Diversity Algorithms for Recommendation Systems

Ecole d'été BDA MDD 2024

idir.benouaret@epita

Epita

June 25, 2024

Table of Contents

Motivation: Why do we need recommendation

In our daily life:

- Which movie should I watch?
- Which playlist should I listen to?
- What could be a nice destination for my holidays?
- Which book should I read?

Motivation: Why do we need recommendation

In our daily life:

- Which movie should I watch?
- Which playlist should I listen to?
- What could be a nice destination for my holidays?
- Which book should I read?

Information overload:

- Many choices available
- The paradox of choice

Motivation: Why do we need recommendation

In our daily life:

- Which movie should I watch?
- Which playlist should I listen to?
- What could be a nice destination for my holidays?
- Which book should I read?

Information overload:

- Many choices available
- The paradox of choice

The goal of recommender systems is to help people discover content

Examples of Applications

- Movies (Netflix, Amazon), online videos (Youtube)
- Music (Spotify)
- Books
- Software (apps)
- Products (e-commerce platforms)
- People (dating, friends)
- Services (restaurants, hotels)
- News (google news)

Examples of Applications

Definition

Recommender system [4]

RecSys are a subclass of information filtering system that provide suggestions for items that are most pertinent to a particular user

Impact and value of recommender systems

User's point of view:

- Find interesting items
- Save fetching time

Impact and value of recommender systems

User's point of view:

- Find interesting items
- Save fetching time

Provider's point of view:

- Increase user satisfaction, fidelity
- Sell more items
- Better understand what users want

A brief history

- 1990s- first systems (e.g., GroupLens), basic algorithms
- 1995-2000 – rapid commercialization, challenges of scale
- 2000-2005 – first research, mainstream applications
- 2006 –2009 Netflix prize
- 2007 – the first Recommender Systems conference
- 2010s – common application of almost any companies related to users/customers
- now- very active research in academia and industry

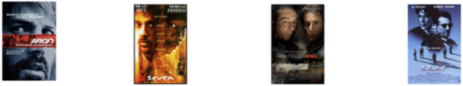
Approaches to Recommendations

Let's consider the previously discussed examples :

- How do the recommendations algorithms work, how are recommendations computed ?
- What is used to generate recommendations ?

Data model

- Input : a set of users, a set of items, the rating matrix. Eventually, auxiliary data such as user attributes, item attributes, etc.



John	5	1	3	5
Tom	?	?	?	2
Alice	4	?	3	?

- Output: Predictions on missing values

Key problems

- **Gathering feedback:** how to collect the data in the rating matrix
- **Predict** Unknown ratings from known ones
  - Mainly interested in possibly high unknown ratings
  - More interested in knowing what you will like than what you won't like
- How to evaluate the **performance** of a recommender system

Gathering ratings (feedback)

- Explicit feedback
  - ask for ratings (e.g., amazon and most of e-commerce websites)
  - users rate items but their own to get good recommendations
- Implicit feedback
  - learn preferences from user interaction. e.g., purchases, clicks, browse time, etc.
  - What about unknown interaction ? Either 'unknown' or 'negative'

Types of recommendation systems

- Non-personalized recommendation
  - Best sellers
  - Most popular
  - Trending now
  - Top 10 in France today
- Content-based filtering
  - how similar is an item *i* to items the user has liked in the past
  - Uses metadata of items (text, keywords, attributes) to measure similarity
- Collaborative filtering
  - Treat items and users as vectors, compute vector similarities
  - Recommend similar items with those the user liked (Item-Knn)
  - Recommend items liked by similar users (User-Knn)
  - Recommendation based on latent factors (MF,SVD)

Table of Contents

Content-based recommendation : in a nutshell

- How similar is the content of item i to items the user has liked in the past ?
- Uses metadata for measuring similarity
- Works even when no ratings are available on items
- Requires metadata! (content)

