Song Recommendation System

By: Shefali Khatri and Jason Miller

Background

- **Problem:** How to identify songs to recommend to users that they will enjoy?
  - Predictive
  - Supervised
- **Data used:** Million Songs Subset Dataset and Echo Nest Taste Profile
- **Why?**
  - Increased user engagement
  - Better user experience
  - Decreased customer attrition
  - Increased revenue generation

Challenges

- **Cold-start problem**
  - What do you do when you have a new user?
- **Implicit Data**
  - What do you do when there is no measure of user preference (i.e. ratings)?
- **Lack of Domain Knowledge**
- **How do you split your Train and Test Set?**
  - How do you split a user-item interaction matrix?
    - Item-user interaction matrix is used to fit model, but the same data represented as user-item interaction matrix is used for predict for a given user id
  - How do you split raw data using song as target feature?
  - User must exist in the train set and test set to be able to generate a prediction

Echo Nest Taste Profile

[4]:

	user	song	count
0	fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOBONKR12A58A7A7E0	1
1	fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOEGIYH12A6D4FC0E3	1
2	fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOFLJQZ12A6D4FADA6	1
3	fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOHTKMO12AB01843B0	1
4	fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SODQZCY12A6D4F9D11	1
...	...	...	...
772657	7eaf05ee4d1e2a3489ccd59d39d49f6712c61dbc	SOHFGKG12A6701C429	1
772658	7eaf05ee4d1e2a3489ccd59d39d49f6712c61dbc	SOZXSEC12A67020AB5	1
772659	7eaf05ee4d1e2a3489ccd59d39d49f6712c61dbc	SOSJRXV12A8C136E1B	1
772660	7eaf05ee4d1e2a3489ccd59d39d49f6712c61dbc	SOHLQRL12A6D4F71DE	1
772661	7eaf05ee4d1e2a3489ccd59d39d49f6712c61dbc	SOTFOAE12A6D4F4511	1

772662 rows × 3 columns

Million Songs Subset Dataset

- **Contains 10,000 songs, and 76 features including genres, audio data, beats, key, tempo, and more**

