offline is not possible for large datasets.
- Approximation methods are needed! (IRLBA)

Part IV

Graph-based methods

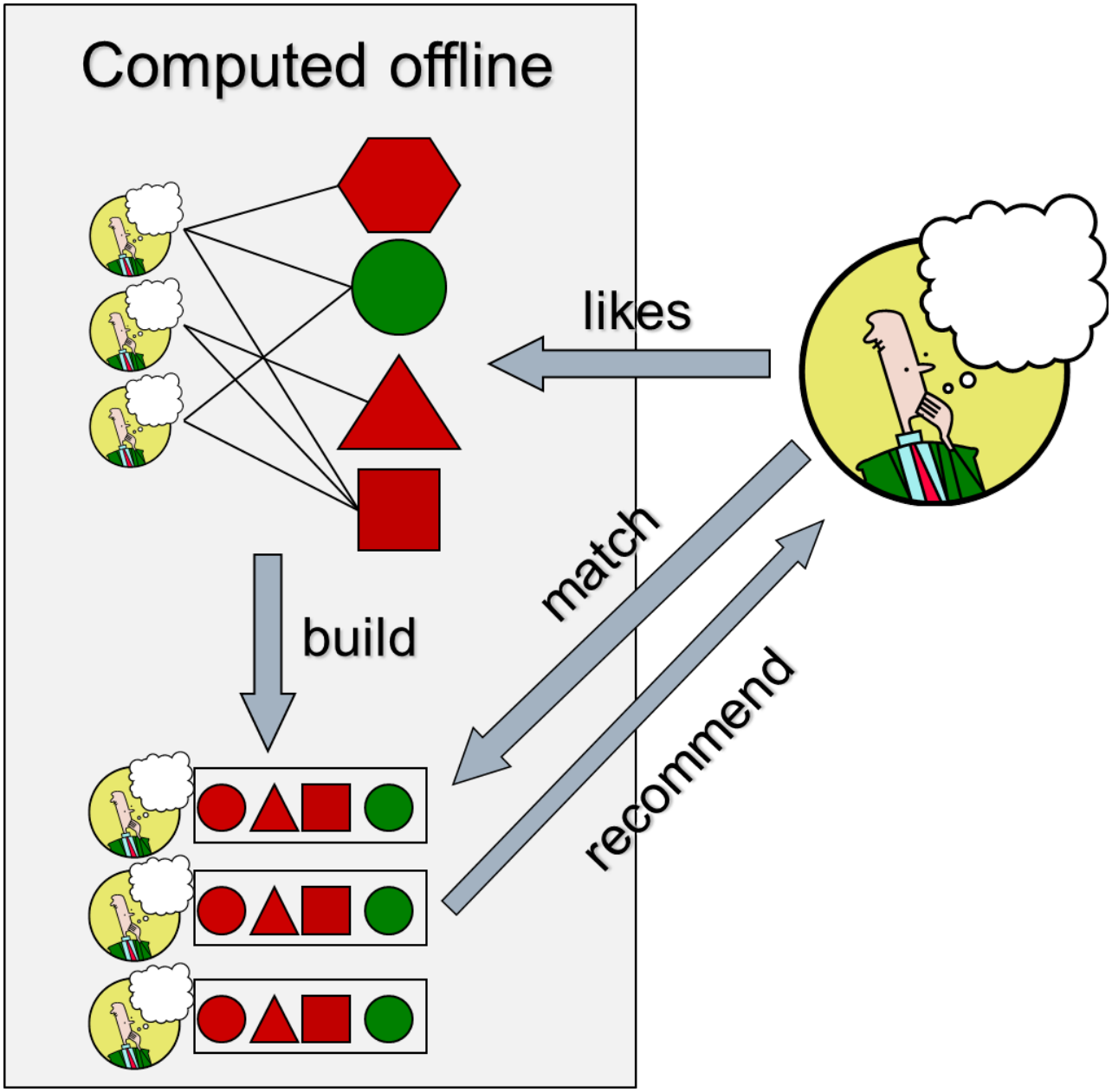
Graph-based methods

- It is similar to collaborative filtering but it uses a bipartite graph to store information.
- Since a bipartite graph is used, recommendations are obtained by inferring characteristics of the network's structure.
- As with the previously described systems, we have two classes of entities: **users** (**U**) and **objects** (**O**). Each of these two entities represents a class of bipartite network nodes.
- The bipartite graph is defined as follows:
  - **G(U, O, E, W)**
  - **E = {e_{ij} : u_i likes o_j)**
  - **W : E** $\rightarrow \Re$
  - where "**E**" is the set of edges (each link means that an user likes an object), and "**W**" is a function that represents the degree of preference of a user for an object.