Content-based recommendations

- Most of CB-based recommendation techniques were applied to recommending text documents (web pages, books, news, etc)
- Content of items can be represented as text documents (Unstructured)
- or structured (Title, genre, author, type, price)

Item profiles

- For each item, create an item profile
- Profile is a vector of features
  - Movies: author, title, actor, director
  - Set of 'important' words in document
  - A vector representation of the text (Word2vec, BERT, etc)

User profile and Prediction

- user profile:
  - weighted average of rated item profiles
  - normalize: weight by difference from average rating for item
- Prediction heuristic:
  - Given a user profile u and item profile i, estimate the utility with cosine similarity

$$cos(u, i) = \frac{u \cdot i}{||u||.||i||}$$

  - Recommend items with high scores

This is very similar to search engines :

- The user profile acts like the query
- Items act as the documents to be ranked

Advantages of content-based models

- **User independence:** does not depend on other users of the system, no need for data on other users
- Able to recommend to users with unique tastes
- **New items:** can be easily incorporated (no cold start for new items), able to recommend unpopular items
- **Transparency:** explainable and understandable recommendations : we recommend to watch **Better Call Saul** because you liked **Breaking Bad**

Limitations of content-based models

- **Limited content analysis**
  - content may not be automatically extractable (multimedia)
  - missing domain knowledge
- **Overspecialization:** 'more of the same', too similar items, expected recommendations
- **New user:** ratings or information about user has to be collected

Collaborative Filtering (CF)

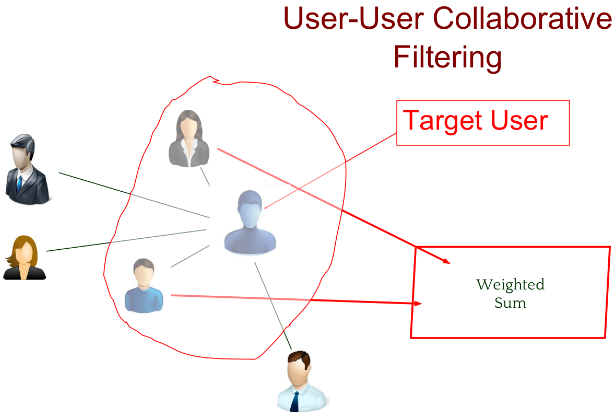
Recommend items based only on the users past behavior

- **User-based CF:** Find similar users to me and recommend what they liked
- **Item-based CF:** Find similar items to those that I have previously liked

The CF ingredients

- List of n users and a list of m items
- Each user has a list of items with associated preference
  - Explicit: rating scores
  - Implicit: purchase records or listened tracks, etc
- Active user for whom the CF prediction task is performed
- Metric for measuring similarity between users/items
- Method for selecting a subset of neighbors
- Method for predicting a rating for items not currently rated by the active user.
- Recommend a list of TopN items

User-based Collaborative filtering

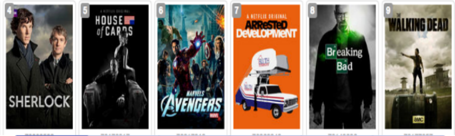


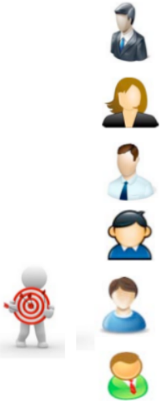
User-based Collaborative filtering

In a nutshell, we have an active user *Bob*

- item *i* is not rated by *Bob*
- find 'similar' users to *Bob* who have rated *i*
- compute predicted rating by *Bob* based on ratings of similar users
- Do the same of others items and recommend those with highest predicted ratings.