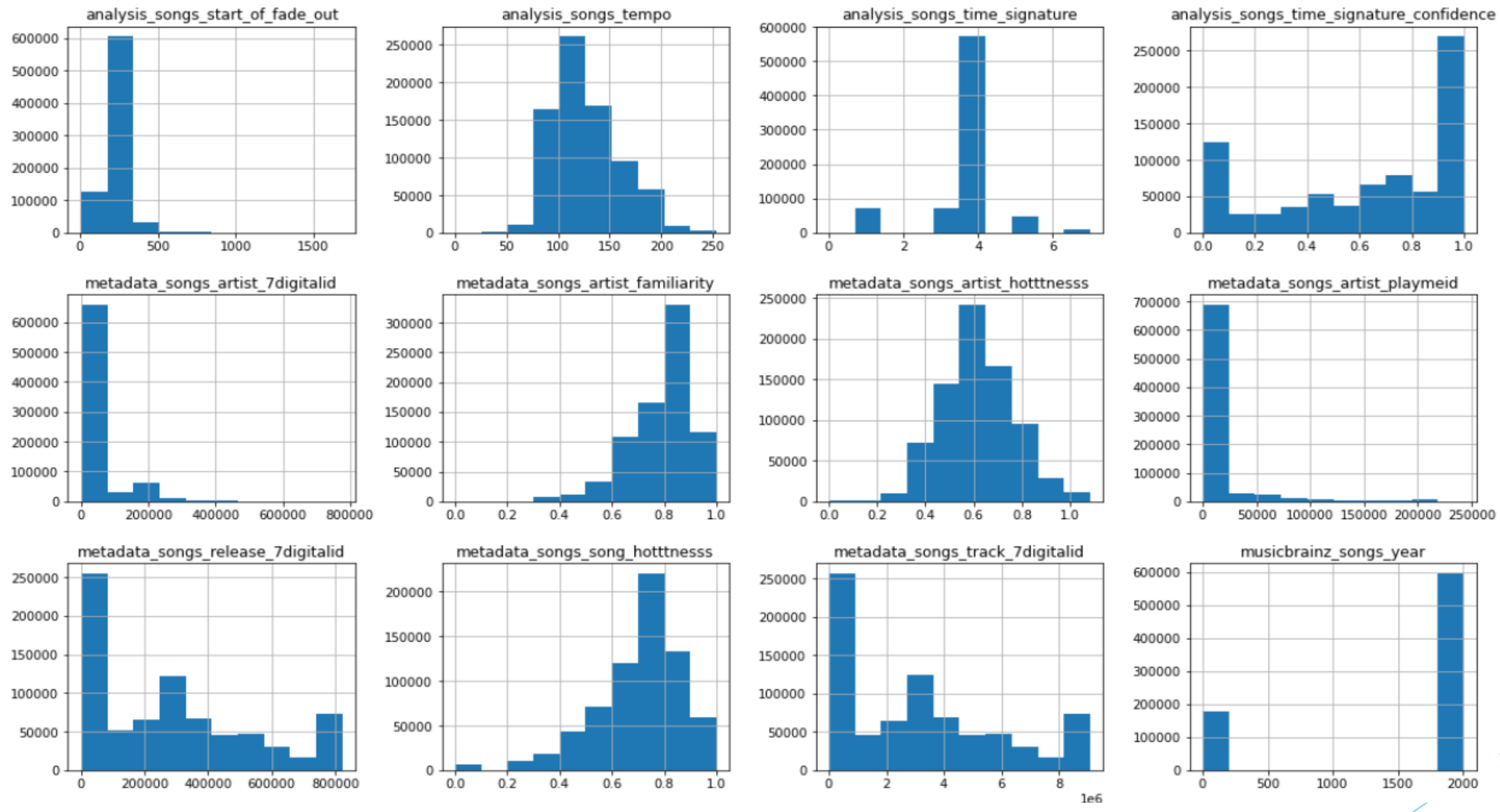
analysis_segments_pitches	analysis_segments_start	analysis_segments_timbre	analysis_songs_analysis_sample_rate	ar
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...	[0.0, 2.30331, 2.67125, 2.84449, 3.07365, 3.25...	[[0.0, 171.13, 9.469, -28.48, 57.491, -50.067,...	22050	ad27a50d9b
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...	[0.0, 2.30331, 2.67125, 2.84449, 3.07365, 3.25...	[[0.0, 171.13, 9.469, -28.48, 57.491, -50.067,...	22050	ad27a50d9b
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...	[0.0, 2.30331, 2.67125, 2.84449, 3.07365, 3.25...	[[0.0, 171.13, 9.469, -28.48, 57.491, -50.067,...	22050	ad27a50d9b
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...	[0.0, 2.30331, 2.67125, 2.84449, 3.07365, 3.25...	[[0.0, 171.13, 9.469, -28.48, 57.491, -50.067,...	22050	ad27a50d9b
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,...	[0.0, 2.30331, 2.67125, 2.84449, 3.07365, 3.25...	[[0.0, 171.13, 9.469, -28.48, 57.491, -50.067,...	22050	ad27a50d9b

Million Songs Subset Dataset - Preprocessing

- **Eliminated 49 features**
  - Missing Data (2 features)
  - No variance (21 features)
  - Unusable format(24 features)
    - Audio data stored in arrays
    - Inconsistent format for Artist Location (i.e. NY vs New York)
  - Irrelevant Data (2 features)
- **Transformed arrays containing words into a string**
  - From: [b’classic pop and rock’, b’folk]
  - To: b’classic pop and rock’| b’folk
- **Standardized Data using z-score normalization**

Million Songs Subset Dataset - Preprocessing

- Why are there so many 0's?