Graph-based methods

- Therefore, we can define an adjacency matrix $\mathbf{A=\{a_{ji}\}_{mxn}}$ that contains in each position $\mathbf{a_{ji}}$ the value **w(i, j)** if user $\mathbf{u_i}$ likes object $\mathbf{o_j}$, **0** otherwise.
- Typically, a graph-based method do not take into account the degree of preference, therefore an element of the adjacency matrix contains either 1 (user likes object) or 0 (blanks).
- The result of a graph-based method is a set

Plan of action



NBI

- NBI (**Network-based inference**) is a method developed in 2007.
- It uses the **bipartite network projection technique** to obtain information on the network.
- The projection transforms a bipartite graph into a new graph where:
  - the nodes are all of the same type (users or objects);
  - two nodes in the projection are connected if there is at least a node (of different type) connected to both;
- The weights of the arcs provide information on the interactions (e.g. number).
- The projection allows us to **compress** the information contained in a bipartite network.

NBI

- The algorithm is based on the idea of a flow of resources through the bipartite network:
  - An initial amount of resource is assigned to the objects;
  - In a two-step process the resource is transferred first from the objects to users and subsequently transferred back to the objects;
  - This process together with a normalization procedure allows us to obtain scores for each user-object pair.
- The **NBI** algorithm is characterized by a **uniform distribution** of the resource from objects to users.

HeatS

- The algorithm **HeatS** is a further recommendation algorithm based on bipartite network.
- It uses the same projection technique and transfer of resources defined for **NBI**.
- The only difference is in the process of resources distribution.
- While **NBI** evenly distributes resources, **HeatS** distributes them so that each object receives a resource quantity equal to the average of the neighbors.

NBI & HeatS

- The result of the process described above corresponds to an **Object-projection** of the bipartite network (graph where all nodes are objects).
- A projection weight "$\mathbf{w_{ij}}$" corresponds to how much resource object "**j**" moved to object "**i**", or how it is likely that if a user likes object "**i**" than he will like object "**j**" .
- Given the adjacency matrix "**A**" of the bipartite network and the weight matrix "**W**", the recommendation matrix "**R**" for all users can be calculated in a single step as:
  - **R=W·A**

NBI & HeatS

- The calculation of the weights by means of resource transfer can be performed through the following equation:

$$w_{ij} = \frac{1}{\Gamma(i,j)} \sum_{l=1}^{n} \frac{a_{il} a_{jl}}{k(u_l)}$$

- where the Γ**(i, j)** function is defined as follows:
  - Γ**(i, j)=D(o_j)** for **NBI**
  - Γ**(i, j)=D(o_i)** for **HeatS**
- and **D(t)** is the degree of node "**t**" in the bipartite network.

Pros and cons of graph-based approach

- Like collaborative filtering, It works for any kind of item
- It solves the problem of sparse matrices considering the entire network for the calculation of each entry of the utility matrix.
- New user problem
- New item problem
- The calculation of the recommendations may require significant computational resources.

Part V

Hybrid methods

Hybrid methods

- The method is to implement two separate recommendation systems and combine the predictions in order to obtain a more robust system.
- Suppose we have two methods "**X**" and "**Y**" that report scores "$\mathbf{x_a}$" and "$\mathbf{y_a}$" respectively. An hybrid score for object a can be given by:

$$z_a = (1-\lambda)\frac{x_a}{\max\limits_{\beta} x_\beta} + \lambda\frac{y_a}{\max\limits_{\beta} y_\beta}$$

- where the normalizations address the fact that different methods may produce scores on different scales.
- By varying the parameter **$\lambda \in [0;1]$** we can tune the **hybrid X+Y** to favor the characteristics of one method or the other.

