Movie Recommendation Model Comparison

Dominick Russo
University of South Florida

ISM6136

Dr. Balaji Padmanabhan

April 10, 2021

Purpose

- Evaluate recommendation system algorithms for predicting user ratings
- Compare two algorithms in the Python Surprise library with an algorithm in Microsoft AzureML Studio to see which is most accurate.
  - Python Surprise: Specialized scikit library used for coding recommendation systems in python
  - AzureML Studio: Microsoft's graphical user interface (GUI) machine learning application
- Python Surprise Algorithms:
  - K-Nearest Neighbors: Memory-based collaborative filtering
  - Singular Value Decomposition: Model-based collaborative filtering
- AzureML Studio:
  - Matchbox algorithm: Hybrid collaborative filtering and content-based approach

Collaborative Filtering

- Algorithms used in recommendation systems that can be user or item-based.
- Usually focuses on the user ratings of items to imply relevance to other users or items
- K-Nearest Neighbor (KNN)
  - Highly successful algorithm in wide use
  - Issues with data sparsity
- Singular Value Decomposition (SVD)
  - Matrix factorization used to compensate for data sparsity
  - Traditional SVD algorithm does not scale well

(Jiang, Li, & Zhou, 2020)

Matchbox Recommender

- Hybrid approach to recommendation system
- Combines collaborative filtering and content-based approach
  - Collaborative filtering: user ratings of items compared to other users that rated some of the same items. Only user and item IDs are used.
  - Content-based approach: Uses features of users (age, gender, etc.) and items (author, manufacturer, etc.) to find similarities.
  - Matchbox uses collaborative filtering with a content-based approach

(Stern, Herbich, & Graepel, 2009)

Dataset Overview

- MovieLens (small)
  - Ratings table used
    - Attributes: UserId, MovieId, Rating, Timestamp
  - No need to clean or transform
  - 100,837 instances
    - Represents 9,000 movies rated by 600 users
  - Retrieved from https://grouplens.org/datasets/movielens/ on March 28, 2021

Ratings

userId	movieId	rating	timestamp
1	1	4	964982703
1	3	4	964981247
2	8798	3.5	1445714960
2	46970	4	1445715013
…	…	…	…

Movies

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
…	…	…

K-Nearest Neighbor (KNN) – Cosine Similarity
User-Based vs Item-Based

Item-Based 5-fold CV

```
#Basic Knn using cosine similarity

sim_options = {'name': 'cosine', 'user_based': False}

algo = KNNBasic(sim_options=sim_options)

# Run 5-fold cross-validation and print results
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9778	0.9749	$\boxed{0.9672}$	0.9799	0.9778	0.9755	0.0045
MAE (testset)	0.7618	0.7599	$\boxed{0.7531}$	0.7629	0.7636	0.7603	0.0038
Fit time	18.44	17.78	19.77	19.35	19.48	18.97	0.74
Test time	7.99	7.49	8.05	9.09	7.11	7.94	0.67

User-Based 5-fold CV

```
#Basic Knn using cosine similarity

sim_options = {'name': 'cosine', 'user_based': True}

algo = KNNBasic(sim_options=sim_options)

# Run 5-fold cross-validation and print results
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9733	0.9681	0.9742	0.9812	$\boxed{0.9661}$	0.9726	0.0053
MAE (testset)	0.7519	0.7462	0.7485	0.7545	$\boxed{0.7438}$	0.7490	0.0038
Fit time	0.50	0.52	0.64	0.67	0.63	0.59	0.07
Test time	1.46	1.53	1.71	1.51	1.99	1.64	0.19