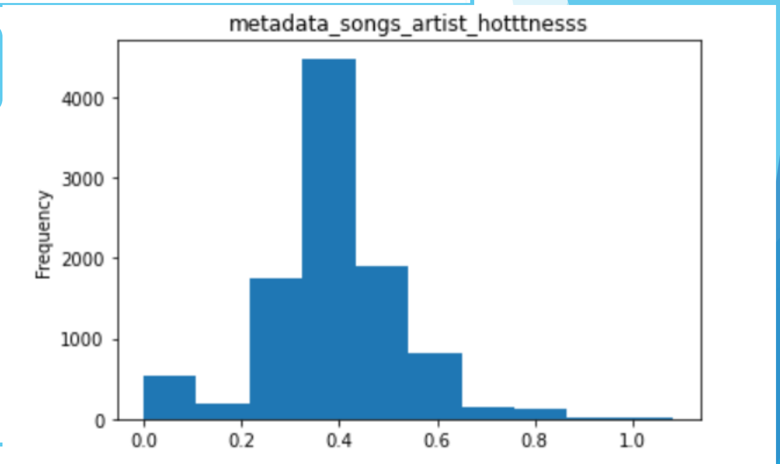
Million Songs Subset Dataset- Preprocessing

8 Features found with values of 0

- 1 feature used 0 to represent categorical values
- 3 features had a range between 0-1
- 3 features used 0 to represent missing data
- 1 feature had an outlier

Outliers

- 3 features used algorithmic estimation
- 2 were found to have ranges of 0-1
- 1 had a maximum value of 1.08 indicating an outlier
- Min-Max scaling was performed to bring values in range

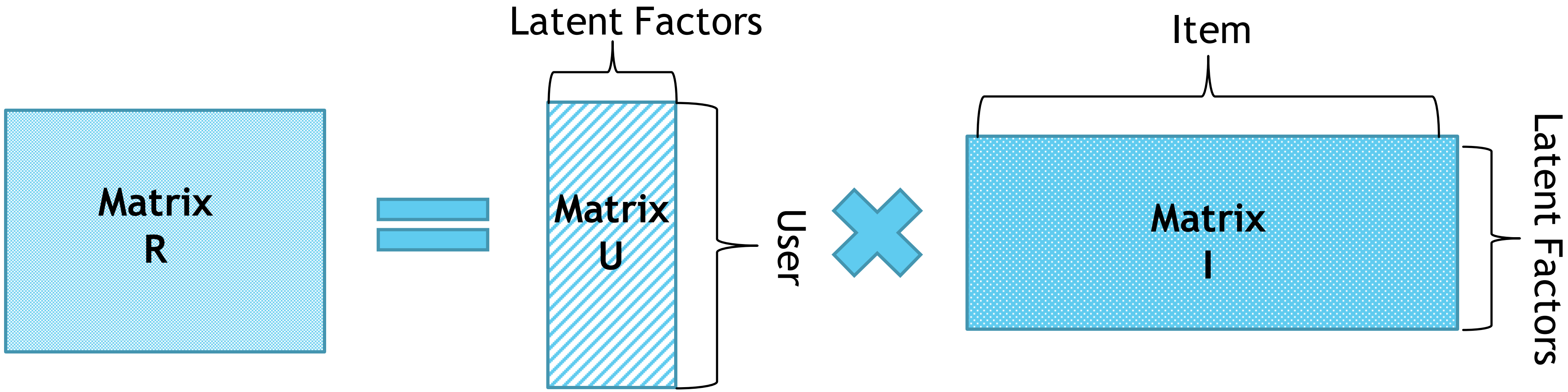


Missing Data

- Equal width binning was used
- Mean imputation

Alternating Least Squares Matrix Factorization

- Method introduced in 2008 paper "Collaborative Filtering for Implicit Feedback Datasets"
- Create a User Item Interaction Matrix



- Latent Factors = hidden features obtained from mathematical models

1) Y. Hu, Y. Koren and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 263-272, doi: 10.1109/ICDM.2008.22.

