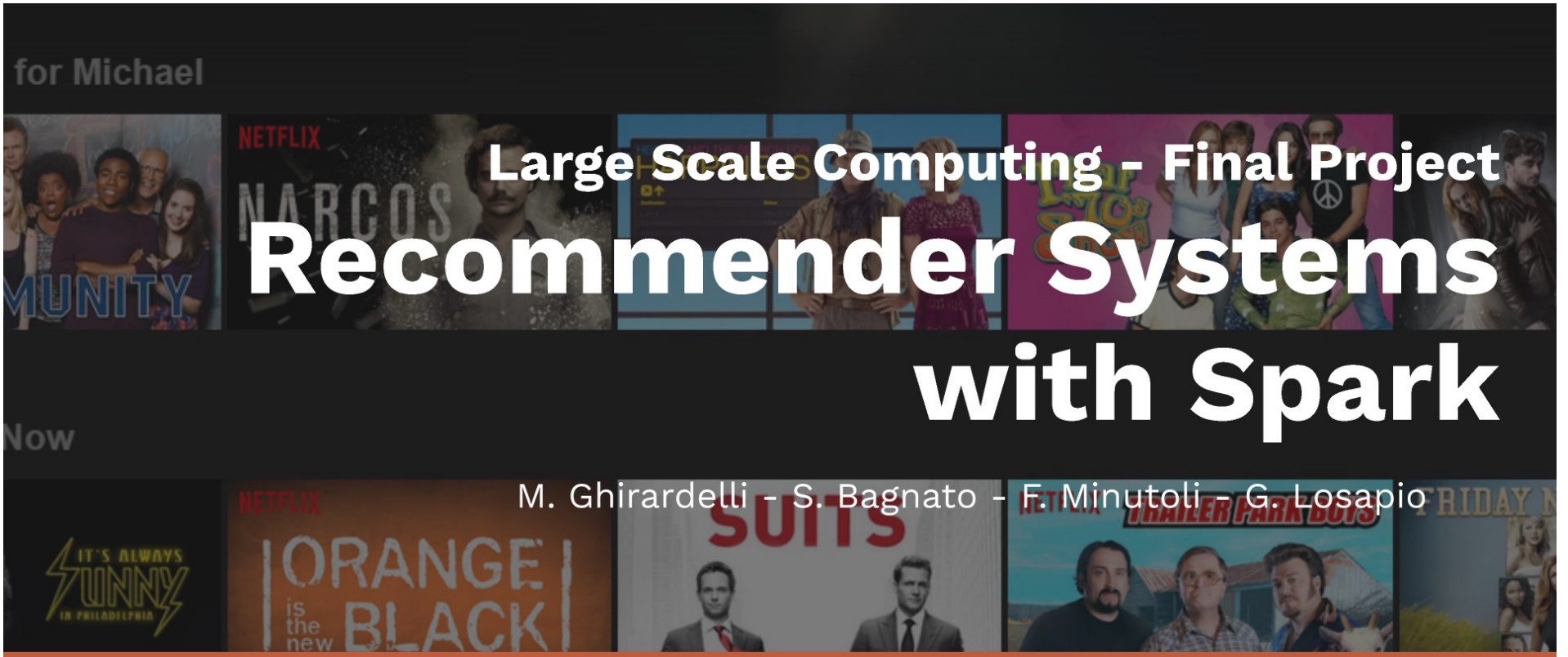


Large Scale Computing - Final Project

Recommender Systems with Spark

M. Ghirardelli - S. Bagnato - F. Minutoli - G. Losapio

Problem & Data **Algorithms** **Results**

Problem & Data

(by Matteo Ghirardelli)

Introduction: LSC Final Project

- Problem addressed: Building a recommender system
- Dataset used: MovieLens dataset
- Focus of the project: Data analysis, processing and machine learning using Spark and the Hadoop architecture
- Members of the team and related work:
  - Matteo Ghirardelli: Data analysis and pre-processing
  - Sofia Bagnato: Cosine similarity (IBCF)
  - Federico Minutoli: Genre correlations (GBCF)
  - Gianvito Losapio: Alternating least squares (ALS)

Introduction: Recommender systems

- Information filtering systems that seek to predict the "rating" a user would give to an item
- Widely used in different companies such as Amazon, Netflix, Spotify
- Techniques:
  - Content-based filtering: analyzes the content of an item (e.g. text)
  - Collaborative filtering: uses the tastes of users to suggest items.
    - IBCF
    - UBCF
  - Hybrid filtering: combination of collaborative and content based filtering

Introduction: MovieLens dataset

- https://grouplens.org/datasets/movielens/latest/
- Describes 5-star a rating activity from the MovieLens recommendation service (http://movielens.org) .
- Raw form: 27753444 ratings and 58098 movies.
- Data created by 283228 users between January 09, 1995 and September 26, 2018.

Movies.csv

movieId	title	genres
1	Toy Story (1995)	Adventure|Animati...
2	Jumanji (1995)	Adventure|Childre...
3	Grumpier Old Men ...	Comedy|Romance

Ratings.csv

userId	movieId	rating	timestamp
1	307	3.5	1256677221
1	481	3.5	1256677456
1	1091	1.5	1256677471

MovieLens dataset: processing steps

1. Selecting valid data with respect to the application domain and to all the existing values for each feature

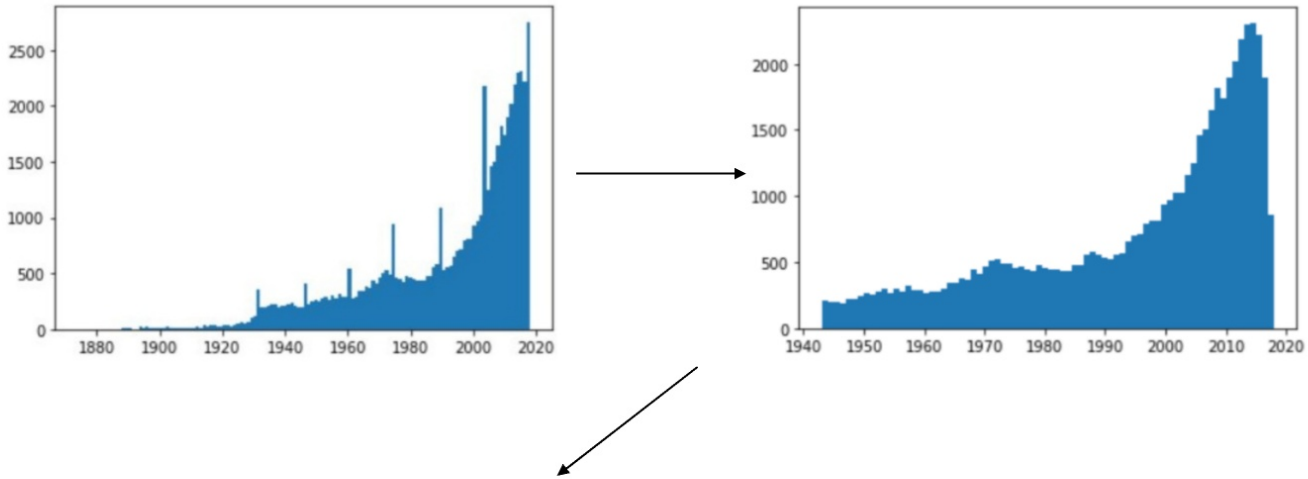
2. Select significant data among the valid data

