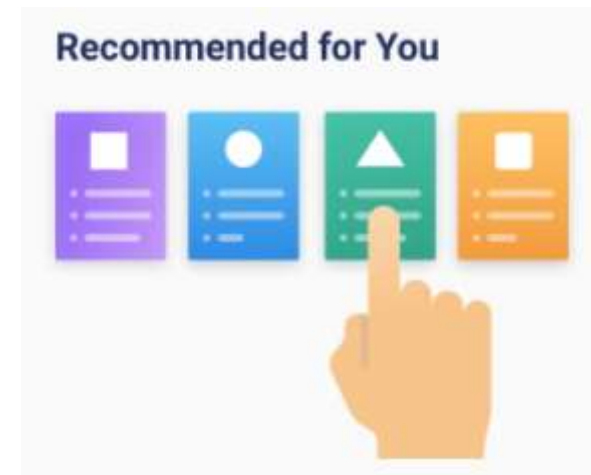


## Graduate Certificate in Intelligent Reasoning Systems

# Reasoning and Knowledge Discovery from (large) Datasets

## Recommender System Workshops



Dr. Barry Shepherd  
Institute of Systems Science  
National University of Singapore  
Email: [barryshepherd@nus.edu.sg](mailto:barryshepherd@nus.edu.sg)

© 2021 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

# 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](http://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:

- r1) A => B, cf = 0.4
- r2) B => F, cf = 0.6
- r3) C => D, cf = 0.3
- r4) D => B, cf = 0.8
- r5) B, C => D, cf = 0.5
- r6) A, B => E, cf = 0.4



## Predictions:

- B cf = 0.4
- F cf = 0.6
- D cf = 0.3
- D cf = 0.5
- E cf = 0.4



## Recommendations:

- F, cf = 0.6
- D, cf = 0.5



Hits = 1

- 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

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

output

# Workshop1: Files

File	Description
apriori-workshop.py	Sample code to walk through
apriori-lib.py	Code for generating association rules
apriori-testing.py	Code for executing and testing association rules
Groceries_dataset.csv	Kaggle groceries dataset
anonymous-msweb-transactions.txt	Microsoft website browsing dataset
baskets.txt	Groceries dataset that includes user demographics

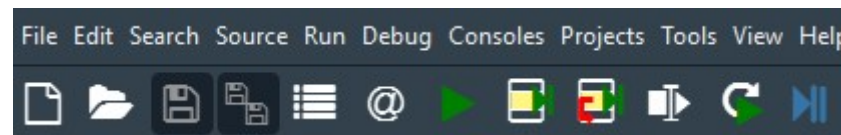
(1) Run Spyder and open the .py files



(2) select the *apriori-lib.py* tab and run the file, repeat with *apriori-testing.py*



(3) select the *apriori-workshop.py* tab and then single step through the code by running each line separately



runs the file




runs current line  
(where the cursor is)

# Workshop2: Collaborative Filtering

- Explore some simple code that implements User-based and Item-based Collaborative Filtering and also implements basic testing
- I will walk you through the code using a simple dataset
- Your task:
  - Apply the code to 3 larger datasets:
    - Movielens (movie ratings)
    - Jester (joke ratings)
    - Bookcrossings (book ratings) - if you have time!
  - 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

To get started, tell MovieLens about your preferences by distributing 3 points among your favorite groups of movies below.

computer animation, good versus evil, mythology

+

Toy Story
 The Lord of the Rings: The Fellowship of the Ring
 Harry Potter and the Philosopher's Stone

dramatic, good acting, intense

+

Forrest Gump
 Million Dollar Baby
 The Social Network

blood, dark humor, social commentary

+

Pulp Fiction
 Kill Bill: Vol. 1
 American History X

action, fun movie, special effects

+

True Lies
 The Mask
 Men in Black II

chick flick, feel-good, touching

+

Titanic
 Dead Poets Society
 Slumdog Millionaire

classic, masterpiece, quotable

+

The Godfather
 Psycho

<https://movielens.org/home>

## Community Tags

view:

x260 Morgan Freeman +	x214 prison +	x206 prison escape +
x158 friendship +	x138 Stephen King +	x121 classic +
x89 justice +	x81 great acting +	x83 reflective +
x54 heartwarming +	x60 Tim Robbins +	x54 imdb top 250 +
x37 redemption +	x42 crime +	x33 sentimental +
x20 good story +	x19 great performances +	x19 inspiring +
x14 prison drama +	x16 mystery +	x13 clever +
x13 violence +	x12 corruption +	x12 intelligent +
x11 must see +	x10 excellent script +	x8 existentialism +

## top picks

The Shawshank Redemption

1994 R 142 min

★★★★★

Schindler's List

1993 R 195 min

★★★★★

The Godfather

1972 R 175 min

★★★★☆

The Dark Knight

2008 PG-13 152 min

★★★★★

The Matrix

1999 136 min

★★★★★

## recent releases

Hellboy

2019

★★★★☆

After

2019

★★★★☆

Little

2019

★★★★☆

The Head Hunter

2019 • 72 min

★★★★☆

The Best of Enemies

2019

★★★★☆



# 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 in 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	Copcat (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 ("99" corresponds to "not rated")
- Dataset1 (tabular format)
  - 24,983 users who have rated 36 or more jokes
- Dataset2 (transaction format)
  - 23,500 users who have rated 36 or more jokes

<http://eigentaste.berkeley.edu/> .....the recommender system

Sherlock Holmes and Dr. Watson go on a camping trip, set up their tent, and fall asleep. Some hours later, Holmes wakes his faithful friend. "Watson, look up at the sky and tell me what you see."

Watson replies, "I see millions of stars."

"What does that tell you?"

Watson ponders for a minute. "Astronomically speaking, it tells me that there are millions of galaxies and potentially billions of planets. Astrologically, it tells me that Saturn is in Leo. Timewise, it appears to be approximately a quarter past three. Theologically, it's evident the Lord is all-powerful and we are small and insignificant. Meteorologically, it seems we will have a beautiful day tomorrow. What does it tell you?"

Holmes is silent for a moment, then speaks. "Watson, you idiot, someone has stolen our tent."



<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/~ciegler/BX/>

# Testing Rating Predictions

- The demo code splits the data into training & test sets by randomly assigning individual ratings to the test set, e.g. assign  $\sim 30\%$  to the test set.
- This is an easy to implement variant of the holdout test scheme. Also it supports matrix factorisation testing in the next workshop (MF cannot handle cold-start users, more about this later)

All data

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
u1		3	4.5		2	4		5		1
u2	4	3.5		2	3		5		4.5	
u3		4	3		2	2.5	5		4	4

 =Test set

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
u1		3			2	4				1
u2				2	3				4.5	
u3		4	3			2.5	5			4

Train set (after deleting test ratings)

Test set

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
u1			4.5					5		
u2	4	3.5					5			
u3					2				4	

# Workshop2: Files

File	Contains
demolib.py	<ul style="list-style-type: none"> <li>• Simple implementation of User-Based and Item-Based CF</li> <li>• Employs a non-optimised data representation (simple array)</li> </ul>
CFworkshop.py	<ul style="list-style-type: none"> <li>• Demo code to apply the above functions to the “Toby” dataset and also provide a code framework for all of the workshop2 datasets</li> </ul>
simplemovies-transactions.csv	<ul style="list-style-type: none"> <li>• The “Toby” dataset</li> </ul>
u_data.csv	<ul style="list-style-type: none"> <li>• Movielens 100K dataset</li> </ul>
u_item.csv	<ul style="list-style-type: none"> <li>• Mapping of movieID to actual movie name (and release date , genre)</li> </ul>
jester_ratings.txt	<ul style="list-style-type: none"> <li>• Jester dataset2</li> </ul>
BX-Book-Ratings.csv	<ul style="list-style-type: none"> <li>• The book crossings dataset</li> </ul>