User-based CF: example





2		2	4	5	
5		4			1
		5		2	
	1		5		4
		4			2
4	5		1		

Similarity Computation

- let r_u be the vector of user u's ratings
- Jaccard Similarity :
  - Problem: ignores the values of ratings.
- Cosine Similarity measure:

$$sim(u,v) = \frac{r_u \cdot r_v}{||r_u||.||r_v||}$$

- Pearson correlation measure:

$$sim(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - r_u)(r_{v_i} - r_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - r_u)^2} . \sqrt{\sum_{i \in I_{uv}} (r_{v_i} - r_v)^2}}$$

Challenges of User-based CF Algorithms

- **Sparsity:** evaluation of large item sets, users interactions are under 1%
- Difficult to make predictions for users with unique preferences
- **Scalability:** Nearest neighbor require computation that grows with both the number of users and the number of items.
- User-User similarity is hard to maintain

Item-based Collaborative Filtering

Item-Item Collaborative Filtering

Item-based Collaborative Filtering

- Look into the items the target user has rated
- Compute how similar they are to the target item
  - Similarity only use past ratings from other users
- Select k most similar items
- Compute Prediction by taking weighted average on the target user's ratings on the most similar items.

$$\hat{r}_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} . r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

with, s_{ij}: similarity of items i and j, r_{xj}: rating of user x on item j and $N(i;x)$ set of items rated by x similar to i

Item-based CF: example

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- unknown rating   - rating between 1 to 5

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Item-based CF: N=2

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- estimate rating of movie **1** by user **5**

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Item-based CF: N=2

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Neighbor selection:
Identify movies similar to movie **1**, **rated by user 5**

Here we use Pearson correlation as similarity:

1) Subtract mean rating m_i from each movie i

$m_1 = (1+3+5+5+4)/5 = 3.6$

row 1: *[-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]*

2) Compute cosine similarities between rows

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 30

Item-Item Similarity Computation

- Similarity between items i and j computed by finding users who have rated them and then applying a similarity function to their ratings.
- Cosine similarity: items are vectors in the n dimensional user space
- correlation based similarity: using the Pearson-r correlation (used only in cases where the users rated both item i and item j)

Advantages of Neighborhood models

- Requires minimal knowledge engineering efforts
- Users and products are symbols without any internal structure or characteristics
- No need of metadata (features) of items/users
- Produces good-enough results in most cases

Limitations of Neighborhood models

The Sparsity problem

- Typically: large product sets, user ratings for a small percentage of them
- Example Amazon: millions of books and a user may have bought hundreds of books
  - the probability that two users that have bought 100 books have a common book (in a catalogue of 1 million books) is 0.01 (with 50 and 10 millions is 0.0002).

Limitations of Neighborhood models

Cold-start

- **New User Problem**: to make accurate recommendations, the system must first learn the user's preferences from the ratings.
- **New Item Problem**: New items are added regularly to recommender systems. Until the new item is rated by a substantial number of users, the recommender system is not able to recommend it.

Limitations of Neighborhood models

The Scalability Problem

- Nearest neighbor algorithms require computations that grows with both the number of users and items
- With millions of users and items a web-based recommender can suffer serious scalability problems
- Sometimes, predictions need to be made in real time and many predictions may potentially be requested at the same time
- The worst case complexity is $O(mn)$ (n users and m items)

Matrix Factorization: The Netflix Prize

- Training data
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- Test data
  - Last few ratings of each user (2.8 million)
  - Evaluation criterion: Root Mean Square Error (RMSE)
  - Netflix's system RMSE: 0.9514
- Competition
  - 2,700+ teams
  - 1 million dollars prize for 10% improvement on Netflix

The Netflix Utility Matrix R

Matrix R

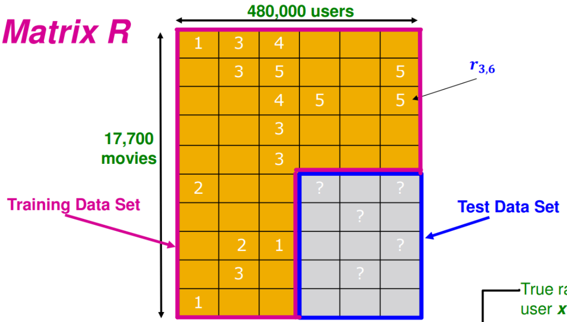
480,000 users

17,700 movies

1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			2		2
				5	
	2	1			1
	3			3	
1					