Alternating Least Squares Matrix Factorization

- Each user-item interaction in the matrix has an associated confidence measure which defines a user's preference towards the item
- Items with more interactions for a given user will have a stronger confidence
- Confidence measures are calculated based on the following equation:

$$C_{ui} = 1 + \alpha r_{ui}$$

C_{ui} = confidence for a given user and item

α = constant

r_{ui} = the number of interactions for a given user item interaction

Alternating Least Squares vs. Traditional Matrix Factorization

- ALS relies on information contained in unobserved user item interactions as these also indicate a user's preference

- ALS uses the alternating least squares approach to minimize loss function. Traditional matrix factorization uses stochastic gradient descent
  - Alternating least squares alternates between recomputing the user factors and item factors with stochastic gradient descent until convergence.

- ALS uses L2 regularization to prevent overfitting
  - L2 regularization adds a penalty term to reduce variance

Alternating Least Squares – Our Approach

- Used Implicit Package in Python
- Train Set = Subset of data that included at least 5 interactions per user
- Model is fit on a Item-user interaction matrix while predictions are made on User-Item interaction matrix
- Model's hyperparameter's tuned using K-Stratified fold with User as the target feature and k = 5

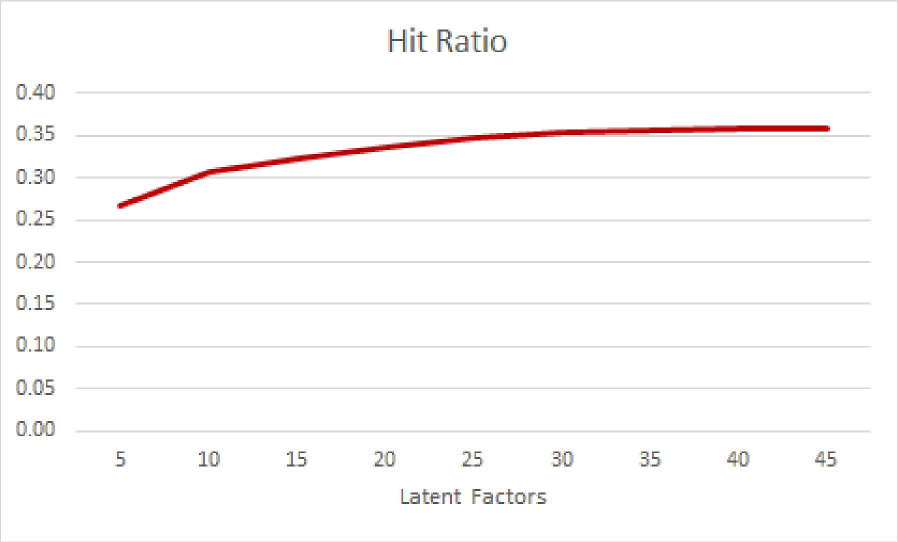
Alternating Least Squares – Our Approach

- Evaluated model parameters on hit ratio:

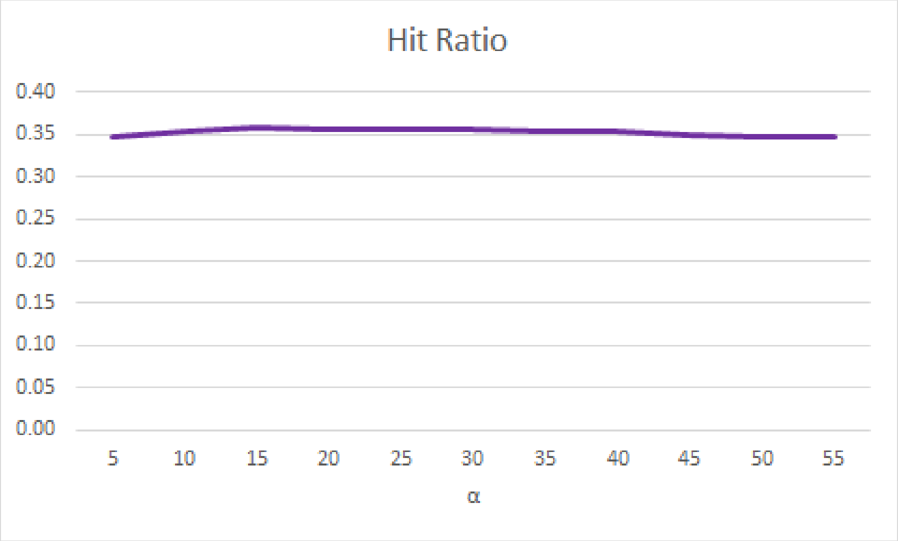
$$Hit\ Ratio = \frac{Number\ of\ songs\ correctly\ recommended}{Total\ number\ of\ observations}$$