Start by loading and running: `demolib.py`  
 Then follow the instructions in `CFworkshop.py`

# 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, threshd=-1					
UserKNN	0.397	0.338	0.334	0.280	0.138	0.116
	neighbors=80, similarity=cos, shrinkage=50, threshd=-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					

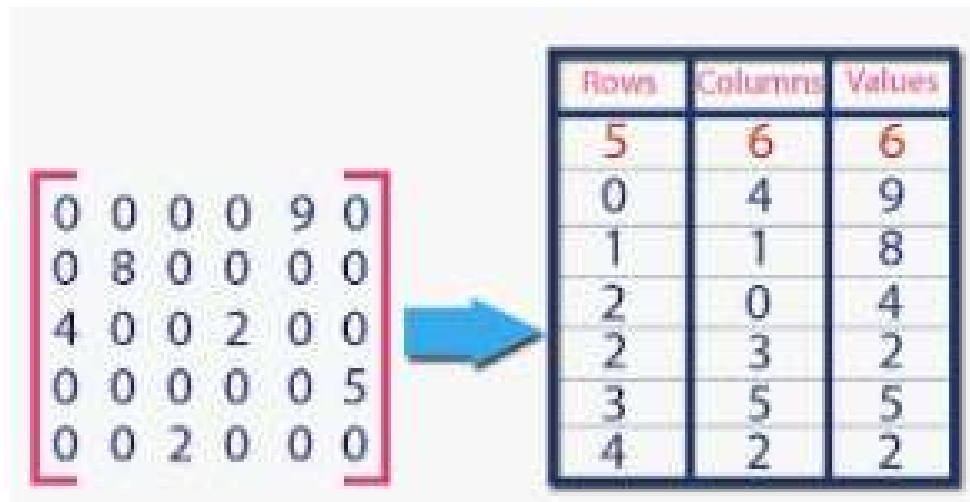
Item Ranking



# Workshop3: Introduction to Surprise

Surprise is a Python scikit for building and analyzing recommender systems that deal with explicit rating data

Stores ratings data in a sparse matrix format, which is a much more efficient representation for very sparse data since only the non-zero data is actually stored



Storing ratings in an uncompressed matrix is impractical if the data is very sparse:

- In the BX dataset there are 270K books and 92K users
- Matrix size = 270K \* 92K ~ 24Billion cells, yet only ~ 1M ratings are given (0.004%)
- Trying to create the full matrix fails... not enough memory



# Surprise: Algorithms Summary

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

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

or

Item-  
based

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

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`.

depending on the `user based` field of the `sim_options` parameter.

```
# 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:

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

or

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

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).



# Workshop3: Your Task\*

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

Do this ***anytime after workshop2***

- Load Movielens , Divide into train & test events
- Perform User-based and Item-based collaborative filtering
- Generate baseline estimates (good for cold start users)
- Compare the best MAE with the results from workshop2

(3B) Use Surprise to perform Matrix Factorisation on Explicit Ratings

Do this ***after the matrix factorisation lecture***

- Load BookCrossings data, use ALL explicit ratings (no need to sub-sample)
- Divide into train & test events
- Build a model using Matrix Factorisation
- Compare the best MAE with the results from workshop2

\*The workshop code: *surpriselib-workshop.py* contains detailed instructions.

# 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:

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

$p \sim$  user properties matrix  
 $q \sim$  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:

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

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  $\gamma$  and the regularization term  $\lambda$ . 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`.

# Surprise: SVD++

performing any rating (whether good or bad) yields extra (implicit) info

