Matrix Factorization for Recommendation

Warning: There are a lot of acronyms in this lecture!

Natalie Hunt

galvanize

- UV Decomposition (UVD)
- SVD vs UVD
- UVD vs NMF
- UVD via Stochastic Gradient Descent (SGD)
- Matrix Factorization for Recommendation:
    - Basic system:
        - UVD + SGD... FTW
    - Intermediate topics:
        - regularization
        - accounting for biases

UV Decomposition (UVD)

$$R_{m\times n} \approx U_{m\times k} V_{k\times n}$$

$$r_{ij} \approx u_{i:} \cdot v_{:j}$$

- You choose *k*.
- *UV* approximates *R* by necessity if *k* is less than the rank of *R*.
- Usually choose: *k << min(n,m)*
- Compute *U* and *V* such that:

$$\arg\min_{U,V} \sum_{i,j} (r_{ij} - u_{i:} \cdot v_{:j})^2$$

Least Squares!

SVD vs UVD

SVD:

- $R = USV^T$
- U is an orthogonal matrix
- S is a diagonal matrix of decreasing positive “singular” values
- V is an orthogonal matrix
- Has a unique, exact solution

UVD:

- $R \sim= UV$
- U and V will not (likely) be orthogonal
- Has many approximate, non-unique solutions:
    - non-convex optimization; has many local minima
- Has a tunable parameter k

UVD vs NMF

UVD:

- By convention: *R ~= UV*
- … (see previous slides)

NMF is a specialization of **UVD**!

Both are approximate factorizations, and both optimize to reduce the RSS.

NMF:

- By convention: *V ~= WH*
- Same as UVD, but with one extra constraint: **all values of *V*, *W*, and *H* must be non-negative!**

UVD vs NMF *(continued)*

UVD and NMF are both solved using either:

- Alternating Least Squares (ALS)
- Stochastic Gradient Descent (SGD)

You did **ALS** yesterday,
so let's do **SGD** today!

(and we'll see why SGD has some
advantages for recommender systems)

UVD via Stochastic Gradient Descent (SGD)

Boardwork…

ALS vs SGD

ALS:

- Parallelizes very well
- Available in Spark/MLlib
- Only appropriate for matrices that don't have missing values
  (we'll call this a **dense** matrix in this lecture)

SGD:

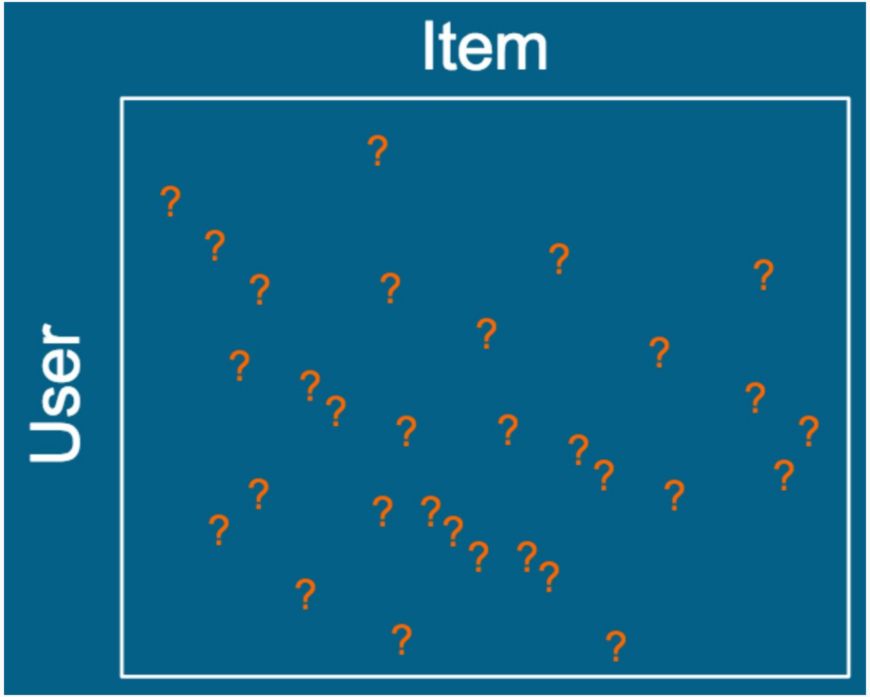
- Faster (if on single machine)
- Requires tuning learning rate
- Anecdotal evidence of better results…
- Works with missing values
  (we'll call this a **sparse** matrix in this lecture)
  (we'll see how missing values are handled soon!)

Matrix Factorization for Recommendation

Recall: An explicit-rating utility matrix is usually VERY sparse…

We've previously used SVD to find latent features (aka, factors)… Would SVD be good for this sparse utility matrix? (Hint: No!)

What's the problem with using SVD on this sparse utility matrix?



Would UVD (or NMF) work better than SVD to find latent factors when the utility matrix is sparse?