Alternating Least Squares – Our Approach

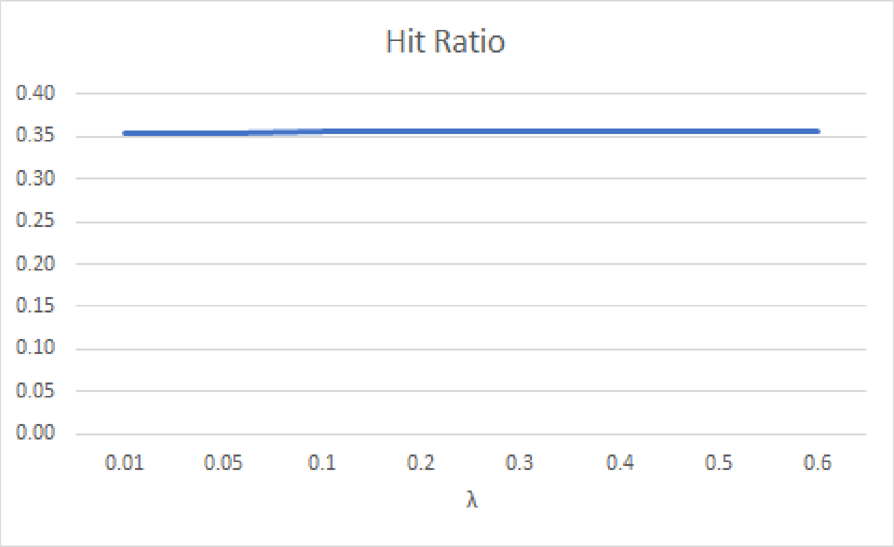
Latent Factors = 35



Alpha = 15



Lambda = 0.01



Iterations = 30

- Bayesian Personalized Ranking (BPR)