3. Split train/test, compute intermediate results for recommendation procedures

The output of such procedure will be a sequence of spark transformations performed using both pair RDDs and Dataframes.
Tools/techniques used:

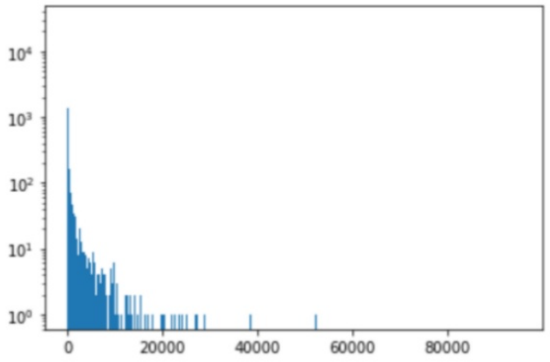
- Map-reduce queries
- Histograms
- Boxplots

MovieLens dataset: Analysis and processing (1)

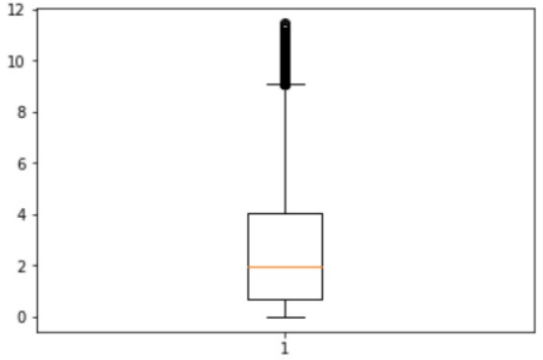


Delete all «IMAX» and «(no genres listed)» movies → The number of filtered movies is now 50282

MovieLens dataset: Analysis and processing (2)

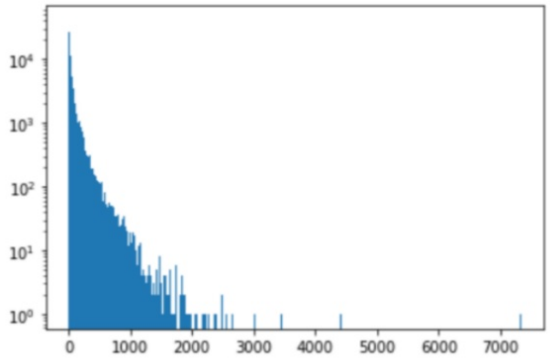


The median value for the counts is 7.0
The 70-th percentile is 34.0
The max value for the outliers is 95139.99999999991
The min value for the outliers is 4917.999999999998
The number of movies that are outliers can be approximated by 1061
The number of movies in the correct range can be approximated by 41696

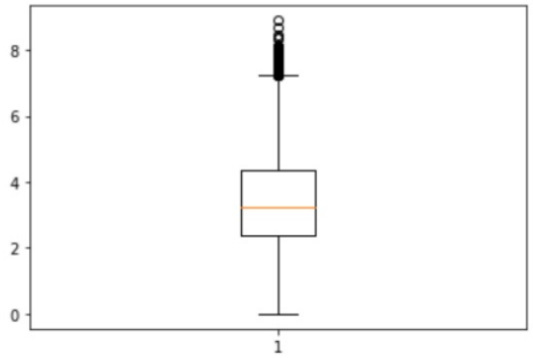


Percentile chosen as a lower bound for the count plot is 34.0

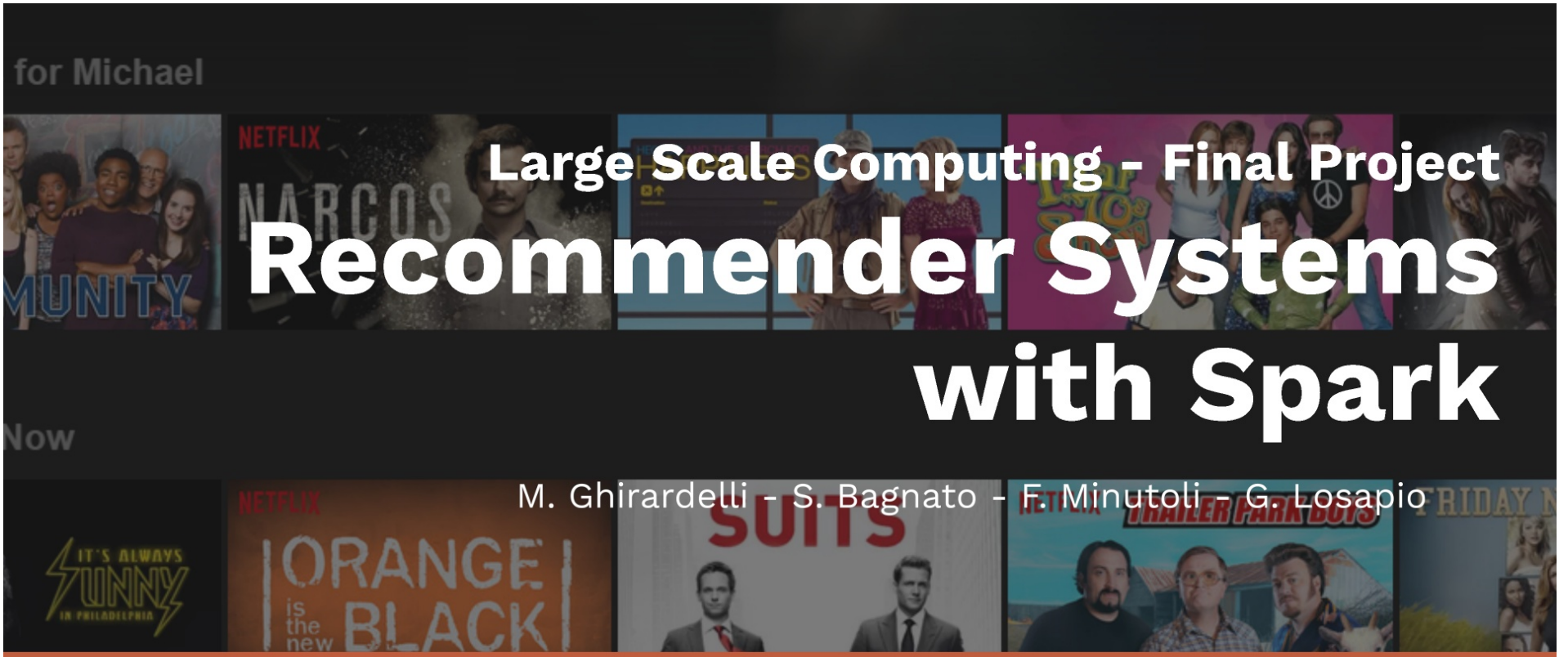
MovieLens dataset: Analysis and processing (3)




```
The max value for the outliers is  7327.999999999999
The min value for the outliers is  1096.9999999999998
The number of users that are outliers can be approximated by  849
The number of users in the correct range can be approximated by  280318
```




```
Percentile chosen as a lower bound for the count plot is  25.0
Percentile chosen as a upper bound for the count plot is  519.0
```



