Music Recommender System

Collaborative Filtering with Matrix Factorization

Joseph Tran

Results from project:

	AUC
Popularity	0.875
ALS Model	0.916
LightFM model	0.937

Types of Recommender Systems

- Most Popular Items
- Content Based
- Collaborative Filtering
- Hybrid Models

Why Use Recommender Systems?

- Enhance user experience
- Increase time on application
- Generate increase in business revenue
- Customer Retention

How do recommenders add value?

- Recommend items user is not aware of
- Increase exposure to larger array of products
- Prolong time spent on a platform
- User acquisition

Implicit v. Explicit Data

Explicit data

- User supplied product ratings data
- Difficult to obtain
- Less prevalent than implicit data
- Easy to interpret

Implicit Data

- Artifacts of user interaction within the platform
- Easy to obtain
- More prevalent than explicit data
- More difficult to interpret

NowPlaying-RS Dataset

- Implicit data
- 17 million listening events
- 3 separate files: LE’s, sentiment scores, content features
- Scraped from Twitter over the course of a year
- Song content features from Spotify
- 20K users
- 80K songs

Data Wrangling Steps

Sentiment score file

- Fix misaligned header rows
- Impute missing values with column-wise average score
- Select dictionary with least amount of missing values
- Drop rows with missing values
- Merge with LE file

Data Wrangling Steps

Content features file

- Drop unused columns: Coordinates, Place, Geo, Time zone, Entities
- Filter language columns to include English only
- Collect user/song hashtags for unique song/unique user
- Map list of hashtags to column associated with user/song
- Ensure unique set of hashtags per user/song
- Filter user/track LE's to > 10
- Join with sentiment/LE file

Data Wrangling Steps

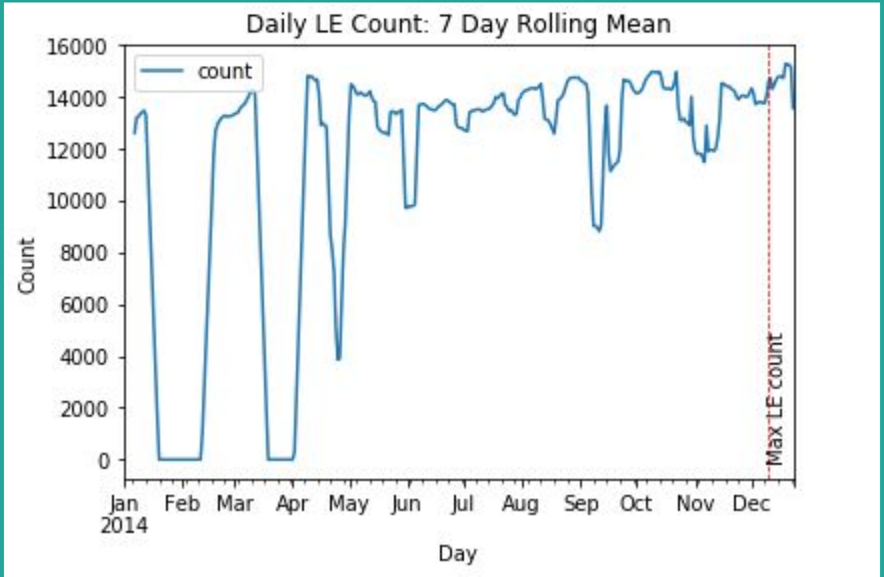
Create user/item matrix for MF model

- Create column of 1's for each unique listening event
- Group by user/song and aggregate by count
- Create lists for unique user, song, count
- Store as sparse csr matrix for efficient storage (required for memory issues and model)

