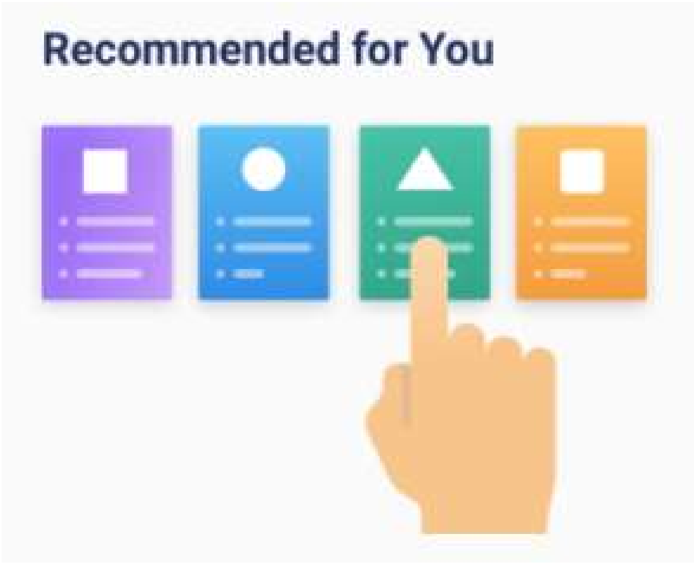
Graduate Certificate in Big Data Analytics

Recommender System Workshops



Dr. Barry Shepherd
Institute of Systems Science
National University of Singapore
Email: barryshepherd@nus.edu.sg

Workshop Rules

- We will use Google Colab and Python
- There is lots of code, introducing multiple concepts, libraries and datasets. Treat the code as a learning resource rather than a test!
- No need to upload any code to demonstrate completion – instead there is a workshop quiz which asks simple questions about the workshops that you should find easy to answer if you do the workshops ☺
- Can revisit the workshops multiple times if you wish. The workshop quiz will be open for the whole course and until Sat Jan 21st midnight

Workshop1: Exploring Association Mining

a) Build and explore association rules for grocery item recommendation
   - Kaggle dataset: each record is a purchase transaction: user ID, date, item purchased

b) Apply association mining to a web-page recommendation scenario
   - Microsoft Vroots dataset: This records the use of www.microsoft.com by 38,000 anonymous, randomly-selected users. It lists all the areas of the web site (Vroots) that each user visited in a one week timeframe.
   - Each record is a pageview record: user ID, vroot visited (there is no datetime field)

c) Experiment with building and using association rules that include virtual items
   - Grocery shopping dataset that includes user demographic data + grocery purchases.
   - Each record ~ one user with a column for every demographic and *every grocery item*
   - Recommend items for purchase bases on past purchases + user demographics

d) Apply a predictive modelling approach to grocery recommendations
   - Use same dataset as used in (c)
   - Build a separate predictive model (decision tree) for every item. Apply all models to a user to make a recommendation. Compare performance with that in (c)

Association Rule Execution & Testing*

- We use a separate test set of baskets (i.e. baskets not used to generate the rules)
- For each item** in each test basket:
  - Remove (holdout) the item or items from the basket
  - Apply the ruleset to the remaining items in the basket
  - Sort the predictions (the rule RHS items) by rule confidence, ignore predictions that are already in the basket. Select the top N: these are the recommendations for that basket
  - If the holdout item(s) is in the recommendations then increment the #hits

E.g. consider a test basket = {A, B, C, D} with D = holdout item and top N = 2

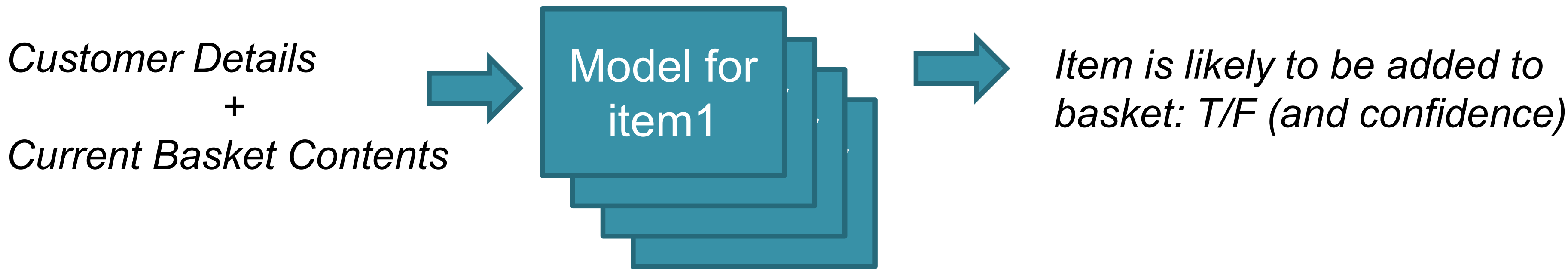
Rules:	Predictions:	Recommendations:	
r1) A => B, cf = 0.4	~~B cf = 0.4~~		
r2) B => F, cf = 0.6	F cf = 0.6	F, cf = 0.6	Hits = 1
r3) C => D, cf = 0.3	D cf = 0.3		
r4) D => B, cf = 0.8			
r5) B, C => D, cf =0.5	D cf =0.5	D, cf = 0.5	
r6) A, B => E, cf = 0.4	E cf =0.4		

- After all tests are performed compute #hits/#tests
- Repeat with random recommendations, ruleset lift = #rulehits/#randomhits

Many variants exist*   * We can also hold out multiple items, e.g. pairs of items*