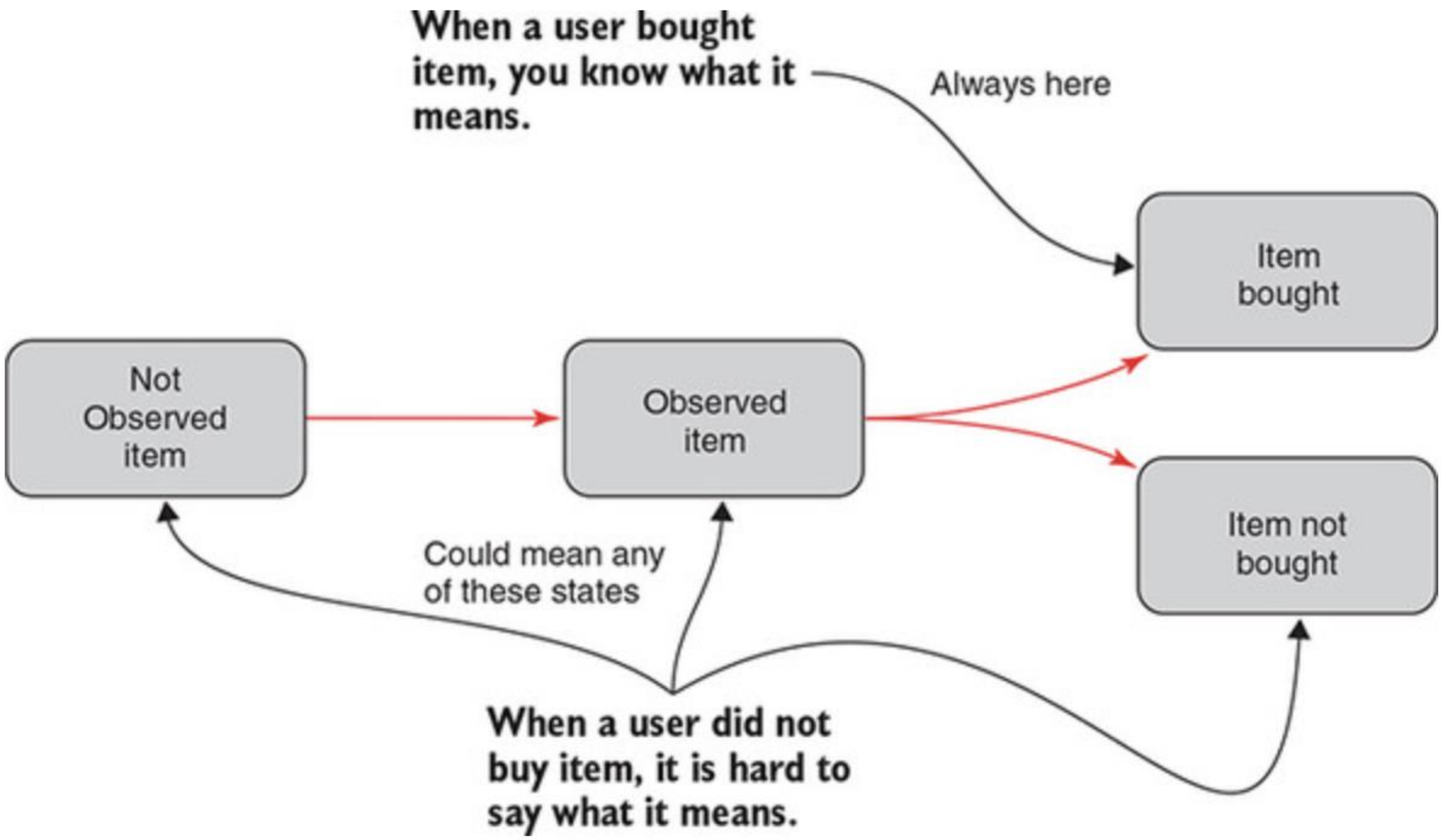


Figure 13.6 [Practical Recommender Systems by Kim Falk]

Bayesian Personalized Ranking (BPR)

- *Why not construct an algorithm that optimizes for ranking itself?*
- This is the goal of **Learning to Rank (LTR) algorithms**
- BPR is a **pairwise LTR algorithm**
  - Take two items and return the ordering of the two
- $>_u$: the ordering of items for a user $\boldsymbol{u}$

$$\forall i, j \in I : i \neq j \implies i >_u j \wedge j >_u i$$

$$p(\Theta | >_u) \propto p(>_u | \Theta) \, p(\Theta)$$

- *If a perfect ranking exists, then there must be some model that produces a perfect ranking.*