Hybrid methods

- First example:
  - Add content-based methods to collaborative filtering
    - item profiles for new item problem
- Second example:
  - Combine NBI and HeatS algorithms:
    - The two algorithms are particularly suitable to be combined.
    - Appropriately altering the $\Gamma(i, j)$ function, it is possible to build a hybrid of the two algorithms without having to calculate two different score.
    - The hybrid Γ function is:
      - $\Gamma(i, j) = D(o_i)^{1-\lambda} D(o_j)^{\lambda}$
    - the parameter $\lambda \in [0;1]$ is the tuning parameter.

Pros and cons of hybrid approach

- It allows to solve the weaknesses of a method combining it with another.
- The calculation of the combined score requires the evaluation of two different methods doubling computation time (in the best case).

Part VI

Results Evaluation

Evaluating Predictions

- Compare predictions with known ratings
  - Root-mean-square error (RMSE)

$$RMSE(\hat{Y}, Y) = \sqrt{\frac{\sum_{i=1}^{|\hat{Y}|}(\hat{Y}_i - Y_i)^2}{|\hat{Y}|}}$$

- Another approach: 0/1 model
  - Coverage
    - Number of items/users for which system can make predictions
  - Precision
    - Accuracy of predictions
  - Receiver operating characteristic (ROC)
    - Tradeoff curve between false positives and false negatives

Evaluating Predictions

- A further method for checking the quality of the predictions consists of four metrics developed to evaluate different aspects of the recommendation algorithm.
- To calculate these metrics, a cross-validation test should be used.
- They four metrics are:
  - **Recovery** (**r**): evaluates the score assigned to deleted links and the ability of the algorithm to recover them;
  - **Precision and Recall Enhancement** ($\mathbf{e_P}$, $\mathbf{e_R}$): evaluate the algorithm in terms of precision and recall, comparing the results with a null model;
  - **Customization** (**h**): measures the uniqueness of users' recommendation lists;
  - **Surprisal/Novelty** (**I**): measures the ability of the algorithm to generate new and unexpected results.
- We will not go into more detail on these metrics. More information can be found in the articles listed at the beginning of this presentation.

Evaluating Predictions

- **Recovery** (**r**): for an user **i** with **k(i)** items and its **p-th** predicted object α (given n objects), it is possible to calculate the relative rank by the expression $\mathbf{r_{\alpha,i}=p/[n-k(i)]}$, which should be smaller for interactions in the test set. The average of such values for the entire test set is **r**.
- **Customization** (**h**): given two users **i** and **j**, it is possible to define their inter-list distance: $\mathbf{h_{ij}(L)=1-(q_{ij}(L)/L)}$ where $\mathbf{q_{ij}(L)}$ is the number of common interactions in the **top-L** places. The average value of such pairwise distances with at least one link in the test set is our index of uniqueness.
- **Surprisal/Novelty** (**I**): Given an object **o**, the probability that it is connected to an user is **k(o)/n**. Self-information can be defined as $\mathbf{I_o=\log_2(n/k(o))}$. The average of these values for the top-L positions in the recommendation list for all users in the test set measures the surprisal.

Evaluating Predictions

- **Precision and Recall Enhancement** ($\mathbf{e_P}$, $\mathbf{e_R}$): Quality is measured in terms of the top-L elements in the recommendation list of each user. Let **Di** be the number of deleted interactions recovered for user **i**, and let **Di(L)** be its position in the **top-L** places of **i's** recommendation list. The average precision and recall for the prediction process can be computed as follows:

$$P(L) = \frac{1}{m'} \sum_{i=1}^{m'} \frac{D_i(L)}{L},$$

$$R(L) = \frac{1}{m'} \sum_{i=1}^{m'} \frac{D_i(L)}{D_i},$$

  - m′ is the number of users with at least one deleted link.
- A better perspective can be obtained by considering these values within random models.
  - $\mathbf{P_{rand}(L) = D/(n*m)}$
    - where **D** is the number of links in the test set
  - $\mathbf{R_{rand}(L) = L/n}$
- Given these random models, it is possible to compute the precision and recall enhancement as follows:
  - $\mathbf{e_P(L) = P(L)/ P_{rand}(L)}$
  - $\mathbf{e_R(L) = R(L)/R_{rand}(L)}$

Part VII

The DT-Hybrid algorithm

DT-Hybrid

- Detecting and verifying new connections among drugs and targets is a costly process.
- Historical point of view: development of compounds acting against particular families of proteins.
- Drugs act by binding to specific proteins, hence changing their biochemical or biophysical activities.
- Since proteins operate as part of highly interconnected cellular networks the ”one gene, one drug, one disease” paradigm has been challenged in many cases.
- For this reason, the concept of polypharmacology has raised for those drugs acting on multiple targets rather than single one.
- Many interactions are unknown and, given the significant amount of resources needed for in situ experimentation, we need algorithmic methodologies to predict new and significant relationships (DTI problem).