Predictive Model Approach

- We try building one model for every item. Each model will predict if, given the current basket contents, the target item is also likely to be placed in the basket
- Then execute all models for a new customer/basket and recommend the top N items (e.g. top 5) with the highest confidences (the most confident model outputs)



Example training data for the model to predict if item N will be added into the current basket

basket ID	User Gender	User Age	User Income	Item1 in basket	Item2 in basket	Item3 in basket	Item4 in basket	……	Item N in basket
1	M	24	5600	T	F	F	T		T
2	F	58	2700	F	F	T	F		F
3	F	31	9231	F	T	F	F		F

output

Input variables (before one-hot encoding of gender, age, income)

Workshop1: Files

File Name
workshop1-apriori.ipynb
anonymous-vrootnames-msweb.csv
anonymous-msweb-transactions.txt
Groceries_dataset.csv
baskets.txt

We will use Google colab for this workshop

Upload all files to your Google drive for access by colab

I suggest you create a drive folder called recsys and upload all data files into this folder

Workshop2: Collaborative Filtering

- Explore some simple code that implements User-based and Item-based Collaborative Filtering and also implements basic testing
  - Work with 3 datasets: Movielens (movies) , Jester (jokes), Bookcrossings (books)
  - Compare the performance of User-based versus Item-based CF
  - Compare the performance of different similarity measures
  - Compare the performance with user-normalised ratings versus un-normalised

Workshop2: Datasets

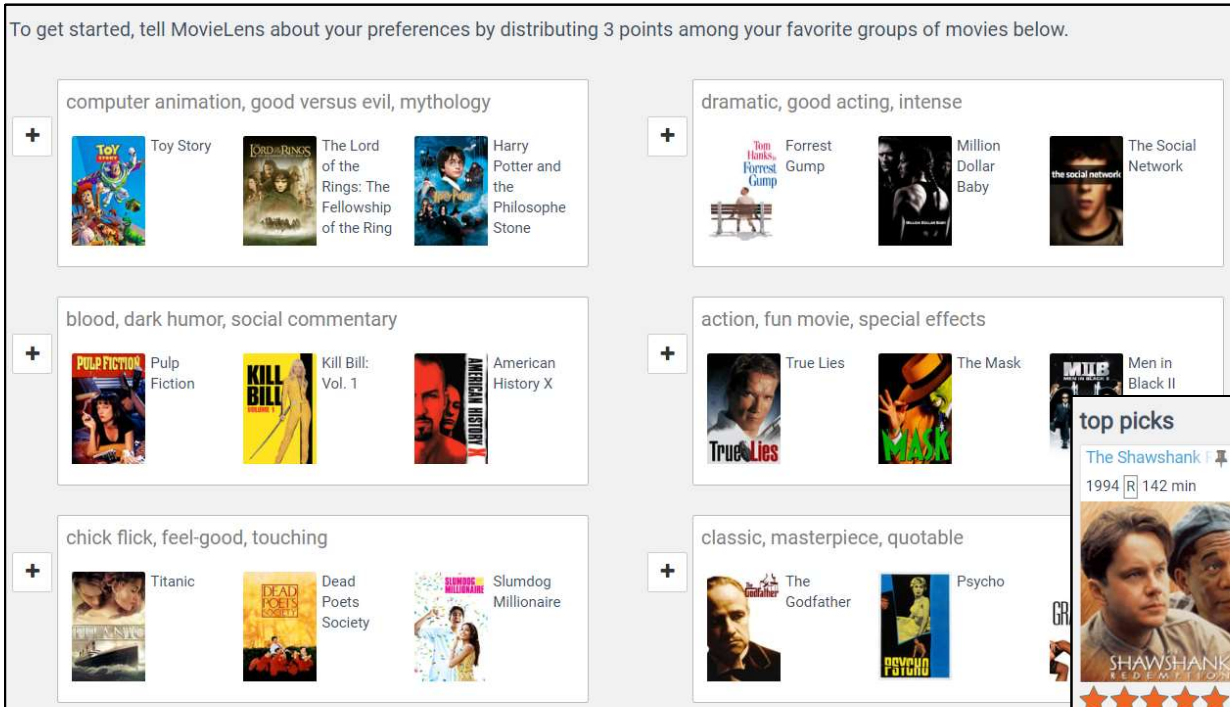
- ***Movielens*** = Ratings from 1 to 5 on movies (has 100K, 1M, 10M, 20M datasets)
- ***Jester*** = Ratings from -10 to + 10 on jokes (~6M ratings of 150 jokes)
- ***BookCrossing*** = Book Ratings from 1 to 10 (~1.1M ratings of 270K books)

Dataset	Users	Items	Ratings	Density	Rating Scale
Movielens 1M	6040	3883	1,000,209	4.26%	[1-5]
Movielens 10M	69,878	10,681	10,000,054	1.33%	[0.5-5]
Movielens 20M	138,493	27,278	20,000,263	0.52%	[0.5-5]
Jester	124,113	150	5,865,235	31.50%	[-10, 10]
Book-Crossing	92,107	271,379	1,031,175	0.0041%	[1, 10], and implicit
Last.fm	1892	17632	92,834	0.28%	Play Counts
Wikipedia	5,583,724	4,936,761	417,996,366	0.0015%	Interactions
OpenStreetMap (Azerbaijan)	231	108,330	205,774	0.82%	Interactions
Git (Django)	790	1757	13,165	0.95%	Interactions