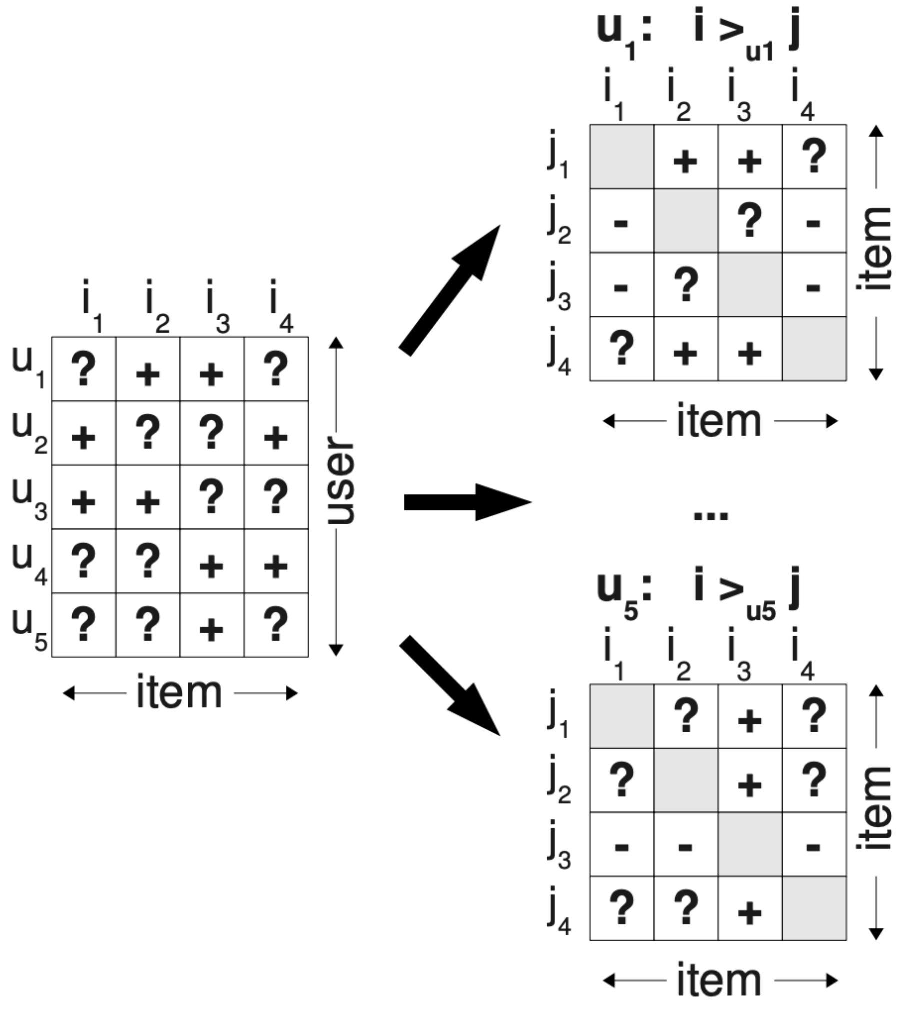
- BPR seeks to find the model Θ that has the highest probability of producing a perfect ranking for all users.

- **$>_u$**
  - Let's adjust our definition of this: represents the ranking of items (songs) ***for all users***
  - One step further: assume there is a ***perfect ranking***

- **Θ**
  - The list of parameters used in a model
  - Think of it as the recommendation system itself

Training data "triplets"

- *(u, i, j)* is used in the training data
- Semantically, user ***u*** is assumed to prefer item ***i*** over item ***j***