Large Scale Computing - Final Project

Recommender Systems with Spark

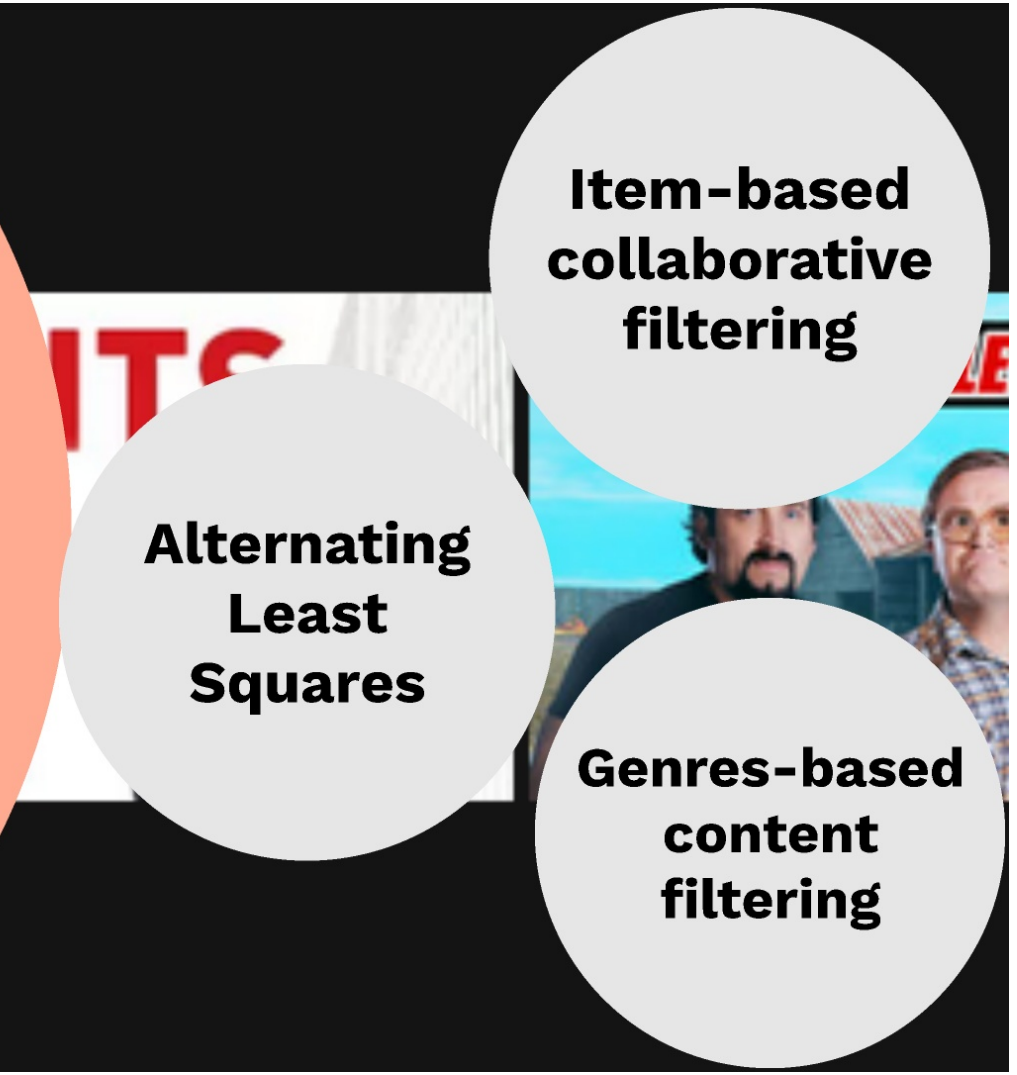
M. Ghirardelli - S. Bagnato - F. Minutoli - G. Losapio

Problem & Data **Algorithms** **Results**

Algorithms

Three different solutions

- Item-based Collaborative Filtering (by Sofia Bagnato)
- Alternating Least Squares (by Gianvito Losapio)
- Genres-based Content Filtering (by Federico Minutoli)



IBCF (by Sofia Bagnato)

IBCF

- It stands for item-based collaborative filtering;
- Collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents;
- Agents are, in this case, movies: the similarity between movies becomes the main metric to use to make recommendations;
- Invented and used by Amazon in 1998.

How it works- initializing

- The predictor first builds a movie-movie similarity matrix, containing for each pair of movies the similarity between them;
- The similarity is computed as the cosine similarity.
- Very high-level pseudocode:

```
for each movie m1 in movies:
    for each user u who saw m1:
        for each movie m2 seen by u:
            calculate similarity(m1,m2)
```

3

After initializing – predicting the score of one movie for one user

- To predict a score (not a rating) of a movie m for a user u we use the following formula:

$$Score(m,u) = avg_m + \frac{\sum_{j \in I_u} (r_{ui} - avg_j) * sim(m,j)}{\sum_{j \in I_u} |sim(m,j)|}$$

Where I_u is the set of movies seen by user u, avg_j is the average rating of movie j across all users, and r_{ui} is the rating user u gave to movie i.

4

Putting all together – ibcf workflow

1. Create movie-movie similarity matrix;
2. For each user in the test set:
   1. Predict scores for all the movies;
   2. Take the first n (n = quantity of movies we want to recommend).
3. When a user asks for recommendations, take them from previously calculated matrix.

5

Space constraints

- The algorithm requires the calculation and the storage of a matrix which dimension can grow as big as $O(N^2)$
- With *12,904* movies the matrix can have $\frac{12{,}904^2}{2} = 83{,}256{,}608$ lines
- Due to the high space storing the similarity matrix took and given the resources we had at hand I could only operate on the movie lens **small** dataset

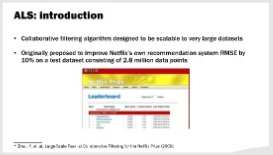
Algorithm cons

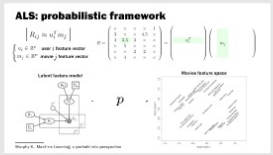
- It is an intuitive algorithm
- However it has some problems:
  - Scalability: worst case $O(N^2M)$, average: $O(N * M)$ (with N movies, M users)
  - Sparsity: even the users who rate most of the movies they see will only have rated a small portion of the available films
  - Cold start: for new users or new movies you don't initially have enough data to make recommendations