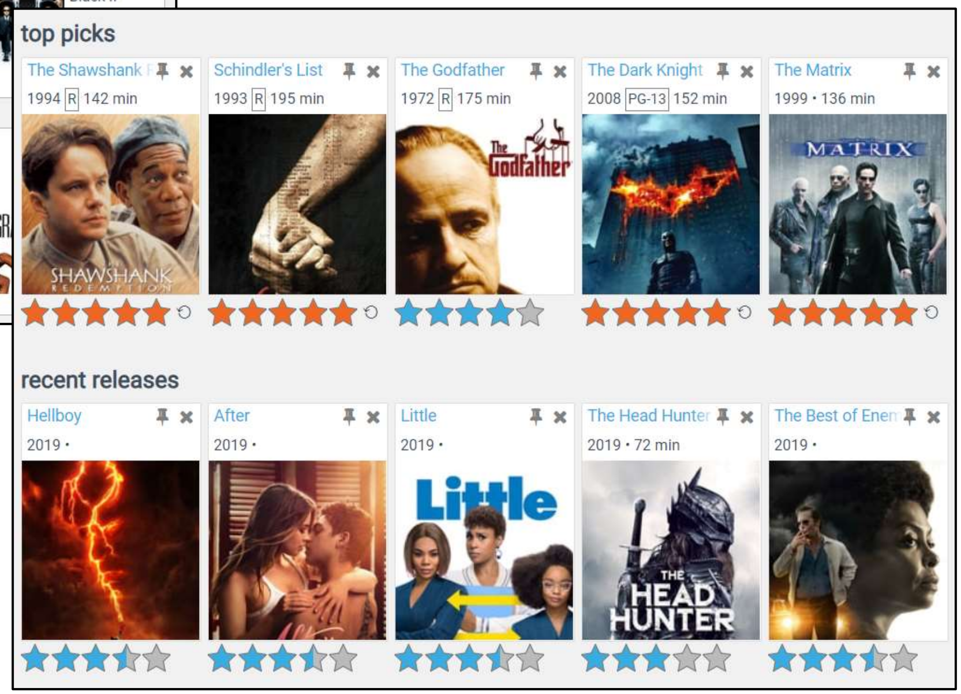
9 Must-Have Datasets for Investigating Recommender Systems

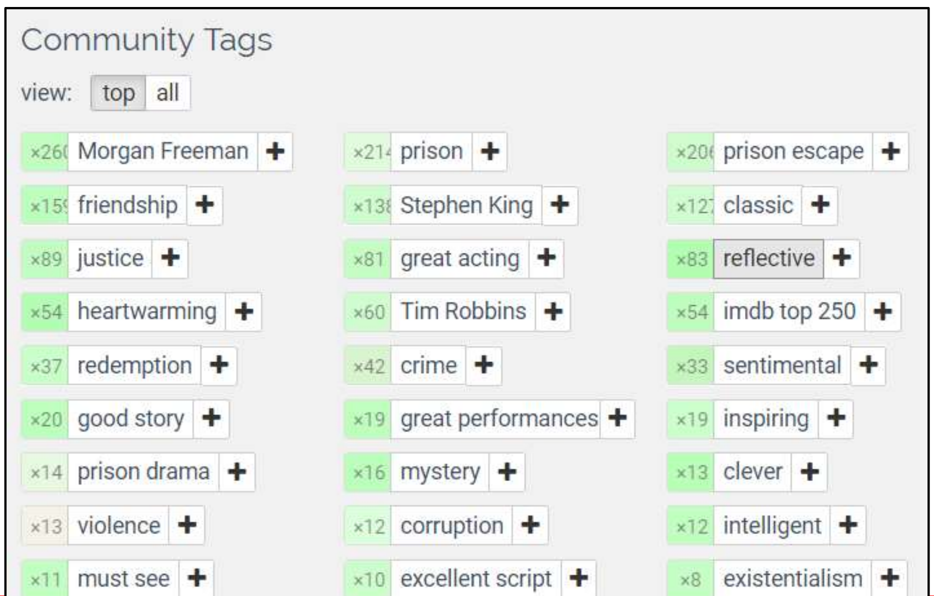
https://www.kdnuggets.com/2016/02/nine-datasets-investigating-recommender-systems.html

MovieLens Website



https://movielens.org/home





MovieLens Data Set (100K ratings)

- Each user has rated at least 20 movies
- The data is randomly ordered. Users & items are numbered consecutively from 1.
- Ratings are made on a 5-star scale (integer only)
- Timestamp is represented is seconds since 1/1/1970 UTC

UserID	movie	rating	datetime
1	61	4	878542420
1	189	3	888732928
1	33	4	878542699
1	160	4	875072547
1	20	4	887431883
1	202	5	875072442
1	171	5	889751711
1	265	4	878542441

Ratings file

movie id	movie name	Action	Adventure	Animation	Children's
1	Toy Story (1995)	0	0	1	1
2	GoldenEye (1995)	1	1	0	0
3	Four Rooms (1995)	0	0	0	0
4	Get Shorty (1995)	1	0	0	0
5	Copycat (1995)	0	0	0	0
6	Shanghai Triad (Yac	0	0	0	0
7	Twelve Monkeys (1	0	0	0	0
8	Babe (1995)	0	0	0	1
9	Dead Man Walking	0	0	0	0

Movie names & Genres

userid	age	gender	occupation	zip code
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213
6	42	M	executive	98101

