Four Common Problems with Recommenders

And How to Address Them

The Problems

1. *Wrong Metric*

2. *Cold Start Problem*

3. *Difficulty Utilizing All Useful Data*

4. *Speed and Stability*

Mean Squared Error Not Right Metric If We Recommend Small Subset of Items

	True	Model A Pred	Model B Pred
Item 1	5	1	4
Item 2	3	3	2
Item 3	3	3	2
Item 4	3	3	2
Item 5	3	3	2
Item 6	1	5	2

Metrics Solutions

- *Alternative metrics*
  - *Average of Top N Recommendations*
  - *Objective function asymmetries*

- *SGD / Optimization*
  - *Differentiable and convex objective function*

- *Approximate true objective with something optimizable*

$$\sum r_i e_i^2$$

Cold Start Problem

Recommendations for new people or items are typically bad

Cold Starts Problems in Various Recommender Systems
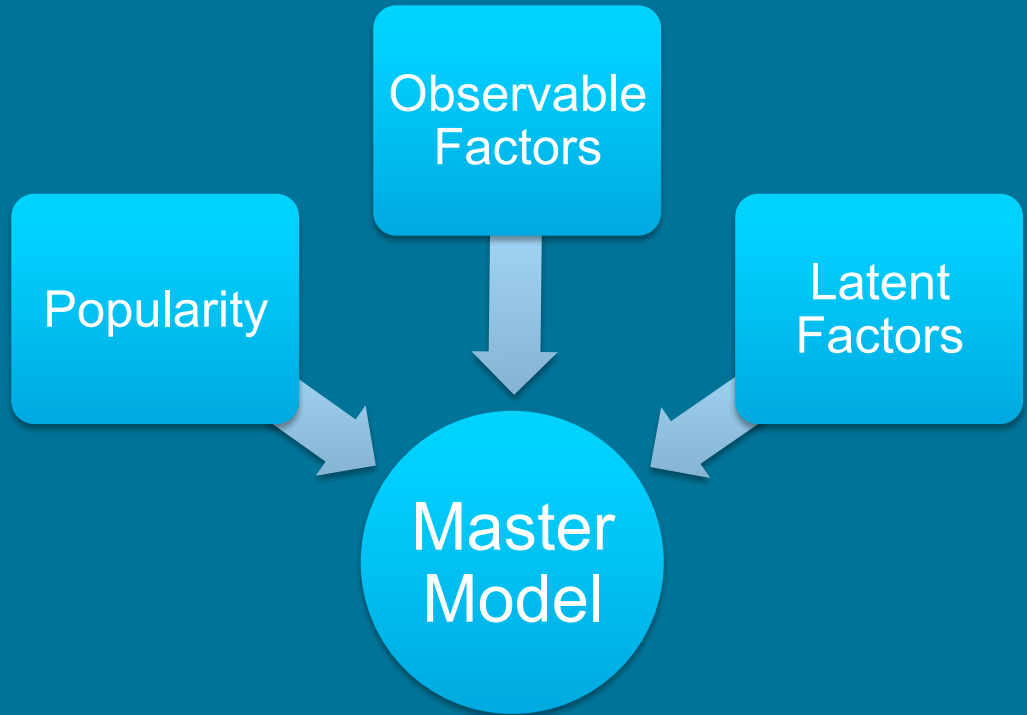
Most Affected
- Item-Item and User-User

Moderately Affected
- Latent Factors (SVD-based)

Least Affected
- Observable factors
- Popularity

Address Cold-Starts Without Sacrificing the Rest of Your Model

Observable Factors

Popularity

Latent Factors

Master Model

- Master model can be weighted average or switching model
- Commonly solved with UI/UX rather than data science

What Predictive Data Do We Have That SVD Isn't Using

What Predictive Data Do We Have That SVD Isn't Using

- Average Ratings for Each User or Item (biases)

- Known Item Characteristics

- Known User Characteristics

- Implicit Feedback

Adding Item and/or User Biases

- Subtract bias before factorization, add back when predicting

- Preserve sparsity

- Item bias more important than user bias

Integrating Known Item Characteristics

SVD Feature

Item Matrix

	Latent 1	Latent 2	Latent 3	Observed 1	Observed 2
Item A	?	?	?	1	3
Item B	?	?	?	0	0
Item C	?	?	?	1	5

User Taste Matrix

	Latent 1	Latent 2	Latent 3	Observed 1	Observed 2
Al	?	?	?	?	?
Betty	?	?	?	?	?
Carl	?	?	?	?	?

Known User Characteristics

An Idea That Hasn't Caught On

Item Matrix

	Latent 1	Latent 2	Latent 3	Observed 1	Observed 2
Item A	?	?	?	?	?
Item B	?	?	?	?	?
Item C	?	?	?	?	?

User Taste Matrix

	Latent 1	Latent 2	Latent 3	Observed 1	Observed 2
Al	?	?	?	1	5
Betty	?	?	?	0	1
Carl	?	?	?	1	0

Implicit Feedback

SVD++

- The items a user has chosen to purchase/rate tells us about their tastes.

SVD

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

SVD++

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

Execution Speed Tips

Serialize The Model
- Optimize from near the optimum
- Good practice in general

Vectorize Operations
- Language dependent
- Important in Python

Switching Models
- Skip unnecessary work

Work in Batches
- Benefits from understanding usage patterns

Another Take on Speed / Stability

Spark and GraphLab

Simple API
- DataFrame objects
- Familiar modeling API

Scalable
- Built for distributed computing
- Fast

Tested
- Reliability
- Documentation

```python
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating


# Load and parse the data
data = sc.textFile("data/mllib/als/test.data")
ratings = data.map(lambda l: l.split(',')).map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))


# Build the recommendation model using Alternating Least Squares
rank = 10
numIterations = 20
model = ALS.train(ratings, rank, numIterations)


# Evaluate the model on training data
testdata = ratings.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).reduce(lambda x, y: x + y) / ratesAndPreds.count()
print("Mean Squared Error = " + str(MSE))


# Save and load model
model.save(sc, "myModelPath")
sameModel = MatrixFactorizationModel.load(sc, "myModelPath")
```

Should You Use Graphlab or Spark

Graphlab
- Matrix factorization, Item-Item and Popularity
- Appropriate for range of scales

MlLib
- Requires Spark
- ALS solver at scale

Your Class
- Flexibility
- Potential for greater predictive accuracy

Framework Conclusions

- Fast, reliable, well documented

- Harder to extend

- Only part of what you need