K-Nearest Neighbor (KNN)
Cosine Similarity vs GridSearchCV using MSD Similarity

User-Based 5-fold CV

```
#Basic Knn using cosine similarity

sim_options = {'name': 'cosine', 'user_based': True}

algo = KNNBasic(sim_options=sim_options)

# Run 5-fold cross-validation and print results
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9733	0.9681	0.9742	0.9812	$\boxed{0.9661}$	0.9726	0.0053
MAE (testset)	0.7519	0.7462	0.7485	0.7545	$\boxed{0.7438}$	0.7490	0.0038
Fit time	0.50	0.52	0.64	0.67	0.63	0.59	0.07
Test time	1.46	1.53	1.71	1.51	1.99	1.64	0.19

User-Based GridSearchCV

```
# Tune with GridSearchCV to see if results are improved
param_grid = {'n_epochs': [5,10], 'lr_all': [0.002, 0.005], 'reg_all': [0.4, 0.6]}

gs = GridSearchCV(KNNBasic, param_grid, measures=['rmse', 'mae'], cv=5)
gs.fit(data)

# Show best score and parameters that gave best rmse
print("Best RMSE: ", gs.best_score['rmse'])
print("RMSE Params: ", gs.best_params['rmse'])
print("Best MAE = ", gs.best_score['mae'])
print("MAE Params: ", gs.best_params['mae'])
```

```
Best RMSE:  0.9463981470334168
RMSE Params:  {'n_epochs': 5, 'lr_all': 0.002, 'reg_all': 0.4}
Best MAE =  0.7253927910936966
MAE Params:  {'n_epochs': 5, 'lr_all': 0.002, 'reg_all': 0.4}
```

K-Nearest Neighbor (KNN) – Pearson Similarity User-Based vs Item-Based

Item-Based 5-fold CV

```
#Basic Knn using pearson similarity

sim_options = {'name': 'pearson', 'user_based': False}

algo = KNNBasic(sim_options=sim_options)

# Run 5-fold cross-validation and print results
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9732	0.9693	0.9731	0.9636	0.9706	0.9700	0.0035
MAE (testset)	0.7565	0.7519	0.7551	0.7470	0.7564	0.7534	0.0036
Fit time	28.29	29.50	26.14	23.17	25.45	26.51	2.21
Test time	9.77	8.83	8.52	8.26	8.32	8.74	0.55

User-Based 5-fold CV

```
#Basic Knn using pearson similarity

sim_options = {'name': 'pearson', 'user_based': True}

algo = KNNBasic(sim_options=sim_options)

# Run 5-fold cross-validation and print results
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9763	0.9730	0.9706	0.9647	0.9770	0.9723	0.0045
MAE (testset)	0.7553	0.7520	0.7471	0.7475	0.7523	0.7508	0.0031
Fit time	0.84	0.72	0.75	0.74	0.75	0.76	0.04
Test time	1.56	1.53	1.58	1.52	1.67	1.57	0.05

KNN Best Model

User-Based GridSearchCV

- User-Based
- Similarity: Mean Squared Difference (MSD)
- Cross Validation: 5-fold GridSearchCV

```
# Tune with GridSearchCV to see if results are improved
param_grid = {'n_epochs': [5,10], 'lr_all': [0.002, 0.005], 'reg_all': [0.4, 0.6]}

gs = GridSearchCV(KNNBasic, param_grid, measures=['rmse', 'mae'], cv=5)
gs.fit(data)

# Show best score and parameters that gave best rmse
print("Best RMSE: ", gs.best_score['rmse'])
print("RMSE Params: ", gs.best_params['rmse'])
print("Best MAE = ", gs.best_score['mae'])
print("MAE Params: ", gs.best_params['mae'])
```

```
Best RMSE:  0.9463981470334168
RMSE Params:  {'n_epochs': 5, 'lr_all': 0.002, 'reg_all': 0.4}
Best MAE =  0.7253927910936966
MAE Params:  {'n_epochs': 5, 'lr_all': 0.002, 'reg_all': 0.4}
```

Singular Value Decomposition (SVD)
5-Fold CV has slightly lower error vs GridSearchCV

5-Fold CV

```
# Select SVD algorithm
algo = SVD()