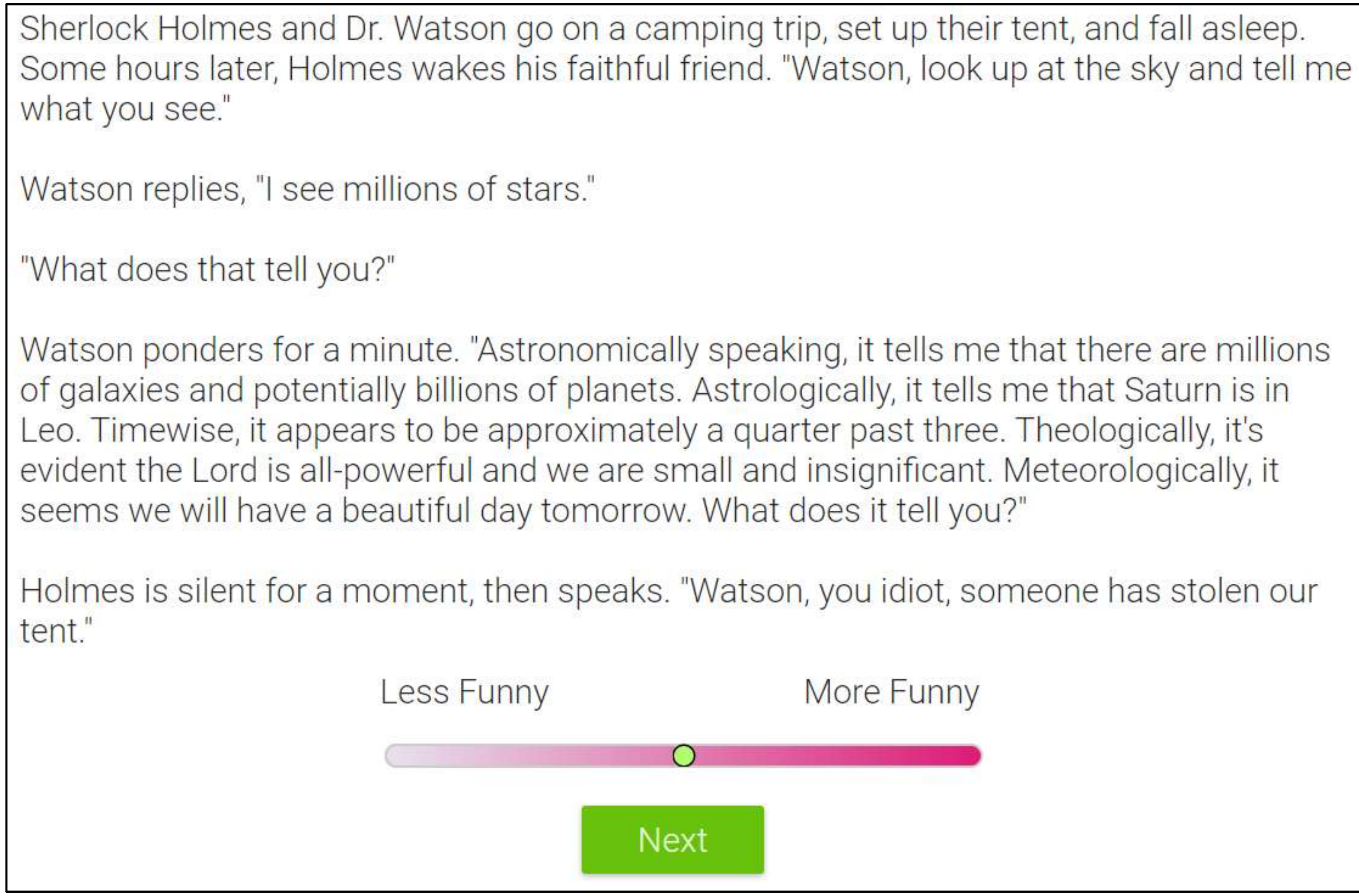
User demographics

https://grouplens.org/datasets/

Jester Dataset

- Data collected from a joke recommendation website
  - Ratings of 100 jokes collected between Apr’99->May’03
  - Ratings are real values from -10.00 to +10.00
- Dataset1 (matrix format)
  - 24,983 users who have rated 36 or more jokes
  - Cells containing "99" correspond to "not rated"
- Dataset2 (transaction format)
  - 23,500 users who have rated 36 or more jokes

http://eigentaste.berkeley.edu/ *…..the recommender system*



http://eigentaste.berkeley.edu/dataset/ *….the datasets*

Book Crossings Dataset

- Book Ratings from 1 to 10 (~1.1M ratings of 270K books)
  - Ratings = 0 are implicit (imply the user read the book) – we ignore these for now
- There are 3 files with this dataset but we use only the ratings file for this workshop
  1. BX-Book-Ratings ~ the book ratings: *User.ID, ISBN, Book.Rating (transaction format)*
  2. BX-Users ~ demographic info: *UserID, Location, Age* (but many fields are blank)
  3. BX-Books ~ content info: *Title, Author, Publication year, Publisher*

```
In [381]: trans = pd.read_csv("BX-Book-Ratings.csv", sep=';', error_bad_lines=False,
encoding="latin-1")

In [382]: trans
Out[382]:
         User-ID         ISBN  Book-Rating
0         276725   034545104X            0
1         276726   0155061224            5
2         276727   0446520802            0
3         276729   052165615X            3
4         276729   0521795028            6
...          ...          ...          ...
1149775   276704   1563526298            9
1149776   276706   0679447156            0
1149777   276709   0515107662           10
1149778   276721   0590442449           10
1149779   276723  05162443314            8

[1149780 rows x 3 columns]
```

Also see
https://www.bookcrossing.com/
http://www2.informatik.uni-freiburg.de/~cziegler/BX/

Some Benchmark Results

- See https://www.librec.net/release/v1.3/example.html

Rating Prediction: MovieLens 1M, 100K, Epinions, FilmTrust, Ciao, Flixster;

Item Ranking: MovieLens 100K, Epinions, Flixster, FilmTrust, Ciao;

MovieLens (100K)