DT-Hybrid

- It is a graph-based hybrid method that has been designed for applications other than the classic ones.
- In particular, thanks to the introduction of additional information, it can be applied to bioinformatics data.
- We have two classes of entities: **drugs** (**D**) and **targets** (**T**).
- For each possible pair of drugs or targets, we compute a similarity measure based on prior biological knowledge.
- The problem is defined as follows:
  - **G(D, T, E)**
  - **E = {e_{ij} : u_i interacts with o_j)**
  - **S_D : D × D → ℜ ; S_T : T × T → ℜ**
  - where "**E**" is the set of edges (each link means that a drug interacts with a target), and "**S_D**" and "**S_T**" are two function that represents the degree of similarity between drugs or targets.

DT-Hybrid

- Now we can define:
  - an adjacency matrix $\mathbf{A=\{a_{ji}\}_{mxn}}$ that contains in each position $\mathbf{a_{ji}}$ the value **1** if drug "**i**" interacts with target "**j**", **0** otherwise.
  - a target similarity matrix $\mathbf{S=\{s_{ij}\}_{mxm}}$ that contains in each position $\mathbf{s_{ij}}$ the value $\mathbf{S_T(i,j)}$ which has been normalized so that $\mathbf{s_{ij} \in [0;1]}$.
  - a drug similarity matrix $\mathbf{S_1=\{s'_{ij}\}_{nxn}}$ that contains in each position $\mathbf{s'_{ij}}$ the value $\mathbf{S_D(i,j)}$ which has been normalized so that $\mathbf{s_{ij} \in [0;1]}$.
- The result of the algorithm is a matrix of scores:

$\mathbf{R=\{r_{ji}\}_{mxn}}$

DT-Hybrid

- Using the matrices we just defined, it is possible to build a global similarity matrix $\bar{S} = \{\bar{s}_{ij}\}_{mxm}$ in the following way:

1. Compute $\mathbf{S_2=\{s''_{ij}\}_{mxm}}$ where:

$$s''_{ij} = \frac{\sum_{k=1}^{n}\sum_{l=1}^{n}(a_{il}a_{jk}s'_{lk})}{\sum_{k=1}^{n}\sum_{l=1}^{n}(a_{il}a_{jk})}$$

2. Compute

$$\bar{S} = \alpha S + (1-\alpha)S_2$$

DT-Hybrid

- Like the other graph-based methods, DT-Hybrid computes an **Object-projection** (**target-projection**) of the bipartite network.
- A projection weight "$\mathbf{w_{ij}}$" corresponds to how likely it is that if a drug binds target "**i**" than it will bind target "**j**"
- The weights are guided by the similarity measures: greater similarity corresponds to greater weight, while less similarity to less weight.
- Given the adjacency matrix "**A**" of the bipartite network and the weight matrix "**W**", the recommendation matrix "**R**" for all drugs can be calculated in a single step as:
  - $\mathbf{R=W \cdot A}$

DT-Hybrid

- The calculation of the weights by means of resource transfer can be performed through the following equation:

$$w_{ij} = \frac{1}{\Gamma(i,j)} \sum_{l=1}^{n} \frac{a_{il} a_{jl}}{k(u_l)}$$

- where the **Γ(i, j)** function is defined as follows:

$$\Gamma(i,j) = \frac{D(o_i)^{1-\lambda} D(o_j)^{\lambda}}{\bar{s}_{ij}}$$

- and **D(t)** is the degree of node "**t**" in the bipartite network.