# Run 5-fold cross-validation and print results
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8778	0.8806	0.8649	0.8759	0.8668	0.8732	0.0062
MAE (testset)	0.6745	0.6742	0.6659	0.6741	0.6676	0.6713	0.0037
Fit time	3.89	4.01	4.07	5.18	6.06	4.64	0.85
Test time	0.16	0.16	0.13	0.30	0.25	0.20	0.06

GridSearchCV

```
# Tune with GridSearchCV to see if results are improved
param_grid = {'n_epochs': [5,10], 'lr_all': [0.002, 0.005], 'reg_all': [0.4, 0.6]}

gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=5)
gs.fit(data)

# Show best score and parameters that gave best rmse
print("Best RMSE: ", gs.best_score['rmse'])
print("RMSE Params: ", gs.best_params['rmse'])
print("Best MAE = ", gs.best_score['mae'])
print("MAE Params: ", gs.best_params['mae'])
```

```
Best RMSE:  0.8907163941701057
RMSE Params:  {'n_epochs': 10, 'lr_all': 0.005, 'reg_all': 0.4}
Best MAE =  0.6895596992047841
MAE Params:  {'n_epochs': 10, 'lr_all': 0.005, 'reg_all': 0.4}
```

Python Surprise
Best Overall

User-Based GridSearchCV

- User-Based
- Similarity: Mean Squared Difference (MSD)
- Cross Validation: 5-fold GridSearchCV

```
# Tune with GridSearchCV to see if results are improved
param_grid = {'n_epochs': [5,10], 'lr_all': [0.002, 0.005], 'reg_all': [0.4, 0.6]}

gs = GridSearchCV(KNNBasic, param_grid, measures=['rmse', 'mae'], cv=5)
gs.fit(data)

# Show best score and parameters that gave best rmse
print("Best RMSE: ", gs.best_score['rmse'])
print("RMSE Params: ", gs.best_params['rmse'])
print("Best MAE = ", gs.best_score['mae'])
print("MAE Params: ", gs.best_params['mae'])
```

```
Best RMSE:  0.9463981470334168
RMSE Params:  {'n_epochs': 5, 'lr_all': 0.002, 'reg_all': 0.4}
Best MAE =  0.7253927910936966
MAE Params:  {'n_epochs': 5, 'lr_all': 0.002, 'reg_all': 0.4}
```