```
class surprise.prediction_algorithms.matrix_factorization.SVDpp
```

Bases: `surprise.prediction_algorithms.algo_base.AlgoBase`

The SVD++ algorithm, an extension of `svd` taking into account implicit ratings.

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

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left( p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$$

The user characteristics are adjusted by adding a factor vector that reflects the 'average' properties of the items that the user has viewed (likes to view)

Individual factor vectors for each item  $j$  rated by the user

Number of items rated by the user

Where the  $y_j$  terms are a new set of item factors that capture implicit ratings. Here, an implicit rating describes the fact that a user  $u$  rated an item  $j$ , regardless of the rating value.

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  $b_i$ ,  $q_i$  and  $y_i$ .

For details, see section 4 of [Koren:2008:FMN]. See also [Ricci:2010], section 5.3.1.

Just as for `svd`, the parameters are learned using a SGD on the regularized squared error objective.

Baselines are initialized to `0`. User and item factors are randomly initialized according to a normal distribution, which can be tuned using the `init_mean` and `init_std_dev` parameters.

You have control over the learning rate  $\gamma$  and the regularization term  $\lambda$ . 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`.



# Workshop4 – Implicit Ratings

- Explore the Implicit Library using
  - Matrix factorisation for Implicit data (ALS algorithm)
- Datasets
  - Movielens: I will give a demo walk-through
  - Bookcrossings: treat the book reads (ratings==0) as implicit ratings
  - Deskdrop (new): this creates an integer implicit rating based on a user's content viewing (more views => bigger rating etc). Just walk through the code by yourself



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

# DeskDrop Dataset

- Deskdrop is an internal communications platform that allows company employees to share relevant articles with their peers, and collaborate around them.
- Dataset size:
  - 12 months logs (Mar. 2016 - Feb. 2017) from DeskDrop with ~73k logged users interactions on more than 3k public articles shared in the platform.
- Dataset content:
  - Item attributes: Articles' original URL, title, and content plain text are available in two languages (English and Portuguese).
  - Contextual information: Context of the users visits, like date/time, client (mobile native app / browser) and geolocation.
  - Logged users: All users are required to login in the platform, providing a long-term tracking of users preferences (not depending on cookies in devices).
  - Rich implicit feedback: Different interaction types were logged, making it possible to infer the user's level of interest in the articles (eg. comments > likes > views).
  - Multi-platform: Users interactions were tracked in different platforms (web browsers and mobile native apps)

# Implicit Library – User Guide Extracts

## Implicit

Fast Python Collaborative Filtering for Implicit Datasets

This project provides fast Python implementations of several different popular recommendation algorithms for implicit feedback datasets:

- Alternating Least Squares as described in the papers [Collaborative Filtering for Implicit Feedback Datasets](#) and in [Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering](#).
- [Bayesian Personalized Ranking](#)
- Item-Item Nearest Neighbour models, using Cosine, TFIDF or BM25 as a distance metric

All models have multi-threaded training routines, using Cython and OpenMP to fit the models in parallel among all available CPU cores. In addition, the ALS and BPR models both have custom CUDA kernels - enabling fitting on compatible GPU's. This library also supports using approximate nearest neighbours libraries such as [Annoy](#), [NMSLIB](#) and [Faiss](#) for [speeding up making recommendations](#).

<https://implicit.readthedocs.io/en/latest/quickstart.html>



# Implicit Library – User Guide Extracts

## Basic Usage

```
import implicit

# initialize a model
model = implicit.als.AlternatingLeastSquares(factors=50)

# train the model on a sparse matrix of item/user/confidence weights
model.fit(item_user_data)

# recommend items for a user
user_items = item_user_data.T.tocsr()
recommendations = model.recommend(userid, user_items)

# find related items
related = model.similar_items(itemid)
```

Not ratings!

For **model building**, the data must be loaded into a *sparse.crs\_matrix*, with rows as items, columns as users, and values as the (implicit) ratings.

For **model testing**, the matrix must have rows as users and columns as items (to enhance algorithm efficiency)

# AlternatingLeastSquares