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Utility Matrix R: Evaluation



True rating of user ***x*** on item ***i***

$$\textbf{RMSE} = \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$$

Predicted rating

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 4

And the Winner is (was)

- BellKor's Pragmatic Chaos
- After 3 years of collaborative work

Recommendation via Optimization

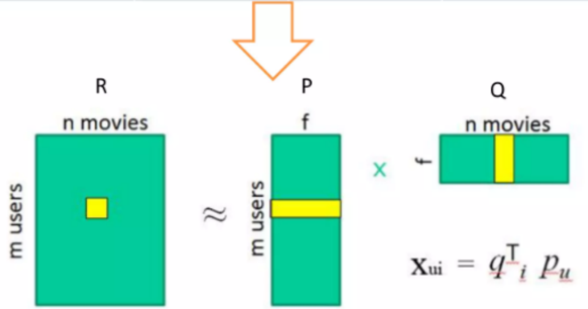
Goal: make good predictions

- Quantify quality of recommendations using RMSE: lower RMSE= better predictions= better recommendations
- We want to make good recommendations on items that user has not yet seen. **Can't really do this!**
- Let's build a system that works well on known (user,item) ratings
- And Hope the system will also be good at predicting unknown ratings (generalization)

Latent factor models

Matrix Factorization

User	HarryPotter	Batman	Spiderman
U1	5	3	4
U2	?	2	4
U3	4	2	?



$$X_{ui} = q^T_i \, p_u$$

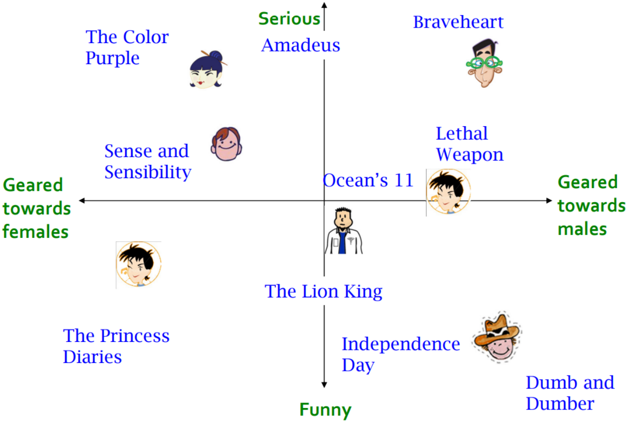
Latent factor models

- R= Rating matrix, m users, n items
- P= User matrix, m users, f latent factors/features
- Q= Item Matrix, n movies, f latent factors/features

Interpretation:

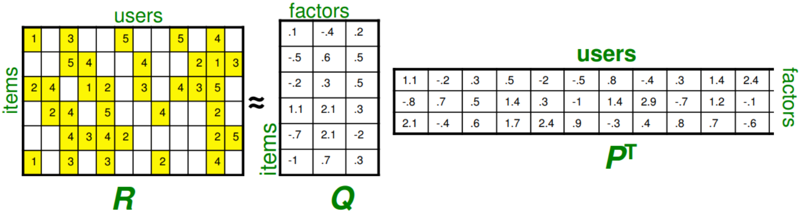
- The vector p_u indicates how much user u likes f latent factors
- The vector q_i indicates how much an item obtains f latent factors

Latent factor models



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

- For now let's assume we can approximate the rating matrix R as a product of QP^T
- R has missing entries but let's ignore that for now! Basically, we will want the reconstruction error to be small on known ratings and we don't care about the values on the missing ones



Ratings as Products of Factors

- **How to estimate the missing rating of user x for item i?**

$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$

q_i = row i of Q
p_x = column x of P^T

users (columns) × items (rows):

1		3			5			5		4	
		5	4	?		4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

Q — items (rows) × factors (columns):