Results are a 99.3% sparse matrix with dimension 21220 x 81343

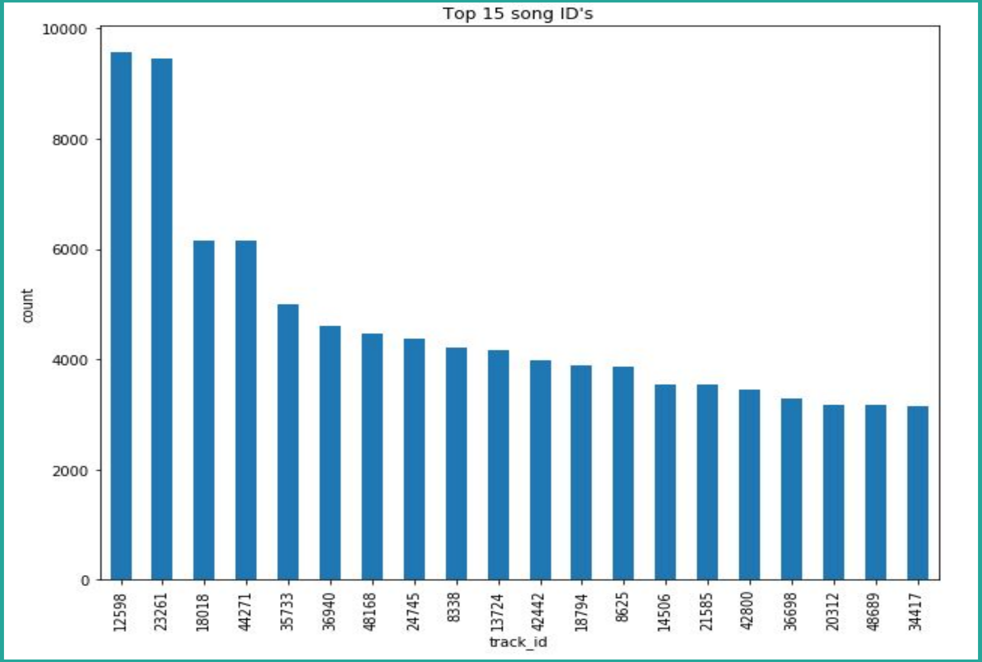
Data Story

- Recorded over course of 2014
- Appears to show crawler breakdown
- Max LE count occurs 4-17-2014