```
class implicit.als.AlternatingLeastSquares(factors=100, regularization=0.01, dtype=<type
'numpy.float32'>, use_native=True, use_cg=True, use_gpu=False, iterations=15, calculate_training_loss=False,
num_threads=0, random_state=None)
```

A Recommendation Model based off the algorithms described in the paper 'Collaborative Filtering for Implicit Feedback Datasets' with performance optimizations described in 'Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering.'

- Parameters:**
- **factors** (*int, optional*) – The number of latent factors to compute
  - **regularization** (*float, optional*) – The regularization factor to use
  - **dtype** (*data-type, optional*) – Specifies whether to generate 64 bit or 32 bit floating point factors
  - **use\_native** (*bool, optional*) – Use native extensions to speed up model fitting
  - **use\_cg** (*bool, optional*) – Use a faster Conjugate Gradient solver to calculate factors
  - **use\_gpu** (*bool, optional*) – Fit on the GPU if available, default is to run on GPU only if available
  - **iterations** (*int, optional*) – The number of ALS iterations to use when fitting data
  - **calculate\_training\_loss** (*bool, optional*) – Whether to log out the training loss at each iteration
  - **num\_threads** (*int, optional*) – The number of threads to use for fitting the model. This only applies for the native extensions. Specifying 0 means to default to the number of cores on the machine.
  - **random\_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

# BayesianPersonalizedRanking

```
class implicit.bpr.BayesianPersonalizedRanking
```

Bayesian Personalized Ranking

A recommender model that learns a matrix factorization embedding based off minimizing the pairwise ranking loss described in the paper [BPR: Bayesian Personalized Ranking from Implicit Feedback](#).

- Parameters:**
- **factors** (*int, optional*) – The number of latent factors to compute
  - **learning\_rate** (*float, optional*) – The learning rate to apply for SGD updates during training
  - **regularization** (*float, optional*) – The regularization factor to use
  - **dtype** (*data-type, optional*) – Specifies whether to generate 64 bit or 32 bit floating point factors
  - **use\_gpu** (*bool, optional*) – Fit on the GPU if available
  - **iterations** (*int, optional*) – The number of training epochs to use when fitting the data
  - **verify\_negative\_samples** (*bool, optional*) – When sampling negative items, check if the randomly picked negative item has actually been liked by the user. This check increases the time needed to train but usually leads to better predictions.
  - **num\_threads** (*int, optional*) – The number of threads to use for fitting the model. This only applies for the native extensions. Specifying 0 means to default to the number of cores on the machine.
  - **random\_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

```
explain(userid, user_items, itemid, user_weights=None, N=10)
```

Provides explanations for why the item is liked by the user.

**Parameters:**

- **userid** (*int*) - The userid to explain recommendations for
- **user\_items** (*csr\_matrix*) - Sparse matrix containing the liked items for the user
- **itemid** (*int*) - The itemid to explain recommendations for
- **user\_weights** (*ndarray, optional*) - Precomputed Cholesky decomposition of the weighted user liked items. Useful for speeding up repeated calls to this function, this value is returned
- **N** (*int, optional*) - The number of liked items to show the contribution for

**Returns:**

- **total\_score** (*float*) - The total predicted score for this user/item pair
- **top\_contributions** (*list*) - A list of the top N (itemid, score) contributions for this user/item pair
- **user\_weights** (*ndarray*) - A factorized representation of the user. Passing this in to future 'explain' calls will lead to noticeable speedups



# Workshop4: Files

File	Contains
implicit-utils.py*	<ul style="list-style-type: none"> <li>Simple utilities to help with the workshop. Execute this file first</li> </ul>
implicit-workshop.py	<ul style="list-style-type: none"> <li>Demo code for processing movielens and bookcrossings</li> </ul>
implicit-demo-deskdrop.py	<ul style="list-style-type: none"> <li>Demo code for processing the deskdrop dataset</li> </ul>
users_interactions.csv	<ul style="list-style-type: none"> <li>The deskdrop user events data</li> </ul>
shared_articles.csv	<ul style="list-style-type: none"> <li>The deskdrop articles metadata</li> </ul>

\*Note: if at any time you get the python error message: *restarting kernel*, then you will need to re-execute 'implicit-utils.py' and start again from the beginning