AzureML
No features included

Split Data

Splitting mode

Recommender Split

Fraction of training-only users

0.75

Fraction of test user ratings for training

0.25

Train Matchbox Recommender

Number of traits

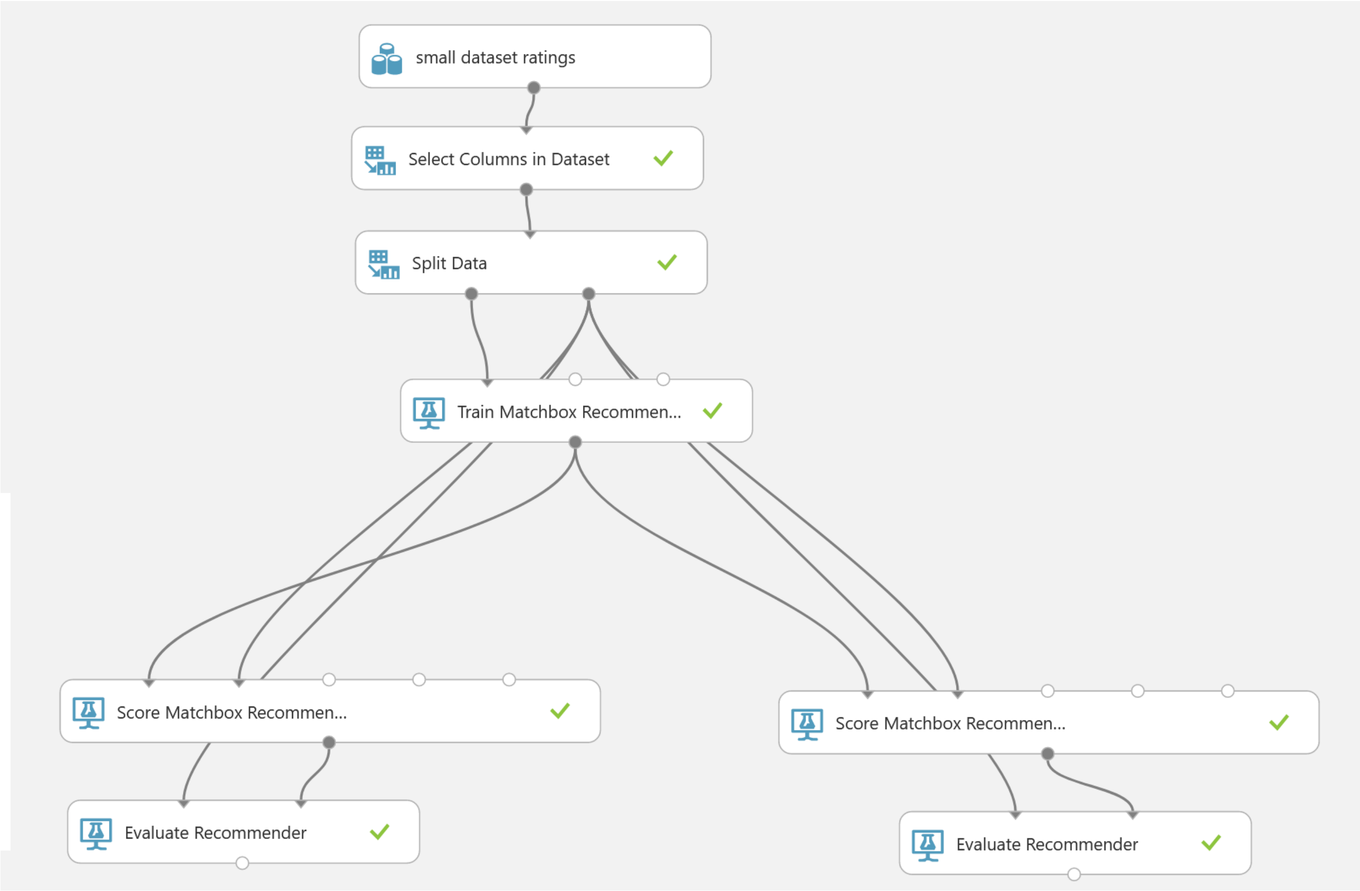
10

Number of recommendation algorithm iterations

5

Number of training batches

4



Rating Prediction Score

MAE	RMSE
0.683413	1.120998

Item Recommendation Score

NDCG
0.87583

AzureML
Movie features included

Split Data

Splitting mode

Recommender Split

Fraction of training-only users

0.75

Fraction of test user ratings for training