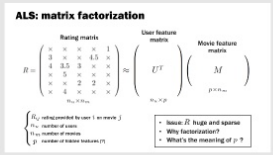
Useful links

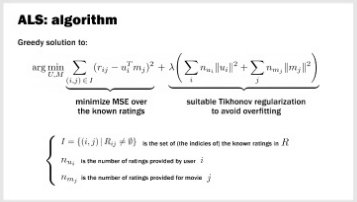
- Implementation:
  - https://medium.com/@wwwbbb8510/python-implementation-of-baseline-item-based-collaborative-filtering-2ba7c8960590
  - https://towardsdatascience.com/collaborative-filtering-recommendation-system-with-apache-spark-using-scala-9a68e02e814d
- Item-based collaborative filtering:
  - https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6C02895D6EE71A705B9FD53032A25FDB?doi=10.1.1.167.7612&rep=rep1&type=pdf

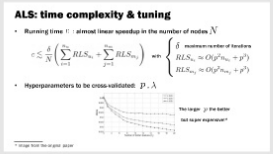
ALS (by Gianvito Losapio)

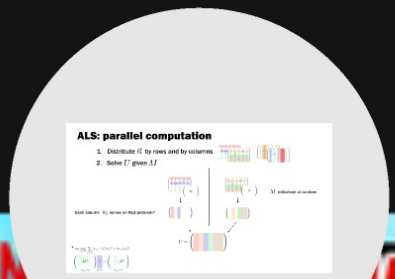


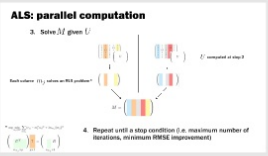






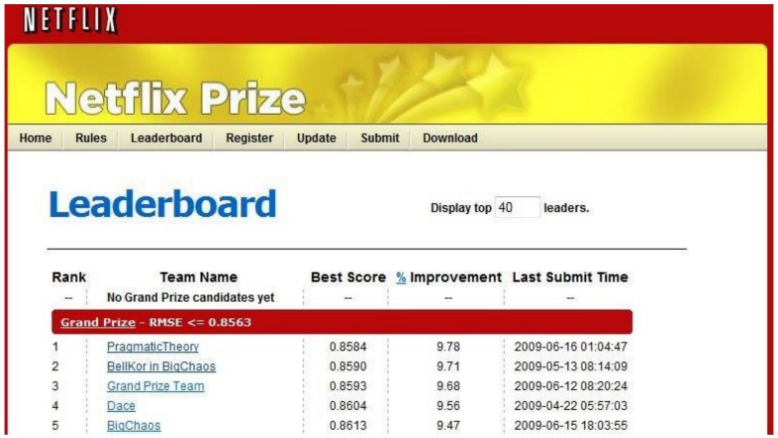






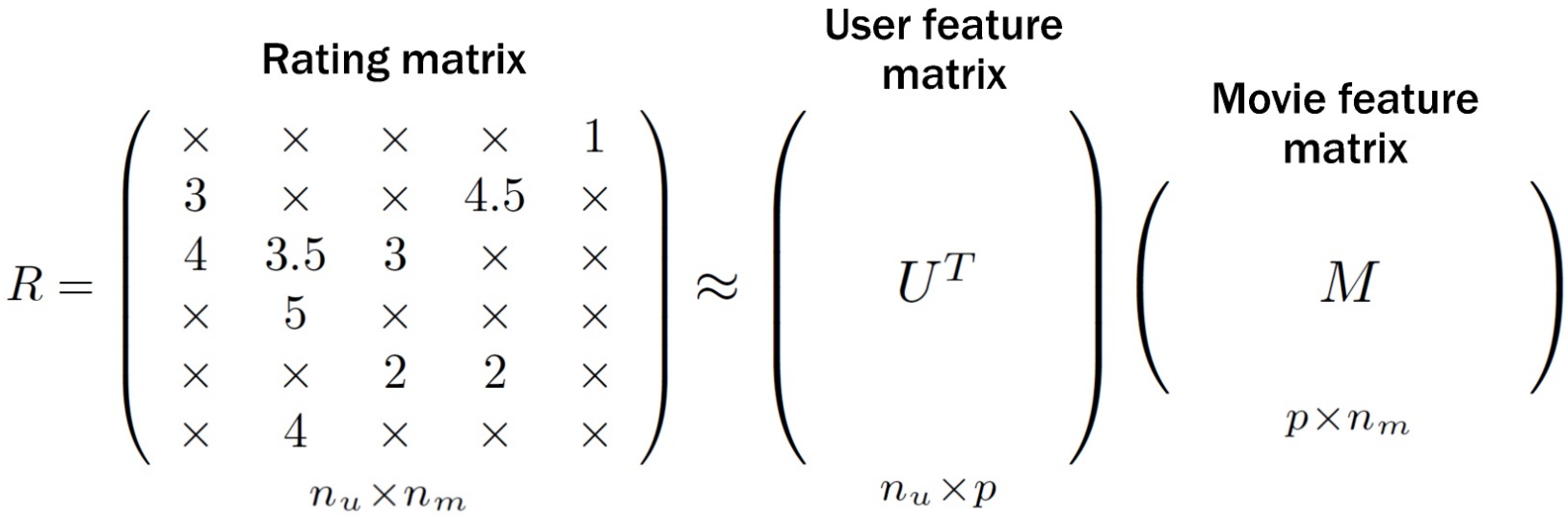
ALS: introduction

- Collaborative filtering algorithm designed to be scalable to very large datasets
- Originally proposed to improve Netflix's own recommendation system RMSE by 10% on a test dataset consisting of 2.8 million data points



* Zhou, Y. et. al, Large-Scale Parallel Collaborative Filtering for the Netflix Prize (2008)

ALS: matrix factorization



- R_{ij} rating provided by user i on movie j
- n_u number of users
- n_m number of movies
- p number of hidden features (?)

- Issue: R huge and sparse
- Why factorization?
- What's the meaning of p ?

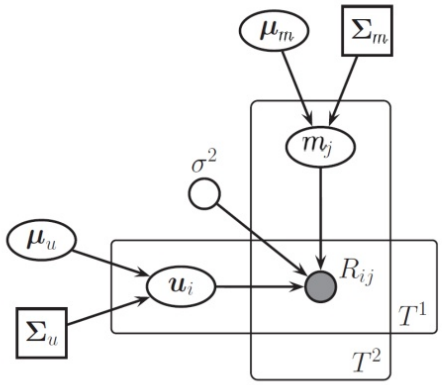
ALS: probabilistic framework

$$R_{ij} \approx u_i^T m_j$$

$$\begin{cases} u_i \in \mathbb{R}^p & \text{user } i \text{ feature vector} \\ m_j \in \mathbb{R}^p & \text{movie } j \text{ feature vector} \end{cases}$$