DT-Hybrid - Benchmarks

- We evaluated our method using four datasets containing experimentally verified interactions between Drugs and genes/proteins.
- The datasets were built by grouping all possible interactions between genes/proteins and drugs (DTI) based on their main gene types:
  - enzymes;
  - ion channels;
  - GPCRs;
  - nuclear receptors.
- We used the following similarity measures:
  - SIMCOMP 2D chemical similarity to compute similarity between drugs;
  - Smith-Waterman sequence similarity to compute similarity between genes.
- We used the following normalization procedures to build the matrices:

$$S_{norm}(i,j) = \frac{S(i,j)}{\sqrt{S(i,i) \cdot S(j,j)}}$$

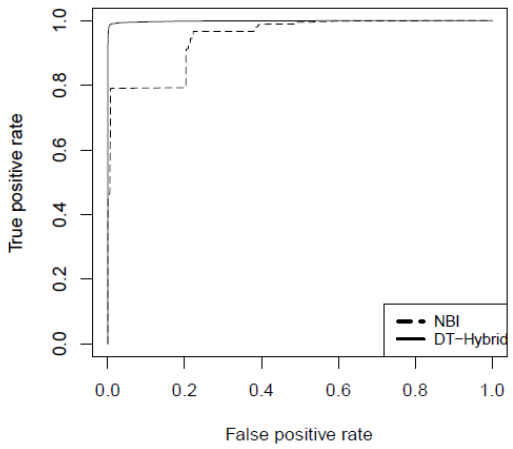
DT-Hybrid - Benchmarks

Dataset	Drugs	Targets	Interactions	Sparsity
Enzymes	445	664	2926	0,0099
Ion Channels	210	204	1476	0,0344
GPCRs	223	95	635	0,0299
Nuclear Receptors	54	26	90	0,0641
Complete	4398	3784	12446	0,0007

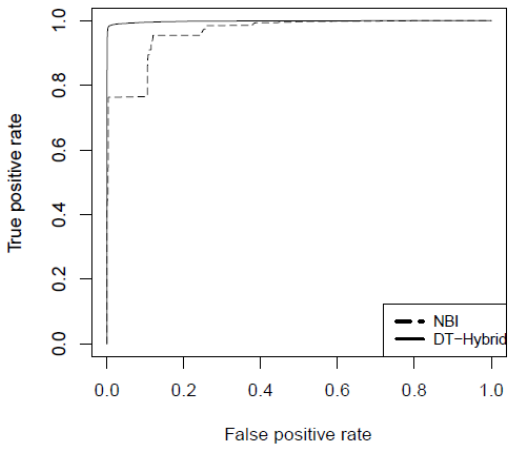
DT-Hybrid - Benchmarks

Data set	e_P			e_R			AUC		
	NBI	Hybrid	DT-Hybrid	NBI	Hybrid	DT-Hybrid	NBI	Hybrid	DT-Hybrid
Enzymes	115.5	189.9	**194.9**	21.1	**31.3**	**31.3**	0.9808±0.0009	0.9988±0.0003	**0.9993±0.0002**
Ion Channels	32.1	**32.4**	**32.4**	9.7	9.7	**9.9**	0.9857±0.0084	**0.9965±0.0022**	0.9964±0.0044
GPCRs	35.7	35.6	**37.6**	**4.6**	4.5	**4.6**	0.9734±0.0024	0.9970±0.0017	**0.9981±0.0008**
Nuclear Receptors	**70.2**	**70.2**	**70.2**	**1.3**	**1.3**	**1.3**	0.9930±0.0010	**0.9993±0.0069**	0.9992±0.0030