Algorithm	MAE			RMSE		
	MMLite	PREA	LibRec	MMLite	PREA	LibRec
GlobalAvg	0.945	0.949	0.945	1.126	1.128	1.126
UserAvg	0.835	0.838	0.835	1.041	1.043	1.042
ItemAvg	0.817	0.823	0.817	1.024	1.030	1.025
PD	N/A	N/A	0.794	N/A	N/A	1.094
	sigma=2.5					
UserKNN	0.721	0.732	0.737	0.921	0.937	0.944
	neighbors=60, shrinkage=25, similarity=pcc; MMLite: reg_u=12, reg_i=1					
ItemKNN	0.703	0.716	0.723	0.899	0.914	0.924
	neighbors=40, shrinkage=2500, similarity=pcc; MMLite: reg_u=12, reg_i=1					
SlopeOne	0.739	0.740	0.739	0.939	0.940	0.940
RegSVD	0.741	0.730	0.730	0.949	0.932	0.936
	factors=10, reg=0.05, learn.rate=0.005, max.iter=100					
BiasedMF	0.724	N/A	0.722	0.918	N/A	0.918

Rating Prediction

MovieLens (100K)

Algo	Prec@5		Prec@10		Recall@5	
	MMLite	LibRec	MMLite	LibRec	MMLite	LibRec
MostPop	0.212	0.211	0.192	0.190	0.071	0.070
ItemKNN	0.314	0.318	0.279	0.260	0.096	0.103
	neighbors=80, similarity=cos, shrinkage=50, thresold=-1					
UserKNN	0.397	0.338	0.334	0.280	0.138	0.116
	neighbors=80, similarity=cos, shrinkage=50, thresold=-1					
BPR	0.358	0.378	0.309	0.321	0.247	0.129
	factors=10, reg=0.01, learn.rate=0.05, max.iter=30					
WRMF	0.416	0.424	0.353	0.358	0.142	0.149
	alpha=1.0, factors=20, reg=0.015, max.iter=10					

Item Ranking

Workshop2: Files

Upload all to your Google drive

- BX-Book-Ratings.csv
- BX-Books.csv
- jester_jokes.csv
- jester_ratings.dat
- simplemovies-transactions.csv
- u_data.csv
- u_item.csv

Data

- workshop2-basicCF.ipynb

Colab code

Workshop3: Surprise Library & Matrix Factorisation

- Surprise is a Python scikit for building and analyzing recommender systems that deal with <u>explicit rating data</u>
- Stores ratings data in a sparse matrix format

surprise

A Python scikit for recommender systems.

(3A) Use Surprise to perform User-Based and Item-Based CF

- Explore performance on MovieLens, Jester, Bookcrossings datasets
- Compare the best MAE with the results from workshop2

(3B) Use Surprise to perform Matrix Factorisation

- Build a models using Matrix Factorisation
- Compare the best MAE with the results from workshop2 and workshop3A

Surprise: Algorithms Summary

`random_pred.NormalPredictor`	Algorithm predicting a random rating based on the distribution of the training set, which is assumed to be normal.
`baseline_only.BaselineOnly`	Algorithm predicting the baseline estimate for given user and item.
`knns.KNNBasic`	A basic collaborative filtering algorithm.
`knns.KNNWithMeans`	A basic collaborative filtering algorithm, taking into account the mean ratings of each user.
`knns.KNNWithZScore`	A basic collaborative filtering algorithm, taking into account
`knns.KNNBaseline`	A basic collaborative filtering algorithm taking into account a *baseline* rating.
`matrix_factorization.SVD`	The famous *SVD* algorithm, as popularized by Simon Funk during the Netflix Prize.When baselines are not used, this
`matrix_factorization.SVDpp`	The *SVD++* algorithm, an extension of `SVD` taking into account implicit ratings.
`matrix_factorization.NMF`	A collaborative filtering algorithm based on Non-negative Matrix Factorization.
`slope_one.SlopeOne`	A simple yet accurate collaborative filtering algorithm.
`co_clustering.CoClustering`	A collaborative filtering algorithm based on co-clustering.

https://surprise.readthedocs.io/en/stable/

Surprise: KNN Algorithms

k-NN inspired algorithms

These are algorithms that are directly derived from a basic nearest neighbors approach.

Note

For each of these algorithms, the actual number of neighbors that are aggregated to compute an estimation is necessarily less than or equal to k. First, there might just not exist enough neighbors and second, the sets $N_i^k(u)$ and $N_u^k(i)$ only include neighbors for which the similarity measure is **positive**. It would make no sense to aggregate ratings from users (or items) that are negatively correlated. For a given prediction, the actual number of neighbors can be retrieved in the `'actual_k'` field of the `details` dictionary of the `prediction`.

Surprise: User-based & Item-based CF

class `surprise.prediction_algorithms.knns.KNNBasic`(*k=40*, *min_k=1*, *sim_options={}*, *verbose=True*, ***kwargs*)

A basic collaborative filtering algorithm.

The prediction $\hat{r}_{ui}$ is set as:

User-based

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$

Item-based

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i, j) \cdot r_{uj}}{\sum\limits_{j \in N_u^k(i)} \text{sim}(i, j)}$$

Note: unlike in the *demolib.py* library, Surprise item-based CF computes a predicted rating for an item using only the k nearest seen items to that item

Parameters:

- **k** (*int*) – The (max) number of neighbors to take into account for aggregation (see this note). Default is `40`.
- **min_k** (*int*) – The minimum number of neighbors to take into account for aggregation. If there are not enough neighbors, the prediction is set to the global mean of all ratings. Default is `1`.
- **sim_options** (*dict*) – A dictionary of options for the similarity measure. See Similarity measure configuration for accepted options.
- **verbose** (*bool*) – Whether to print trace messages of bias estimation, similarity, etc. Default is True.