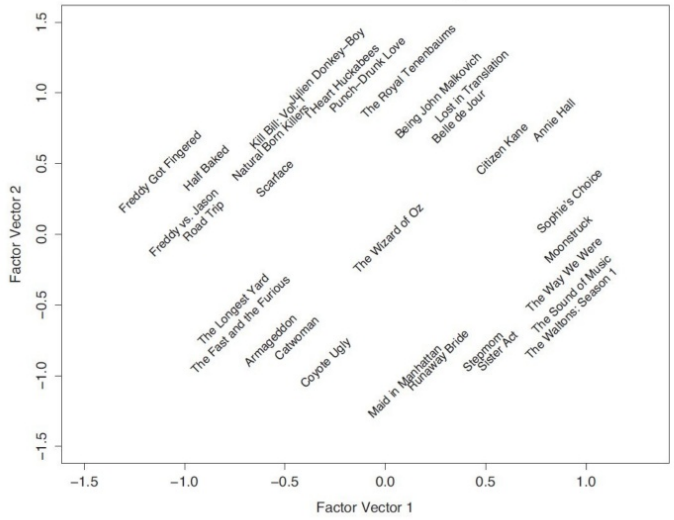
$$R = \begin{pmatrix} \times & \times & \times & \times & 1 \\ 3 & \times & \times & 4.5 & \times \\ 4 & 3.5 & 3 & \times & \times \\ \times & 5 & \times & \times & \times \\ \times & \times & 2 & 2 & \times \\ \times & 4 & \times & \times & \times \end{pmatrix} = \begin{pmatrix} & u_i^T & \end{pmatrix} \begin{pmatrix} & m_j & \end{pmatrix}$$

Latent factors model



p

Movies feature space



Murphy K., Machine Learning: a probabilistic perspective

ALS: algorithm

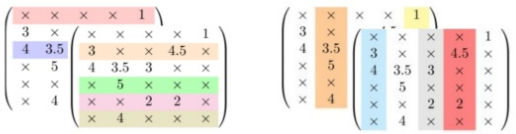
Greedy solution to:

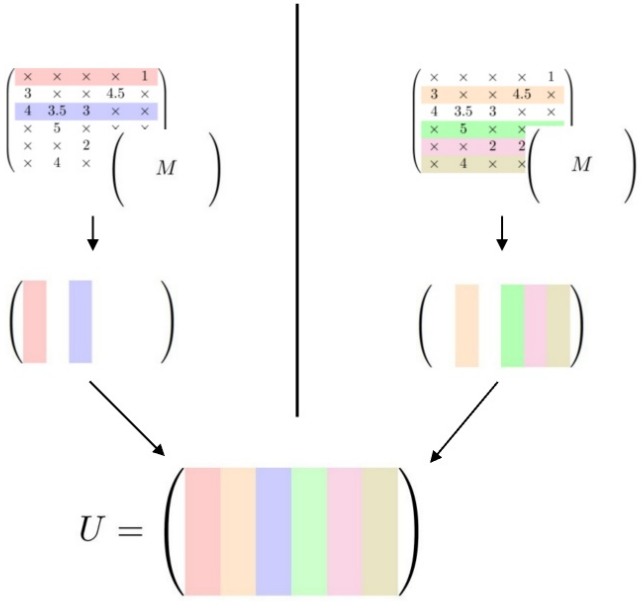
$$\arg\min_{U,M} \underbrace{\sum_{(i,j)\in I} (r_{ij} - u_i^T m_j)^2}_{\text{minimize MSE over the known ratings}} + \underbrace{\lambda\left(\sum_i n_{u_i}\|u_i\|^2 + \sum_j n_{m_j}\|m_j\|^2\right)}_{\text{suitable Tikhonov regularization to avoid overfitting}}$$

- $I = \{(i,j) \mid R_{ij} \neq \emptyset\}$ is the set of (the indicies of) the known ratings in R
- n_{u_i} is the number of ratings provided by user i
- n_{m_j} is the number of ratings provided for movie j

ALS: parallel computation

1. Distribute R by rows and by columns
2. Solve U given M





M initialized at random

Each column u_i solves an RLS problem*

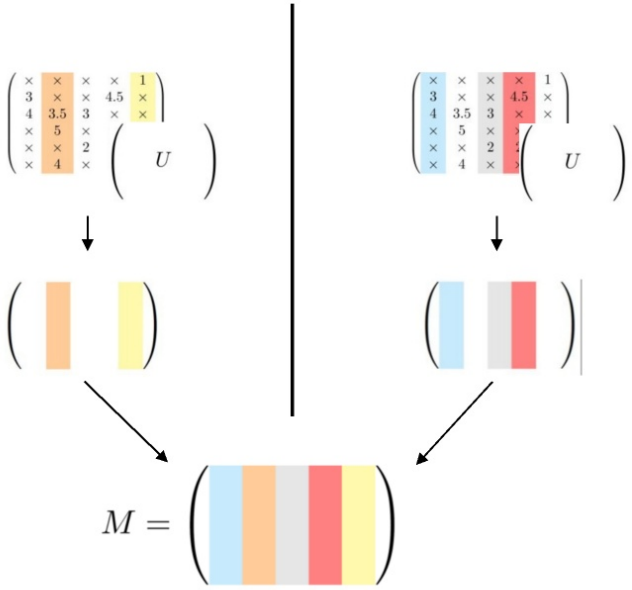
---

* $\arg\min_{u_i \in \mathbb{R}^p} \sum_{j \in I_i} (r_{ij} - u_i^T m_j)^2 + \lambda n_{u_i} \|u_i\|^2$

$$\underset{n_{u_i} \times p}{\begin{pmatrix} M^T \end{pmatrix}} \underset{p \times 1}{\begin{pmatrix} ? \end{pmatrix}} = \underset{n_{u_i} \times 1}{\begin{pmatrix} R^T \end{pmatrix}}$$

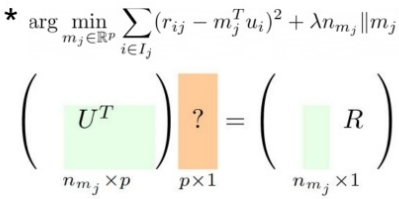
ALS: parallel computation

3. Solve M given U



U computed at step 2

Each column m_j solves an RLS problem*



* $\arg\min_{m_j \in \mathbb{R}^p} \sum_{i \in I_j} (r_{ij} - m_j^T u_i)^2 + \lambda n_{m_j} \|m_j\|^2$

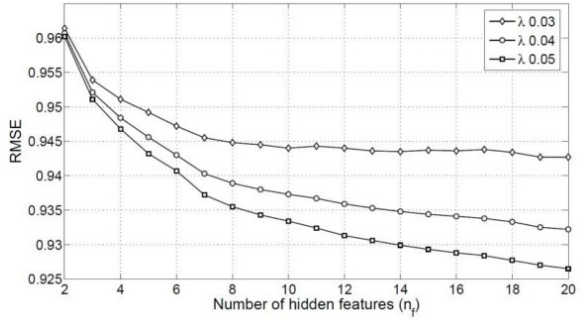
4. Repeat until a stop condition (i.e. maximum number of iterations, minimum RMSE improvement)