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
positive_item_input (InputLayer [(None, 1)]          0
__________________________________________________________________________________________________
negative_item_input (InputLayer [(None, 1)]          0
__________________________________________________________________________________________________
user_input (InputLayer)         [(None, 1)]          0
__________________________________________________________________________________________________
item_embedding (Embedding)      (None, 1, 350)       1286250     positive_item_input[0][0]
                                                                 negative_item_input[0][0]
__________________________________________________________________________________________________
user_embedding (Embedding)      (None, 1, 350)       146388200   user_input[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 350)          0           item_embedding[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 350)          0           item_embedding[1][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 350)          0           user_embedding[0][0]
__________________________________________________________________________________________________
lambda (Lambda)                 (None, 1)            0           flatten[0][0]
                                                                 flatten_1[0][0]
                                                                 flatten_2[0][0]
==================================================================================================
Total params: 147,674,450
Trainable params: 147,674,450
Non-trainable params: 0
__________________________________________________________________________________________________
```

Hyperparameter optimization

- **Latent dimensions**
  - How many latent features are there in the matrix factorization?
- **Learning rates**
  - In our stochastic gradient descent, how big are the steps we're taking to find the minimum of the loss function?

latent_dim	precision	recall	hit_ratio
200	0.839399	0.140424	0.760135
250	0.847948	0.141944	0.768363
300	0.851909	0.142688	0.772392
350	0.858071	0.143770	0.778249

learning_rate	precision	recall	hit_ratio
0.001	0.923205	0.162472	0.879481
0.010	0.967822	0.169617	0.918158
0.100	0.656968	0.094532	0.511715

Model Evaluation Metrics

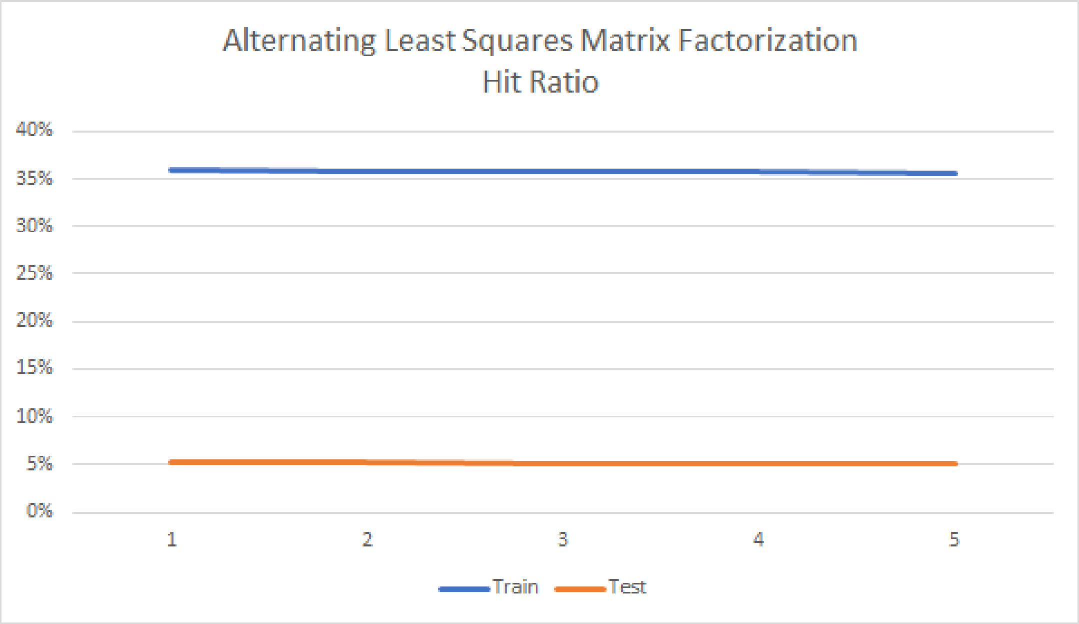
$$Hit\ Ratio = \frac{Number\ of\ songs\ correctly\ recommended}{Total\ number\ of\ observations}$$

$$Recall = average\left(\frac{Number\ of\ songs\ correctly\ recommended\ per\ user}{Number\ of\ songs\ listened\ to\ per\ user}\right)$$

$$Precision = average\left(\frac{Number\ of\ songs\ correctly\ recommended\ per\ user}{Number\ of\ songs\ recommended\ per\ user}\right)$$

Model Evaluation - Performance

- ALS model results were found to severely overfit
- A paired t-test was conducted using test results and recommending the top 10 most popular songs. A statistically significant difference was not found.
- ALS predictions were correct roughly 5% of the time, while the top 10 most popular songs were correct ~2.5% of the time



Model Performance

- Initial comparison of ALS and BPR validation results suggest the following model performance

Model	Hit Ratio	Mean Avg Recall	Mean Avg Precision
ALS	35%	36%	5%
BPR*	92%	17%	97%

*Performance metrics for BPR were based on training/validation dataset

Summary

- Recommendation systems are very complex
  - Many different methods and techniques
  - Not too much publicly available information for state-of-the-art systems
- Requires ML and DL approaches

Questions?