Data Story

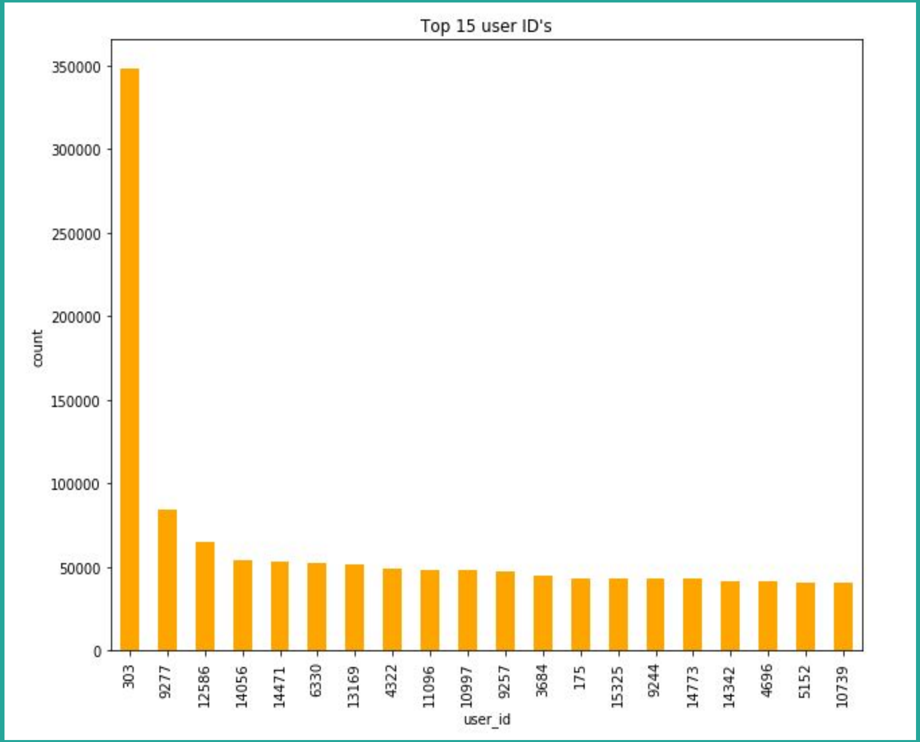
Top Song ID's



Data Story

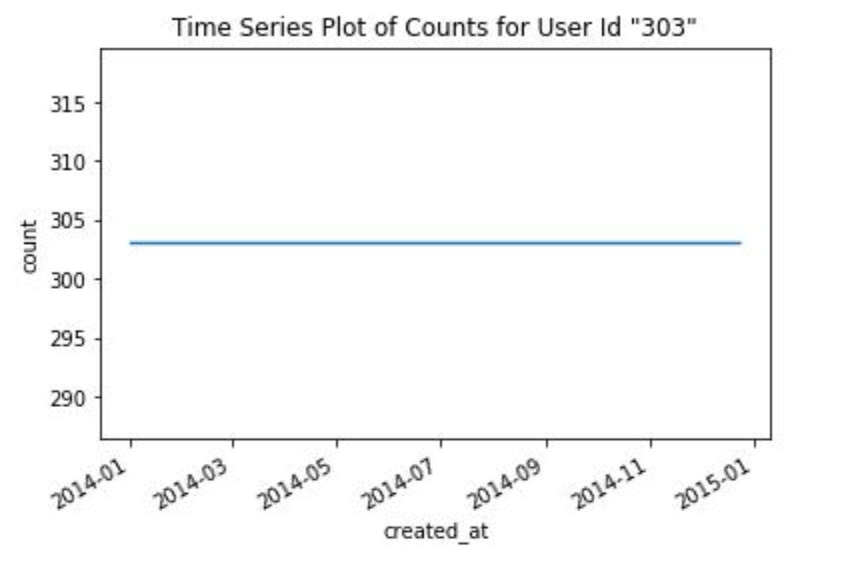
Most active users

Shows odd behavior with User 303



Data Story

User 303 time series plot: suspected bot



Python Libraries for Recommender Systems

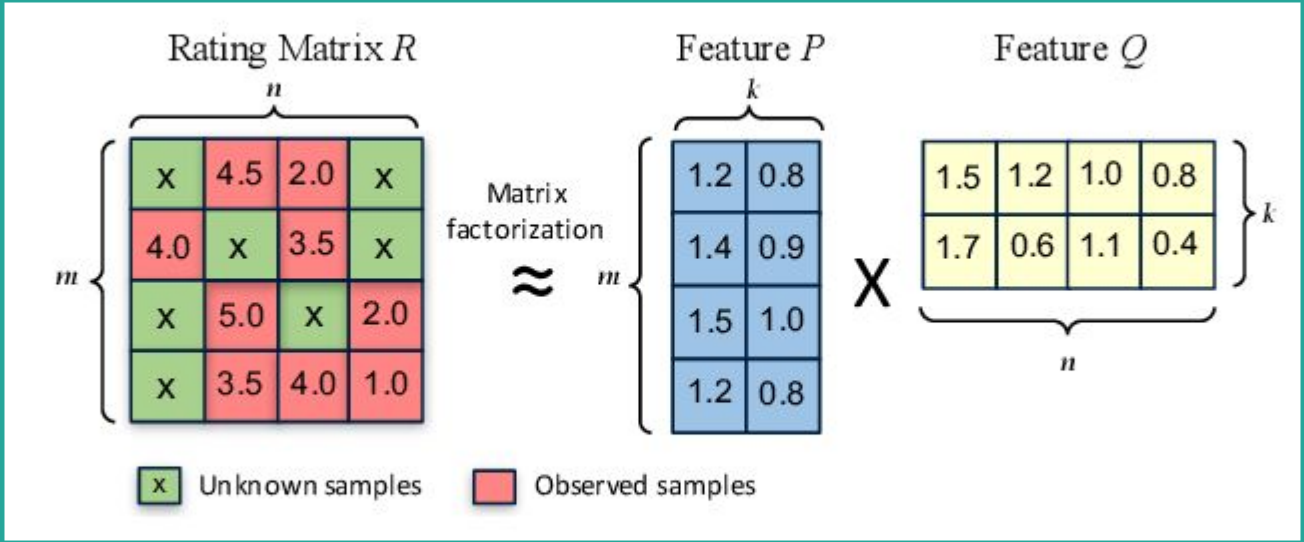
Implicit

- Supports implicit data
- Fast implementation of Alternating Least Squares algorithm
- Easy to use

LightFM

- Support implicit data
- Fast implementation of matrix factorization algorithm
- ‘Easy’ to include content features for hybrid system

Matrix Factorization



https://www.semanticscholar.org/paper/CuMF_SGD%3A-Fast-and-Scalable-Matrix-Factorization-Xie-Tan/bcc0fec44cff759a5f48d4fcab943b379ff9bb3f/figure/0

Implicit: Alternating Least Squares

Solving the MF problem

- Alternating least squares
- Singular value decomposition
- Non-negative matrix factorization

Implicit: Alternating Least Squares

Relevant equations

p_{ui} ε (0,1) - preference: probability user liked a song

r_{ui} ε R - recording: # of user/song interactions

p_{ui}= {1 if r_{ui} >0, 0 if r_{ui} =0} - relationship between preference and recording

$C_{ui} = 1 + \alpha r_{ui}$ - notion of confidence

$C_{ui}(p_{ui} - U_u X_i^T)$ - Equation to minimize

Creating Implicit ALS model

```
alpha = 40

user_vecs, item_vecs = implicit.alternating_least_squares((train*alpha).astype('double'),
                                                          factors=20,
                                                          regularization = 0.1,
                                                          iterations = 50)
```

Evaluating ALS model

Create train and test sets

- Need to use entire data for training
- Mask percentage of interactions in train data by setting equal to 0
- Compare the masked interaction values to the same indices subsequent to performing the dot product of the user/item latent features
- Calculate AUC score at indices in which interactions were masked

LightFM model

Instantiate model:

```
#instantiate light fm model with 20 components (same as als model)
modelfm = LightFM(
    no_components=20,
    learning_rate=0.05,
    loss='warp',
    random_state=2019)
```

Fit model:

```
#fit the model on the training data
modelfm.fit(
    train,
    item_features=None,
    user_features=None, sample_weight=None,
    epochs=5, num_threads=4, verbose=True)
```

Evaluate AUC:

```
#score the model on the test data
score = auc_score(
        modelfm, test,
        item_features=None,
        user_features=None,
        num_threads=4).mean()
```

Future Work

Extend basic LightFM model by supplying song content features to create hybrid model.