ALS: time complexity & tuning

- Running time c : almost linear speedup in the number of nodes N

$$c \lesssim \frac{\delta}{N}\left(\sum_{i=1}^{n_u} RLS_{u_i} + \sum_{j=1}^{n_m} RLS_{m_j}\right) \quad \text{with} \quad \begin{cases} \delta & \text{maximum number of iterations} \\ RLS_{u_i} \approx O(p^2 n_{u_i} + p^3) \\ RLS_{m_j} \approx O(p^2 n_{m_j} + p^3) \end{cases}$$

- Hyperparameters to be cross-validated: p , λ

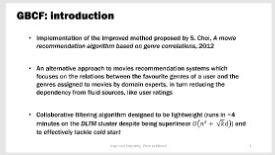


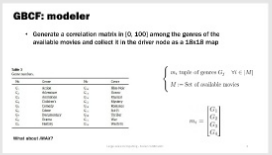
The larger p the better

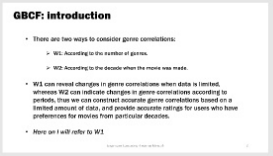
but super expensive!*

* Image from the original paper

GBCF (by Federico Minutoli)







GBCF: introduction

- Implementation of the improved method proposed by S. Choi, *A movie recommendation algorithm based on genre correlations*, 2012

- An alternative approach to movies recommendation systems which focuses on the relations between the favourite genres of a user and the genres assigned to movies by domain experts, in turn reducing the dependency from fluid sources, like user ratings

- Collaborative filtering algorithm designed to be lightweight (runs in ~4 minutes on the ***DLTM*** cluster despite being superlinear $O\left(n^2 + \sqrt{k}\mathrm{d}\right)$) and to effectively tackle *cold start*

GBCF: introduction

- There are two ways to consider genre correlations:
  - W1: According to the number of genres.
  - W2: According to the decade when the movie was made.
- W1 can reveal changes in genre correlations when data is limited, whereas W2 can indicate changes in genre correlations according to periods, thus we can construct accurate genre correlations based on a limited amount of data, and provide accurate ratings for users who have preferences for movies from particular decades.
- *Here on I will refer to W1*

GBCF: modeler

- Generate a correlation matrix in [0, 100] among the genres of the available movies and collect it in the driver node as a 18x18 map

Table 3
Genre numbers.

No	Genre	No	Genre
G_1	Action	G_{10}	Film-Noir
G_2	Adventure	G_{11}	Horror
G_3	Animation	G_{12}	Musical
G_4	Children's	G_{13}	Mystery
G_5	Comedy	G_{14}	Romance
G_6	Crime	G_{15}	Sci-Fi
G_7	Documentary	G_{16}	Thriller
G_8	Drama	G_{17}	War
G_9	Fantasy	G_{18}	Western

$$\begin{cases} m_i \text{ tuple of genres } G_j \quad \forall i \in |M| \\ M := \text{Set of available movies} \end{cases}$$

$$m_i = \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

What about *IMAX*?

GBCF: modeler

- Compute left-to-right occurences map

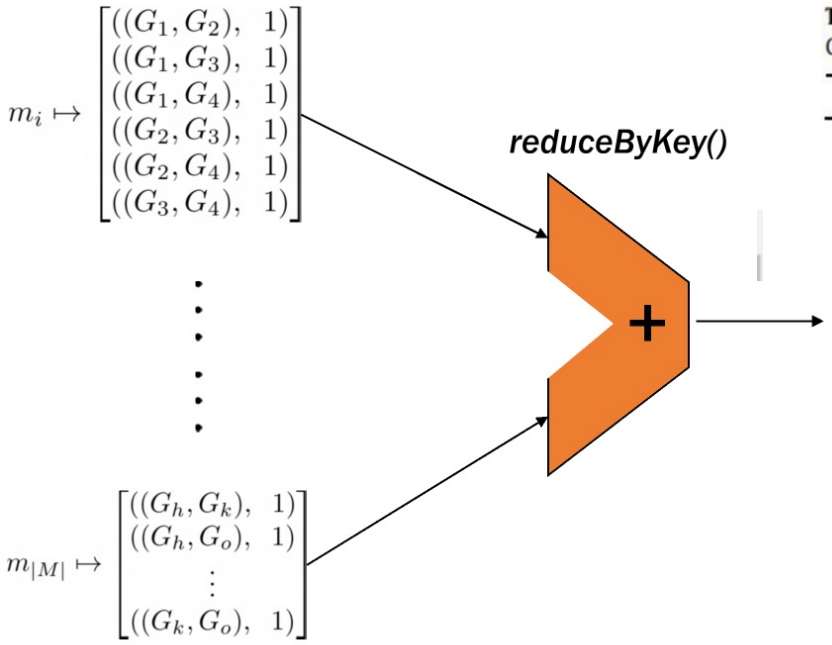


Table 1
Genre correlation matrix.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9
G_1	–	25.29	2.13	3.23	7.15	18.45	0	10.19	10.2
G_2	17.04	–	7.48	20.14	4.84	3.02	0	3.36	23.81
G_3	0.53	2.76	–	20.89	2.75	0	0	0.1	4.081
G_4	1.73	16.01	44.91	–	10.23	0	0	2.75	25.85
G_5	8.65	8.69	13.36	23.13	–	12.08	36.36	23.03	12.92
G_6	7.32	1.77	0	0	3.96	–	0	9.17	0.68
G_7	0	0	0	0	0.44	0	–	0.41	0
G_8	13.31	6.52	0.53	6.71	24.86	30.2	36.36	–	6.12
G_9	1.99	6.91	3.21	9.45	2.09	0.33	0	0.91	–
G_{10}	0	0.19	0.53	0	0.11	5.03	0	0.61	0
G_{11}	3.32	1.58	0.53	0.24	4.51	2.01	0	1.22	0
G_{12}	0.399	1.77	17.11	9.2	4.51	0	18.18	1.52	1.36
G_{13}	1.59	0.59	0.53	0.49	1.43	4.36	0	3.26	0
G_{14}	4.66	5.33	2.13	1.74	22.44	3.02	0	20.79	4.76
G_{15}	14.24	13.24	4.27	3.48	3.41	2.01	0	2.34	8.84
G_{16}	17.7	6.12	2.13	0.24	3.41	19.46	0	11.21	0.68
G_{17}	6.12	2.37	1.06	0.49	1.98	0	9.09	7.74	0.68
G_{18}	1.33	0.79	0	0.49	1.87	0	0	1.32	0

GBCF: predictor

- Implement the formula derived from the paper to predict the ratings between 0 and 5 of the unseen movies for each user

$$R_{p1} = \frac{\sum_{i \in up} \sum_{j \in mg} (r_{i=j} + \frac{r_{i \neq j}}{|mg|-1})}{|up|} \cdot M_{\mu}$$

$$R_{p2} = \frac{\sum_{i \in up} \sum_{j \in mg} r_{i \neq j}}{|up| \cdot |mg|} \cdot M_{\mu}.$$

up	Set of user-preferred genres
Mg	Set of movie M_i genres
M_{μ}	M_i average rating
$r_{i,j}$	Correlation between G_i,G_j