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

•

P^T — factors (rows) × users (columns):

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 19

Ratings as Products of Factors

- **How to estimate the missing rating of user *x* for item *i*?**

$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$

q_i = row i of Q
p_x = column x of P^T

users (columns) / items (rows)

1		3			5			5		4	
		5	4	?		4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

Q — items (rows) × factors (columns)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

•

P^T — factors (rows) × users (columns)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 20

Ratings as Products of Factors

- **How to estimate the missing rating of user *x* for item *i*?**

$$\hat{r}_{xi} = q_i \cdot p_x$$

$$= \sum_f q_{if} \cdot p_{xf}$$

q_i = row i of Q

p_x = column x of P^T

users (columns) × items (rows):

1		3			5			5		4	
		5	4	**2.4**		4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

Q — items (rows) × *f* factors (columns):

.1	-.4	.2
-.5	**.6**	**.5**
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

•

P^T — *f* factors (rows) × users (columns):

1.1	-.2	.3	.5	**-2**	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	**.3**	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	**2.4**	.9	-.3	.4	.8	.7	-.6	.1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 21

Finding the Latent Factors

- Our goal is to find P and Q such that:

$$Min_{P,Q} \sum_{(i,x)\in R} (r_{xi} - q_i.p_x)^2$$

- This is our loss function
- Recall we have many missing values (a big risk of overfitting)
- The idea is to use only available ratings and hope it works of missing data
- Now it's a matter of Optimization

Dealing with missing entries

- To solve overfitting we introduce **regularization:**

$$Min_{P,Q} \sum_{(i,x) \in R} (r_{xi} - q_i.p_x)^2 + \lambda(||q_i||^2 + ||p_x||^2)$$

- Optimization is usually performed using stochastic Gradient Descent (SGD)

Table of Contents

Why do we need diversity in recommendations?

- Prevent Monotony: Avoid recommending too many similar items, which can lead to user fatigue.
- Increase Engagement: Diverse recommendations keep users interested and engaged over time. (Spotify [1])
- Expose Users to New Items: Help users discover items they wouldn't have found on their own.
- Broad Coverage: Ensure a wide range of items get exposure, not just the most popular ones.

Diversity Enhancement Problem

The general relevance-diversity problem [5]

The task of selecting a subset S of k items from the set of all items $\mathcal{I}$ in order to maximize an objective function that considers both **relevance** of items as well as their **diversity**.

$$\max_{S} \quad (1-\lambda).\frac{1}{|S|}\sum_{i\in S} rel(i) + \lambda.div(S)$$

with $|S| = k$, $rel(i)$ can be computed using any RecSys algorithm and $div(S)$ is the diversity of the recommended items.

- Warning : this problem is generally **NP-hard**

Measuring Diversity between two items

Explicit v.s. Implicit Features

- Diversity often involves a dissimilarity measure between items.
- How do we measure diversity of two items i and j

Diversity with explicit features (category-based diversity)

- Attributes of items (artists, genres, brands, prices, etc)
- Semantic taxonomies (e.g., topic hierarchy)
- e.g., $div(i,j) = 1 - \frac{topics(i) \cap topics(j)}{topics(i) \cup topics(j)}$

Diversity with implicit Features (distance-based diversity)

- Observed features: views, clicks, purchases
- Learned features: latent factors, embeddings
- e.g, $div(i,j) = 1 - cosine(v_i, v_j)$

- **Pairwise measures** use the average dissimilarity to characterize the diversity of a recommendation list : ILD [7]

$$div(S) = \frac{\sum_{i \in S} \sum_{j \in S \setminus \{i\}} (dist(i,j))}{|S|(|S|-1)}$$