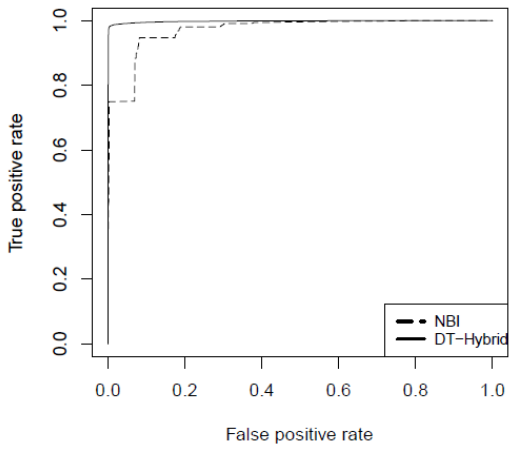
DT-Hybrid - Benchmarks



(a) $L = 10$

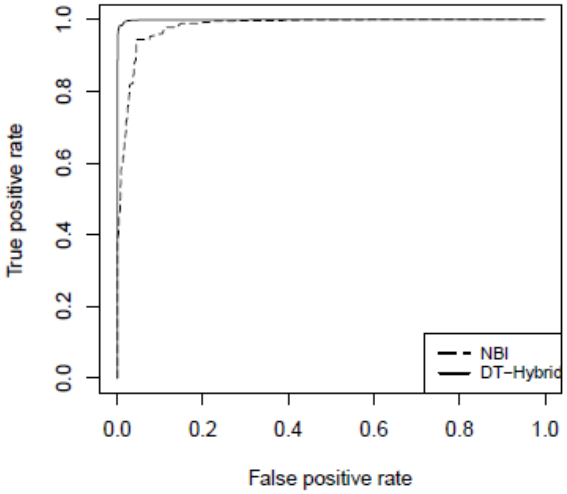


(b) $L = 20$

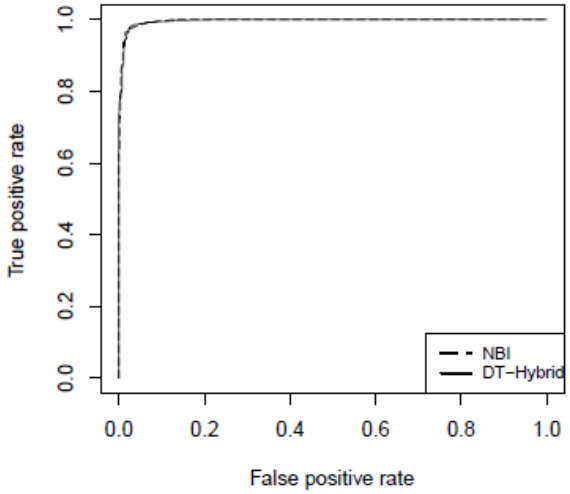


(c) $L = 30$

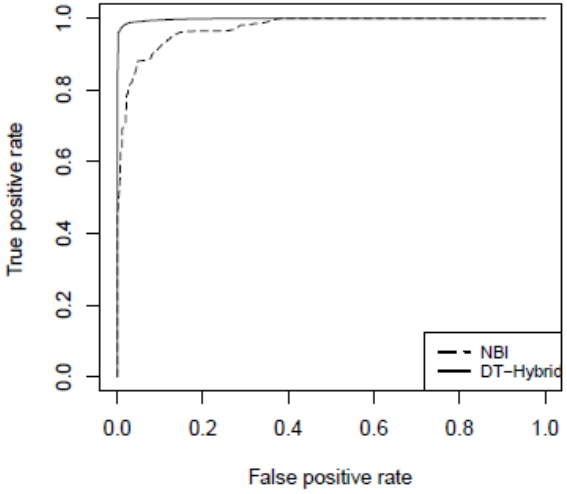
DT-Hybrid - Benchmarks



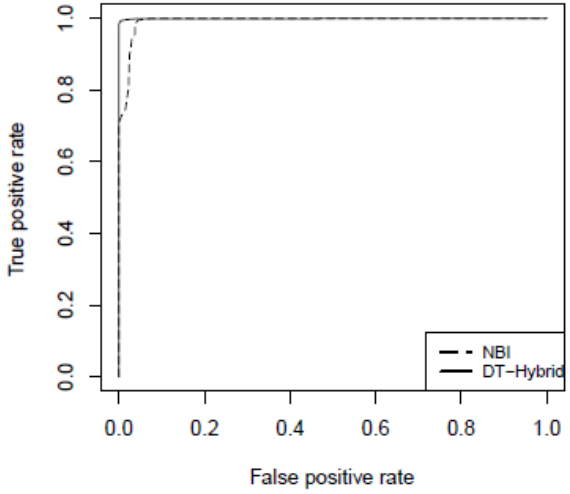
(a) Enzymes



(b) Ion Channels



(c) GPCRs



(d) Nuclear Receptors

Evaluation of DT-Hybrid

- It is an excellent method that allows us to use the recommendation technique in other areas in addition to the standard ones.
  - We have applied it with good results in bioinformatics.
- The method was designed to be used in a distributed environments with high degree of parallelization.

Part VIII

Recommendation and R

Practical Examples

- **http://alpha.dmi.unict.it/~alaimos/2014recommendationExamples.R**

THE END