Set user-based = True or False in sim_options

```
# compute  similarities between items

sim_options = {'name': 'cosine',
               'user_based': False
               }
algo = KNNBasic(sim_options=sim_options)
```

Surprise: User-based & Item-based CF

Similarity measure configuration

Many algorithms use a similarity measure to estimate a rating. The way they can be configured is done in a similar fashion as for baseline ratings: you just need to pass a `sim_options` argument at the creation of an algorithm. This argument is a dictionary with the following (all optional) keys:

- `'name'`: The name of the similarity to use, as defined in the `similarities` module. Default is `'MSD'`.
- `'user_based'`: Whether similarities will be computed between users or between items. This has a **huge** impact on the performance of a prediction algorithm. Default is `True`.
- `'min_support'`: The minimum number of common items (when `'user_based'` is `'True'`) or minimum number of common users (when `'user_based'` is `'False'`) for the similarity not to be zero. Simply put, if $|I_{uv}| < \text{min\_support}$ then $\text{sim}(u, v) = 0$. The same goes for items.
- `'shrinkage'`: Shrinkage parameter to apply (only relevant for `pearson_baseline` similarity). Default is 100.

Surprise: Similarity Measures

Available similarity measures:

`cosine`	Compute the cosine similarity between all pairs of users (or items).
`msd`	Compute the Mean Squared Difference similarity between all pairs of users (or items).
`pearson`	Compute the Pearson correlation coefficient between all pairs of users (or items).
`pearson_baseline`	Compute the (shrunk) Pearson correlation coefficient between all pairs of users (or items) using baselines for centering instead of means.

MSD Similarity (Euclidean)

`surprise.similarities.msd()`

Only **common** users (or items) are taken into account. The Mean Squared Difference is defined as:

$$\text{msd}(u, v) = \frac{1}{|I_{uv}|} \cdot \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2$$

or

$$\text{msd}(i, j) = \frac{1}{|U_{ij}|} \cdot \sum_{u \in U_{ij}} (r_{ui} - r_{uj})^2$$

depending on the `user_based` field of `sim_options` (see Similarity measure configuration).

The MSD-similarity is then defined as:

$$\text{msd\_sim}(u, v) = \frac{1}{\text{msd}(u, v) + 1}$$
$$\text{msd\_sim}(i, j) = \frac{1}{\text{msd}(i, j) + 1}$$

The $+1$ term is just here to avoid dividing by zero.

Note: msd ~ Euclidean as defined in demolib

Pearson Similarity

`surprise.similarities.pearson()`

Compute the Pearson correlation coefficient between all pairs of users (or items).

Only **common** users (or items) are taken into account. The Pearson correlation coefficient can be seen as a mean-centered cosine similarity, and is defined as:

$$\text{pearson\_sim}(u, v) = \frac{\sum\limits_{i \in I_{uv}} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$$

or

$$\text{pearson\_sim}(i, j) = \frac{\sum\limits_{u \in U_{ij}} (r_{ui} - \mu_i) \cdot (r_{uj} - \mu_j)}{\sqrt{\sum\limits_{u \in U_{ij}} (r_{ui} - \mu_i)^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} (r_{uj} - \mu_j)^2}}$$

depending on the `user_based` field of `sim_options` (see Similarity measure configuration).

Note: if there are no common users or items, similarity will be 0 (and not -1).

Surprise Matrix Factorisation : SVD

class `surprise.prediction_algorithms.matrix_factorization.SVD`

The famous *SVD* algorithm, as popularized by Simon Funk during the Netflix Prize. When baselines are not used, this is equivalent to Probabilistic Matrix Factorization [SM08] (see note below).

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

p ~ user properties matrix

q ~ item properties matrix

If user u is unknown, then the bias b_u and the factors p_u are assumed to be zero. The same applies for item i with b_i and q_i.

To estimate all the unknown, we minimize the following regularized squared error:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right)$$

The minimization is performed by a very straightforward stochastic gradient descent:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \end{aligned}$$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$. These steps are performed over all the ratings of the trainset and repeated `n_epochs` times. Baselines are initialized to `0`. User and item factors are randomly initialized according to a normal distribution.

You also have control over the learning rate γ and the regularization term λ. Both can be different for each kind of parameter (see below). By default, learning rates are set to `0.005` and regularization terms are set to `0.02`.

Surprise: SVD Parameters

- **n_factors** – The number of factors. Default is `100`.
- **n_epochs** – The number of iteration of the SGD procedure. Default is `20`.
- **biased** (*bool*) – Whether to use baselines (or biases). See note above. Default is `True`.
- **init_mean** – The mean of the normal distribution for factor vectors initialization. Default is `0`.
- **init_std_dev** – The standard deviation of the normal distribution for factor vectors initialization. Default is `0.1`.
- **lr_all** – The learning rate for all parameters. Default is `0.005`.
- **reg_all** – The regularization term for all parameters. Default is `0.02`.

- **lr_bu** – The learning rate for b_u.
- **lr_bi** – The learning rate for b_i.
- **lr_pu** – The learning rate for p_u.
- **lr_qi** – The learning rate for q_i.

- **reg_bu** – The regularization term for b_u.
- **reg_bi** – The regularization term for b_i.
- **reg_pu** – The regularization term for p_u.
- **reg_qi** – The regularization term for q_i.

You can choose to use an unbiased version of this algorithm, simply predicting:

$$\hat{r}_{ui} = q_i^T p_u$$

This is equivalent to Probabilistic Matrix Factorization ([SM08], section 2) and can be achieved by setting the `biased` parameter to `False`.