User-preferred genres: G_1, G_3, G_6

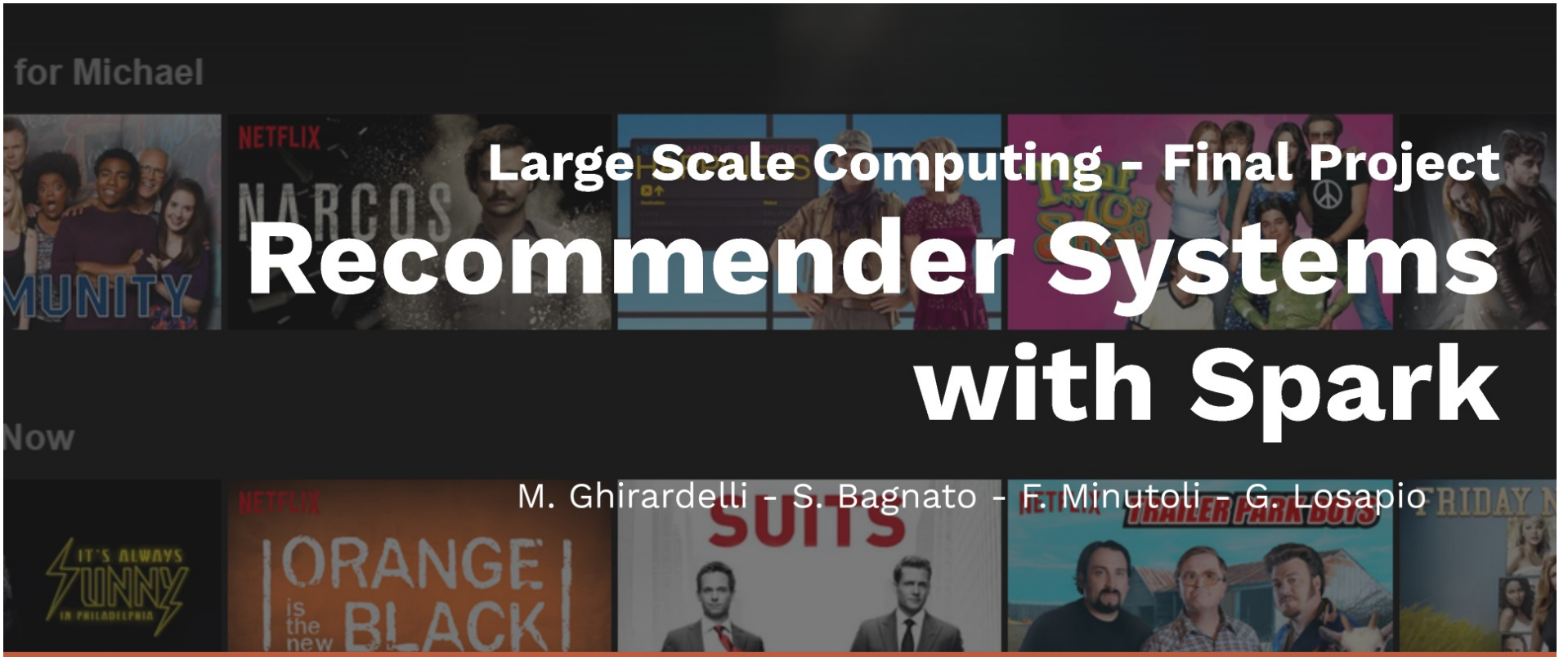
Genre combination of a movie M_i: G_1, G_5

$$\begin{array}{|l|l|}
\hline
G_1 \rightarrow & G_1 \quad r(G_1,G_1) \cdot \mu R_A = V_{G_1,G_1} \\
& G_5 \quad r(G_1,G_5) \cdot \mu R_A = V_{G_1,G_5} \\
\hline
G_3 \rightarrow & G_1 \quad r(G_3,G_1) \cdot \mu R_A = V_{G_3,G_1} \\
& G_5 \quad r(G_3,G_5) \cdot \mu R_A = V_{G_3,G_5} \\
\hline
G_6 \rightarrow & G_1 \quad r(G_6,G_1) \cdot \mu R_A = V_{G_6,G_1} \\
& G_5 \quad r(G_6,G_5) \cdot \mu R_A = V_{G_6,G_5} \\
\hline
\end{array}$$

$$V_{G_1,G_1} + \frac{V_{G_1,G_5}}{|mg|-1} = (1) \searrow$$

$$\frac{V_{G_3,G_1}+V_{G_3,G_5}}{|mg|} = (2) \rightarrow \frac{(1)+(2)+(3)}{|up|}$$

$$\frac{V_{G_6,G_1}+V_{G_6,G_5}}{|mg|} = (3) \nearrow$$



Large Scale Computing - Final Project

Recommender Systems with Spark

M. Ghirardelli - S. Bagnato - F. Minutoli - G. Losapio

Problem & Data **Algorithms** **Results**

Results

Performance metrics

- Root-mean-square-error between the known ratings $r_{ij} \in I$ and the corresponding predictions $\hat{r}_{ij}$

$$RMSE = \sqrt{\frac{1}{|I|}\sum_{(i,j)\in I}(r_{ij} - \hat{r}_{ij})^2}$$

- Accuracy in the form of mean ratio of the matchings between recommended and truly liked movies for each user

$$accuracy = \frac{1}{n_u}\sum_{i=1}^{n_u}\left(\frac{1}{\min(n_r, |L_i|)}\sum_{k=1}^{\min(n_r,|L_i|)} 1(l_{ik} \in L_i)\right)$$

$\begin{cases} n_u & \text{number of users,} \quad n_r \quad \text{number of recommendations provided for each user} \\ l_{ik} & \text{k-th best recommendation for user } i, \quad L_i \quad \text{set of liked movies for user } i \text{ (rating} \geq 3) \end{cases}$

Results

Algorithm	Execution time	RMSE	Accuracy
ALS	13 m *	0.86	0.9953
Cosine-based-similarity	17 m **		0.9938
Centroid	5 m	0.92	0.92

* Hold-out cross-validation with 12 cpus (best model with 50 latent factors and regularization 0.1)

** Calculated on the small version of Movielens dataset

Conclusions

- ALS is designed to minimize RMSE through a probabilistic matrix factorization, hence a good result in RMSE. Moreover, as a matrix factorization algorithm, ALS makes no use of other information (e.g. user preferences, genres, explicit movie ratings) and is very accurate only in cross-validation on a large dataset. As a conclusion, ALS is good for an overall forecasting of the ratings but has to be supported with other filtering criteria to provide decent-tailor recommendations.
- CBCF proves to be reliable even on a large dataset's domain depending on the user-cold-age only in the form of the average per-movie. It works incredibly fast and thus it is well-suited for situations where new ratings (or entire) are available to the system and a new matrix has to be computed. Also, since we can assume that a streaming service will always ask a new user to provide some preferences (in the form of movies, genres, etc...), this algorithm can still provide somewhat accurate predictions at the cold-start phase.
- ICBF is a very accurate algorithm, showing an accuracy comparable to CBCF. It has, however, poor performances compared to them, as it doesn't scale very well to large datasets, due to its high accuracy it is however still in use by today's companies (such as Amazon). It suffers from the classical collaborative filtering approaches problems: cold-start, sparsity and scalability.