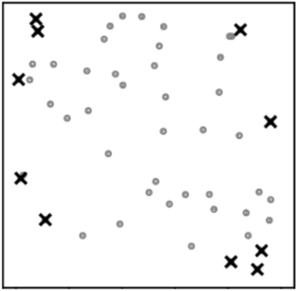
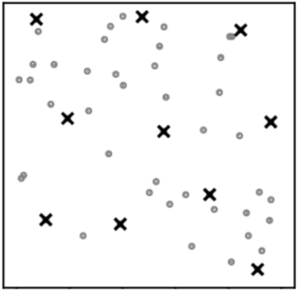
Another popular diversity metrics is **dispersion** :

$$div(S) = \min_{i,j \in S} dist(i,j)$$

Recall that we want to maximize $div(S)$, **What's the difference between the two measures?**

Measuring Diversity of a Set

Pairwise v.s. Set-level Measures

- Dispersion favors scattered items
- Intra-List Distance favors extremes

Measuring Diversity of a Set

Pairwise v.s. Set-level Measures

- Set-level measures define the utility of the entire set of recommended items as a whole
- One of the widely used set-level diversity metrics is coverage
- Coverage can be measured in proportionate to the number of distinct items that are recommended [6]
- Or by the number of topics covered by the recommended items[2]

- MMR is the most pioneering diversity approach

1. Start with the most relevant item, add it to the recommendation list
2. Repeatedly select the next most relevant item, but penalize relevance if it's too similar to already selected items

$$i^* = \underset{i \in \mathcal{I} \setminus (P \cup S)}{\arg\max} \left((1 - \lambda) \cdot rel(i) + \lambda \min_{j \in S} dist(i, j) \right)$$

3. Repeat until we have the desired number of items

Algorithms
MMR algorithm

Algorithm 1 MMR: Maximal Marginal Relevance

Require: Ground set of items I

Ensure: List of recommended items S

$S \leftarrow \emptyset$
while $|S| < k$ **do**
  $i^* \leftarrow \arg\max_{i \in I - S} (1 - \lambda) rel(i) + \lambda \min_{j \in S} dist(i, j)$
  Append item i^* to list S
end while

Algorithms

Max-Sum Diversification

- The goal is to select a set of items S so that it maximizes the sum of relevance and distance between items

$$S^* = \underset{|S|=k, S \subset \mathcal{I} \setminus P}{\arg\max} \left((1-\lambda) \cdot \sum_{i \in S} rel(i) + \lambda \sum_{i \in S} \sum_{j \in S - \{i\}} dist(i,j) \right)$$

- The problem is known to be **NP-Hard**
- But a greedy algorithm gives a 1/2- approximation guarantee [3].

Algorithms

re-ranking algorithms

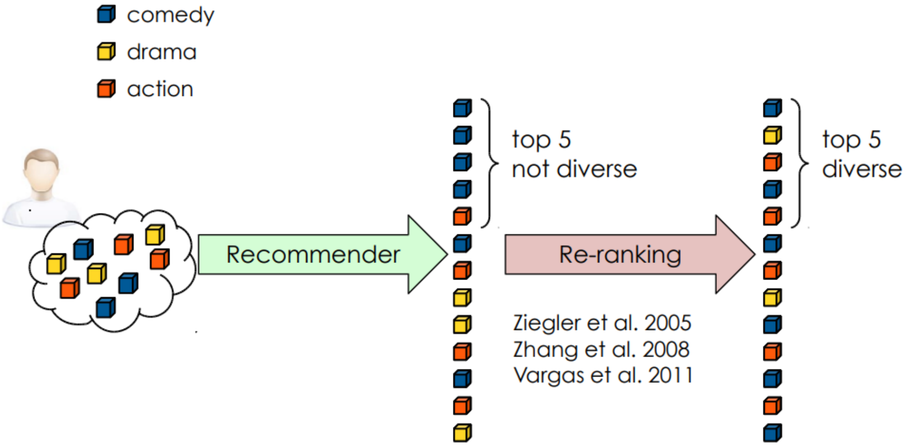


Figure: Adapted from *Alexandros Karatzoglou – September 06, 2013 – Recommender Systems*

Algorithms

re-ranking with SWAP

- Picks the k items with the highest utilities
- Iterates through items and swaps the item from the top-k list that contributes the least to diversity with the next highest utility
- Swap is performed only if the swapped item brings more diversity to the recommendation list
- Uses a utility upper bound ub as a condition to stop the swapping which corresponds to how much drop in relevance is tolerated.