- **random_state** (int, RandomState instance from numpy, or `None`) – Determines the RNG that will be used for initialization. If int, `random_state` will be used as a seed for a new RNG. This is useful to get the same initialization over multiple calls to `fit()`. If RandomState instance, this same instance is used as RNG. If `None`, the current RNG from numpy is used. Default is `None`.

Precision at K (P@K)

- P@K is the proportion of recommended items in the top-K set that are relevant
- An item is relevant if the user is known to like it (e.g. actual rating >=4)

<u>E.g. for K = 10:</u>

As ordered by the recommender system

Item ID's for the top 10 recommended items:

Test user1	29	3	102	56	38	120	12	18	83	33	p@k = 3/10 = 0.3

Items that the user has liked (e.g. rated >= T)

This user has only 8 unrated items:

Test user2	56	18	12	43	5	19	77	39	-	-	p@k = 4/8 = 0.5

**When computing p@k for a user from a test set, we remove from the recommended item list all items not in the test set for that user (these are the items that we do not know the actual rating for). For users with few ratings this may cause issues.*

https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

Mean Average Precision at K (MAP@K)

P@K

> How many relevant items are present in the top-k recommendations of your system

For example, to calculate P@3: *take the top 3 recommendations for a given user and check how many of them are good ones. That number divided by 3 gives you the P@3*

AP@K

> The mean of *P@i* for *i=1, ..., K*.

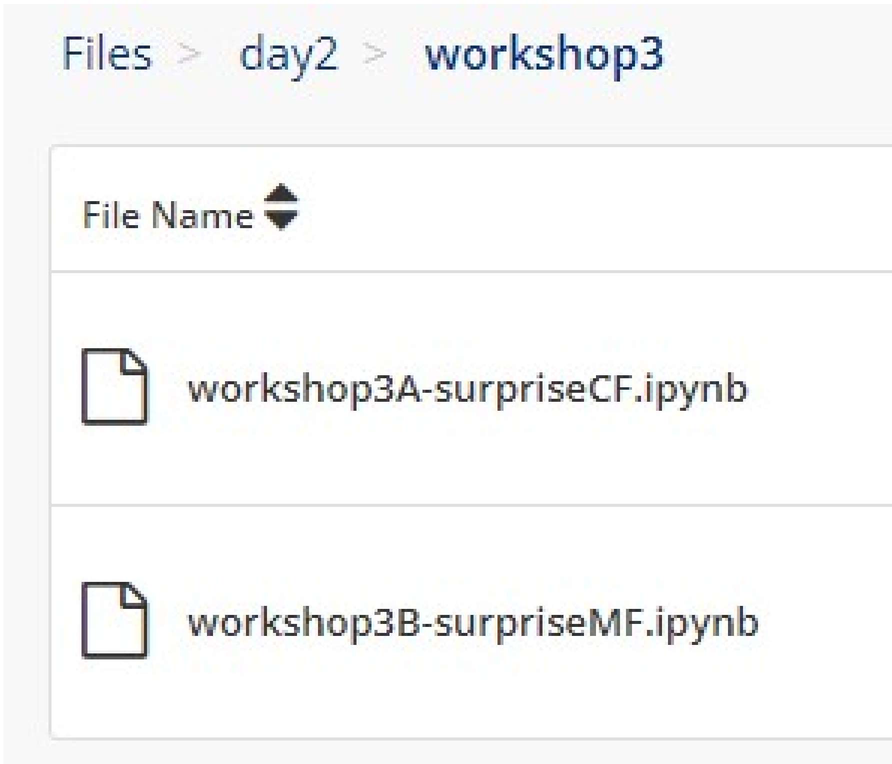
For example, to calculate AP@3: *sum P@1, P@2 and P@3 and divide that value by 3*

AP@K is typically calculated for one user.

MAP@K

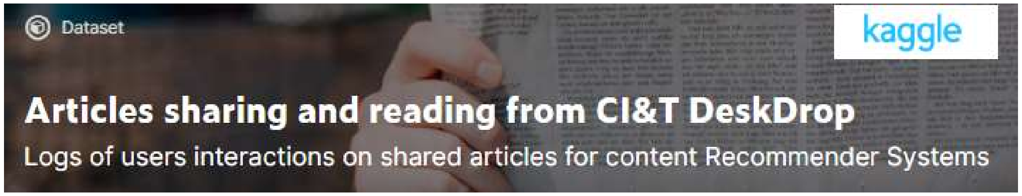
> The mean of the AP@K for all the users.

Workshop3 Files



Upload all to your Google drive

Workshop4 – Handling Implicit Ratings



https://www.kaggle.com/gspmoreira/articles-sharing-reading-from-cit-deskdrop

- Deskdrop is a platform that allows employees to share relevant articles with their peers, and collaborate around them. All users must login to use the platform.
- The logged data (12 months) includes 73K user interactions on 3K public articles that are shared on the platform, including the type of interaction

The eventType values are:

- **VIEW**: The user has opened the article. A page view in a content site can mean many things. It can mean that the user is interested, or maybe user is just lost or clicking randomly.
- **LIKE**: The user has liked the article.
- **BOOKMARK**: The user has bookmarked the article for easy return in the future. This is a strong indication that the user finds something of interest.
- **COMMENT CREATED**: The user left a comment on the article.
- **FOLLOW**: The user chose to be notified on any new comment about the article.