0.25

Train Matchbox Recommender

Number of traits

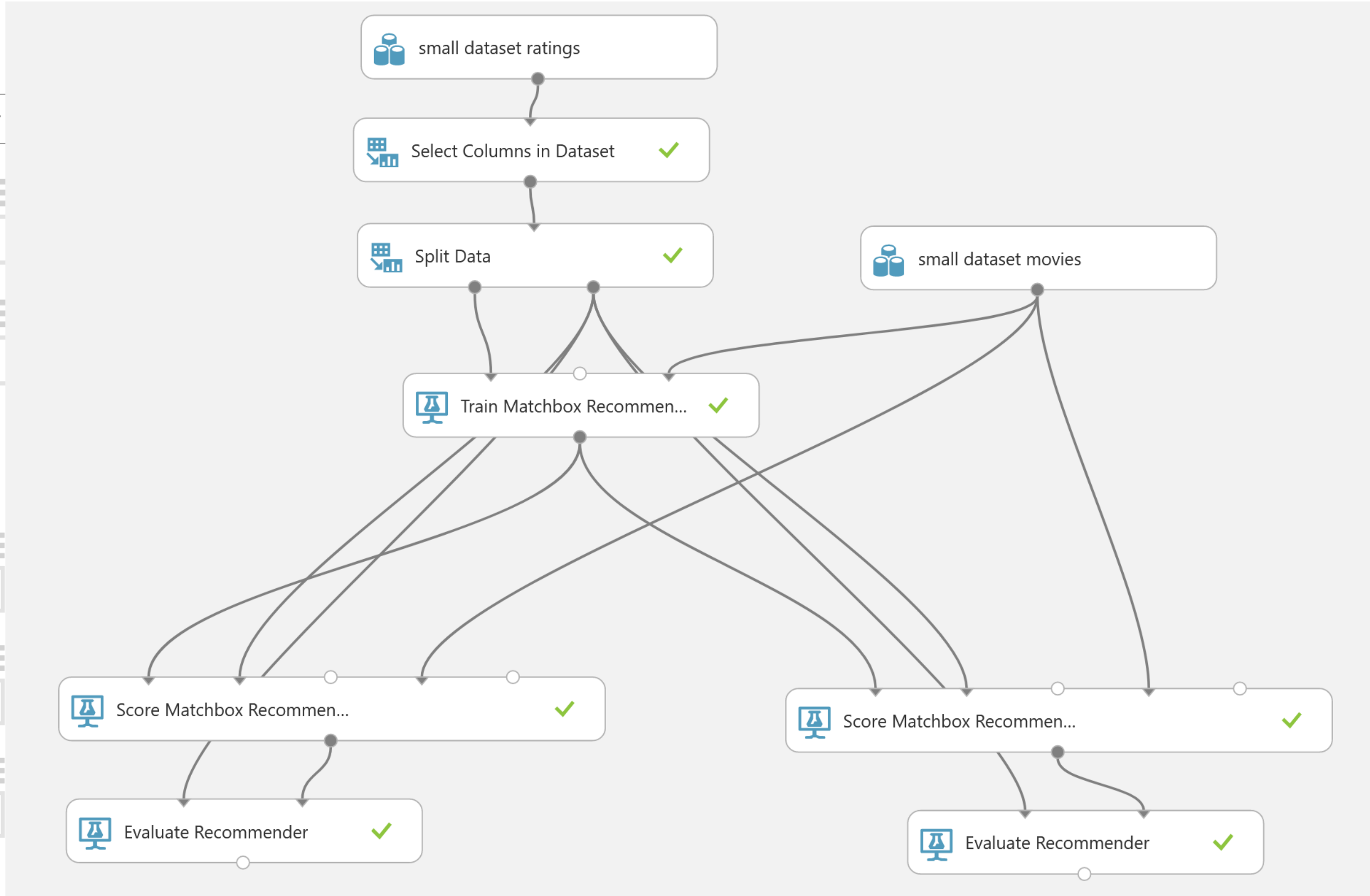
10

Number of recommendation algorithm iterations

5

Number of training batches

4



Rating Prediction Score

MAE	RMSE
0.680722	1.111091

Item Recommendation Score

NDCG
0.87865

AzureML Movie features Included Best Overall

Split Data

Splitting mode

Recommender Split

Fraction of training-only users

0.75

Fraction of test user ratings for training

0.25

Train Matchbox Recommender

Number of traits

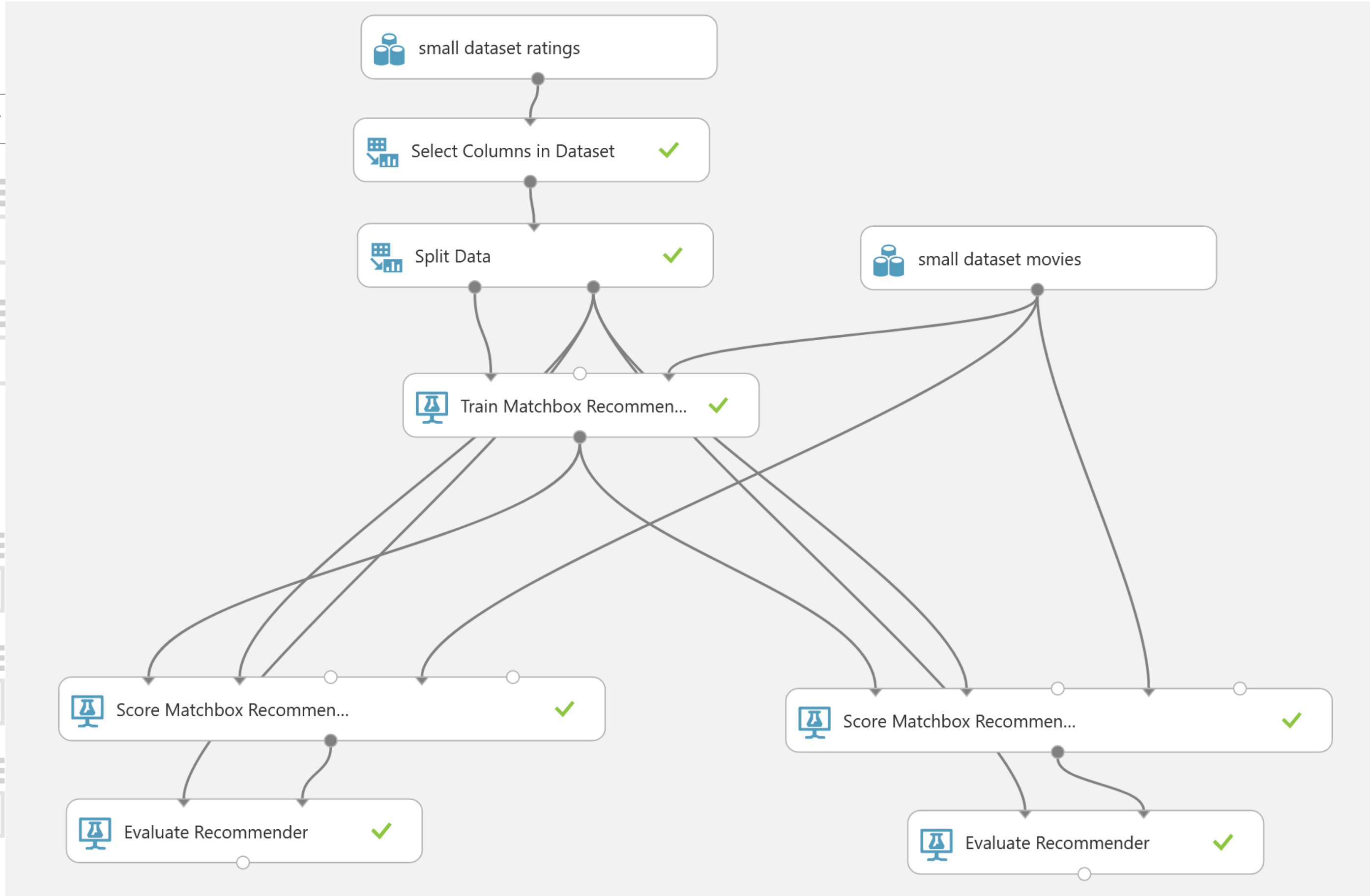
10

Number of recommendation algorithm iterations

5

Number of training batches

4



Rating Prediction Score

MAE	RMSE
0.680722	1.111091

Item Recommendation Score

NDCG
0.87865

References

- Jiang, S., Li, J., & Zhou, W. (2020). An Application of SVD++ Method in Collaborative Filtering*. 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, Chengdu, China. pp. 192-197, doi: 10.1109/ICCWAMTIP51612.2020.9317347.
- Stern, D., Herbich, R., Graepel, T. (2009). *Matchbox: Large Scale Online Bayesian Recommendations. World Wide Web Conference Com-mittee (IW3C2). https://www.microsoft.com/en-us/research/wp-content/uploads/2009/01/www09.pdf*