- DUM [2] tackles the diversity problem by maximizing the utility of the selected items weighted by the gain they provide in diversity

$$S^* = \arg\max_{S \in \Theta} \sum_{k=1}^{|\mathcal{I}|} [f(S_{k-1}) - f(S_k)] \cdot u(i_k)$$

- The term $f(S_{k-1}) - f(S_k)$ measures the gain in diversity that is achieved when i_k is added to S_{k-1}, and $u(i)$ measures the utility of item i.
- DUM first ranks all items according to their utility and then iteratively selects the next item if an only if it increases the diversity.

Future Research and Trends

- Personalized and adaptive diversity
  - It is natural that different users may have different preferences to diversity
  - There is a need to adapt the level of diversity for each user
- Temporal diversity
  - Preferences of users towards diversified recommendations might change over time
- Explainable diversity
  - Most of the current explainable recommendation approaches focus on finding associations between the recommendation results and relevant users or items
  - It is still a void field to study how diversified recommendations can be properly explained.

Table of Contents

Evaluation Metrics for Diversity

Intra-list Distance

- Measures how dissimilar recommended items are:

$$ILD = \frac{1}{|\mathcal{U}|} \frac{\sum_{i \in S_u} \sum_{j \in S \setminus \{i\}} (dist(i, j))}{|S_u|(|S_u| - 1)}$$

Catalog Coverage

- Measures the fraction of relevant items that are recommended at least once

$$CCOV = \frac{|\cup_{u \in \mathcal{U}} S_u|}{|\mathcal{I}|}$$

Topic Coverage

- Measures the topic coverage of the user's interests by counting the number of relevant topics (*topics()* function) that are recommended to the user out of the number of topics in the user profile:

$$UCOV = \frac{1}{|\mathcal{U}|} \times \sum_{u \in \mathcal{U}} \frac{|\cup_{i \in S_u} topics(u) \cap \cup_{i \in P} topics(i)|}{|\cup_{i \in P} topics(i)|}$$

Lab session

- Form groups of 3-4 students
- Implement some classical algorithms using a recommendation library to build movie recommendations
- Only some Python knowledge (pandas,numpy, scipy) is required
- Apply a diversification algorithm and compare recommendations
- Please talk to me :)

References I

Ashton Anderson, Lucas Maystre, Ian Anderson, Rishabh Mehrotra, and Mounia Lalmas.
Algorithmic effects on the diversity of consumption on spotify.
In *Proceedings of the web conference 2020*, pages 2155–2165, 2020.

Azin Ashkan, Branislav Kveton, Shlomo Berkovsky, and Zheng Wen.
Optimal greedy diversity for recommendation.
In *IJCAI*, volume 15, pages 1742–1748, 2015.

Allan Borodin, Aadhar Jain, Hyun Chul Lee, and Yuli Ye.
Max-sum diversification, monotone submodular functions, and dynamic updates.
ACM Transactions on Algorithms (TALG), 13(3):1–25, 2017.

References II

Robin Burke, Alexander Felfernig, and Mehmet H Göker.
Recommender systems: An overview.
Ai Magazine, 32(3):13–18, 2011.

Naoto Ohsaka and Riku Togashi.
A critical reexamination of intra-list distance and dispersion.
In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1619–1628, 2023.

Shameem A Puthiya Parambath, Nicolas Usunier, and Yves Grandvalet.
A coverage-based approach to recommendation diversity on similarity graph.
In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 15–22, 2016.

References III

Mi Zhang and Neil Hurley.
Avoiding monotony: improving the diversity of recommendation lists.
In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 123–130, 2008.