We create an integer implicit ratings variable:

```
event_type_strength = {
    'VIEW': 1.0,
    'LIKE': 2.0,
    'BOOKMARK': 3.0,
    'FOLLOW': 4.0,
    'COMMENT CREATED': 5.0,
}
```

Workshop4: Using pySpark ALS algorithm

Can handle explicit ratings and implicit preference data. For implicit data, the algorithm used is based on [Collaborative Filtering for Implicit Feedback Datasets]

In this workshop we compare explicit versus implicit factorization on the same dataset

class **pyspark.ml.recommendation.ALS**(*, rank=***10***, maxIter=***10***, regParam=***0.1***, numUserBlocks=***10***, numItemBlocks=***10***, implicitPrefs=***False***, alpha=***1.0***, userCol=***'user'***, itemCol=***'item'***, seed=***None***, ratingCol=***'rating'***, nonnegative=***False***, checkpointInterval=***10***, intermediateStorageLevel=***'MEMORY_AND_DISK'***, finalStorageLevel=***'MEMORY_AND_DISK'***, coldStartStrategy=***'nan'***, blockSize=***4096***)

Example code:

```
train_df = spark.createDataFrame([(0, 0, 4.0), (0, 1, 2.0), (1, 1, 3.0), (1, 2, 4.0), (2, 1, 1.0), (2, 2, 5.0)],["user", "item", "rating"])
als = ALS(rank=10, seed=0)
model = als.fit(train_df)
predictions = model.transform(test_df)
```

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.recommendation.ALS.html

Workshop4 Files

- shared_articles.csv
- users_interactions.csv
- workshop4-pysparkMF.ipynb

Upload all to your Google drive

Workshop5: Neural Collaborative Filtering

- Build and test the simple two tower architecture using the Keras library

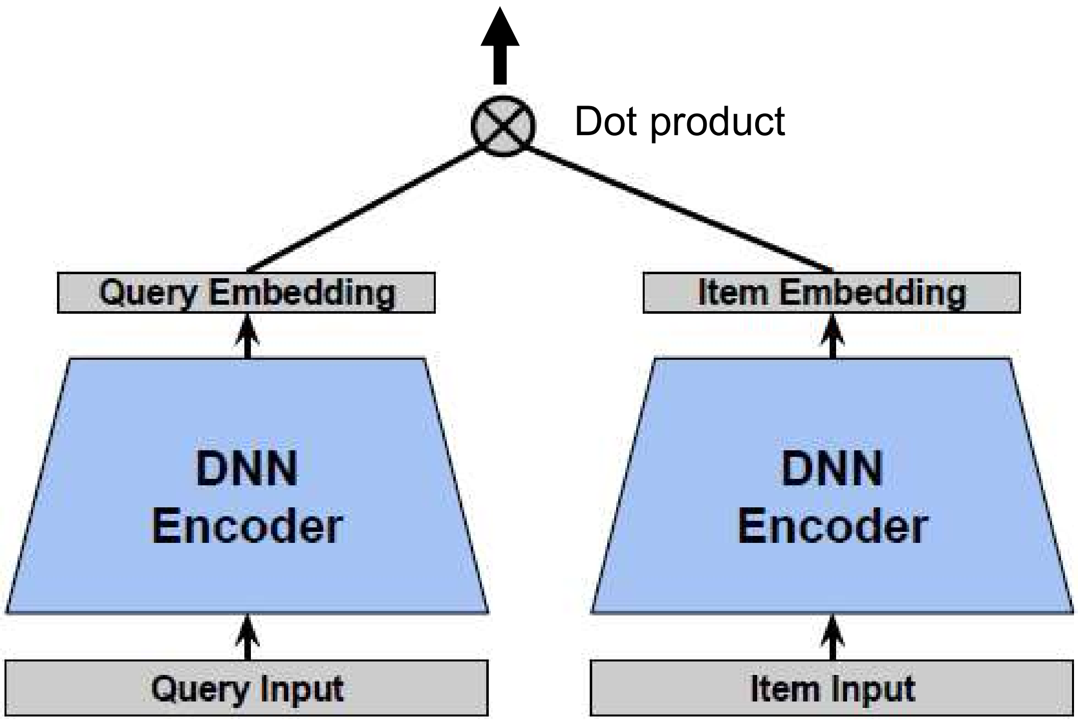
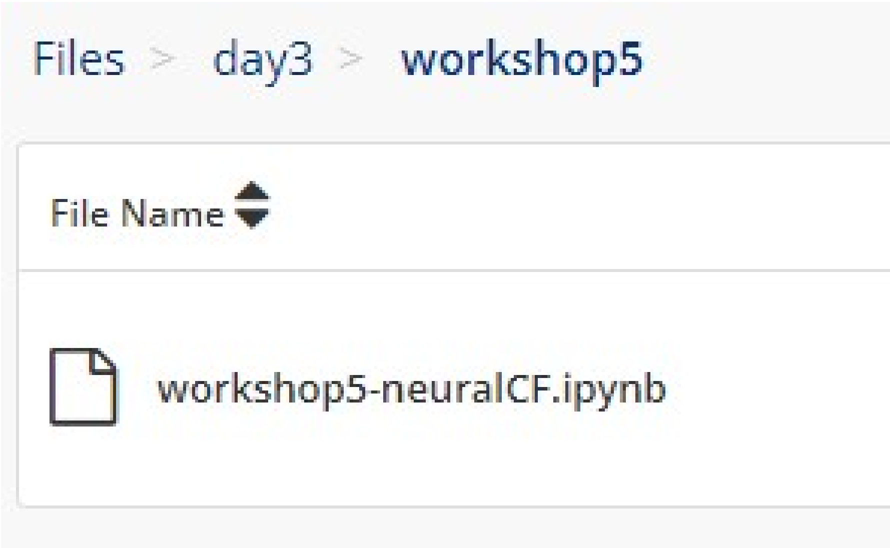


Figure 1: Two-tower Neural Network.

Workshop5: Files



Upload to Google drive