What's next?

- Web-server in conjunction with Spark-service (i.e., Ality web-server) to allow for real time interactions between the recommender system and customers
- Online-learning and data-ingestion (with Spark-streaming) using newer algorithms derived from ALS that allow for a faster re-computation of the matrix factors. Hybrid approaches that mix explicit and implicit preferences are actively being explored by Netflix.

Performance metrics

- Root-mean-square-error between the known ratings $r_{ij} \in I$ and the corresponding predictions $\hat{r_{ij}}$

$$RMSE = \sqrt{\frac{1}{|I|} \sum_{(i,j) \in I} (r_{ij} - \hat{r_{ij}})^2}$$

- Accuracy in the form of mean ratio of the matchings between recommended and truly liked movies for each user

$$accuracy = \frac{1}{n_u} \sum_{i=1}^{n_u} \left(\frac{1}{\min(n_t, |L_i|)} \sum_{k=1}^{\min(n_t, |L_i|)} 1\{t_{ik} \in L_i\} \right)$$

n_u number of users, n_t number of recommendations provided for each user

t_{ik} k-th best recommendation for user i , L_i set of liked movies for user i (rating >= 3)

Results

Algorithm	Execution time	RMSE	Accuracy
ALS	143 m *	0.89	0.0002
Cosine-based similarity	17 m **	-	0.9088
Genres	3 m	0.51	0.92

* Hold-out cross validation with 12 pairs (best model with 50 latent factors and regularization 0.1)

** Calculated on the small version of MovieLens dataset

Conclusions

- ALS is designed to minimize RMSE through a probabilistic matrix factorization, hence a good result in RMSE is expected. On the other hand, ALS makes no use of other information (e.g. user preferences, genres, popular movies) and a very low accuracy is also expected on a large dataset. As a conclusion, ALS is good for an overall forecasting of the ratings but has to be supported with other filtering criteria to provide meaningful recommendations.

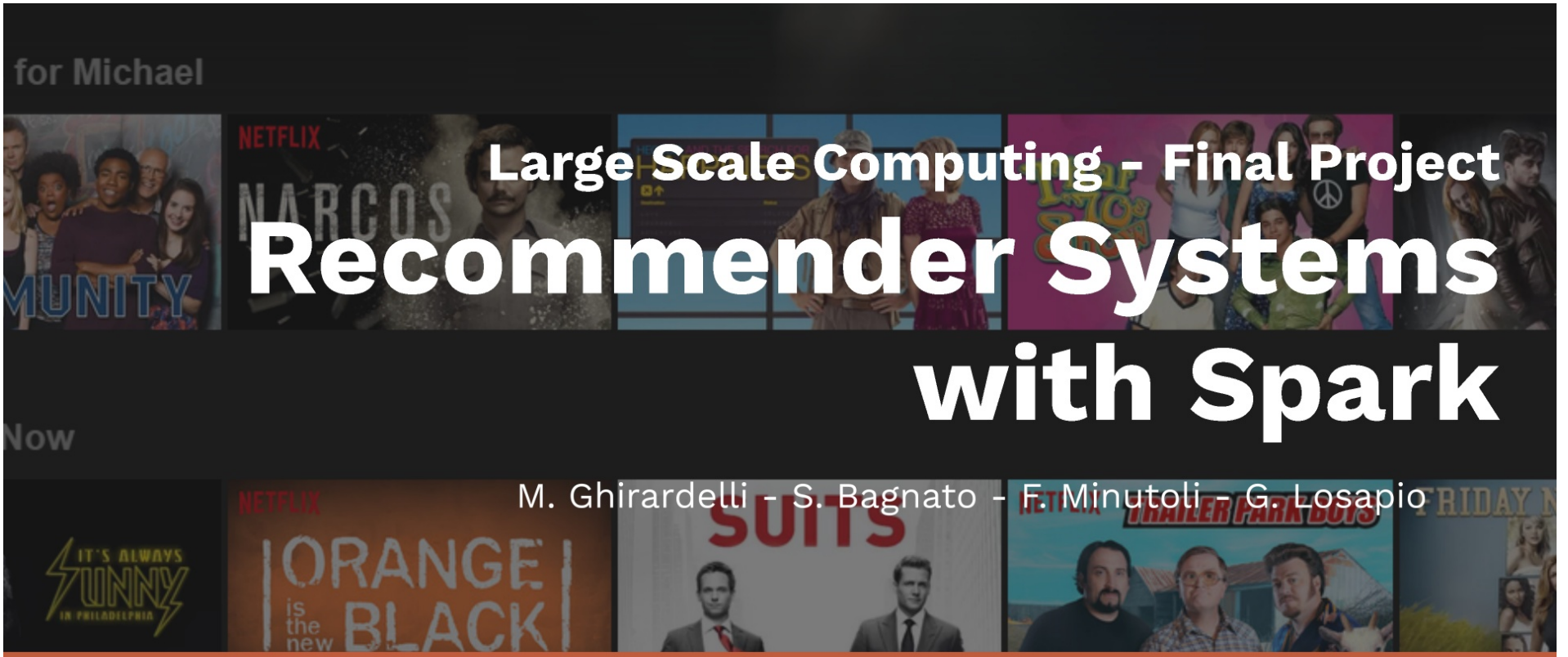
- GBCF proves to be reliable even on a large datasets despite depending on the user ratings only in the form of the average per-movie. It works incredibly fast and thus it is well suited for situations where new ratings (or movies) are available to the system and a new matrix has to be computed. Also, since we can assume that a streaming service will always ask a new user to provide some preferences (in the form of movies, genres, etc...), this algorithm can still provide somewhat accurate predictions at the cold start phase.

- ICBF is a very accurate algorithm, showing an accuracy comparable to GBCF. It has, however, poor performances compared to them, as it doesn't scale very well to large datasets; due to its high accuracy it is however still in use by today's companies (such as Amazon). It suffers from the classical collaborative filtering approaches problems: cold start, sparsity and scalability.

What's next?

- Web server in conjunction with Spark service (i.e., *Jitty web server*) to allow for real time interactions between the recommender system and customers

- Online learning and data ingestion (with Spark streaming) using newer algorithms derived from ALS that allow for a faster re-computation of the matrix factors; hybrid approaches that mix explicit and implicit preferences are actively being explored by Netflix.



Large Scale Computing - Final Project

Recommender Systems with Spark

M. Ghirardelli - S. Bagnato - F. Minutoli - G. Losapio

Problem & Data **Algorithms** **Results**