(Hint: Consider ways to change the SGD algorithm to handle missing values in the sparse utility matrix.)

SVD vs UVD *(revisited)*

SVD:

- $R = USV^T$
- …
- **Bad if *R* has missing values!**
  - You are forced to fill in missing values.
  - Solution fits these fill-values (which is silly).
  - Makes for a much larger memory footprint.
  - Slow to compute for large matrices.

UVD:

- $R \sim= UV$
- …
- **Handles missing values when computed via SGD.**

$$\arg\min_{U,V} \sum_{i,j \in \mathcal{K}} (r_{ij} - u_{i:} \cdot v_{:j})^2$$

Set of indices of known rating

UVD (or NMF) + SGD... FTW!

UVD + SGD makes a lot of sense for recommender systems.

In fact, **UVD + SGD** is ‘**best in class**’ option for *many* recommender domains:

- No need to impute missing values.
- Use regularization to avoid overfitting.
- Optionally include biases terms to communicate prior knowledge.
- Can handle time-dynamics (e.g. change in user preference over time).
- Used by the winning entry in the Netflix challenge.

Warning: Don’t forget to regularize!

Since now we’re fitting a large parameter set to sparse data, you’ll most certainly need to regularize!

$$\underset{U,V}{\arg\min} \sum_{i,j \in \mathcal{K}} (r_{ij} - u_{i:} \cdot v_{:j})^2 + \boxed{\lambda(||u_{i:}||^2 + ||v_{:j}||^2)}$$

Tune lambda: the amount of regularization

Accounting for Biases (let’s capture our domain knowledge!)

In practice, much of the observed variation in rating values is due to item bias and user bias:

- Some items (e.g. movies) have a tendency to be rated high, some low.
- Some users have a tendency to rate high, some low.

We can capture this prior domain knowledge using a few bias terms:

$$b_{ij} = \mu + b_i^* + b_j'$$

The overall bias of the rating by user *i* for item *j*

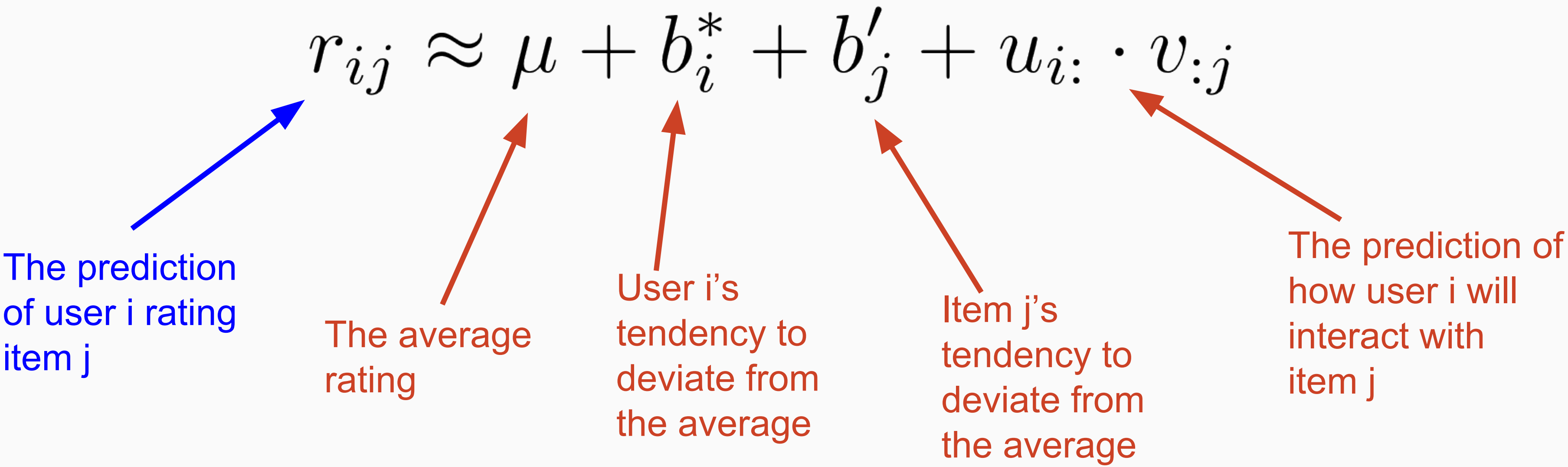
The overall average rating
(i.e. the overall bias)

User i’s average deviation from the overall average

Item j’s average deviation from the overall average

We added bais terms… now:

The 4 parts of a prediction



$$r_{ij} \approx \mu + b^*_i + b'_j + u_{i:} \cdot v_{:j}$$

The prediction of user i rating item j

The average rating

User i’s tendency to deviate from the average

Item j’s tendency to deviate from the average

The prediction of how user i will interact with item j

Accounting for Biases (the new cost function)

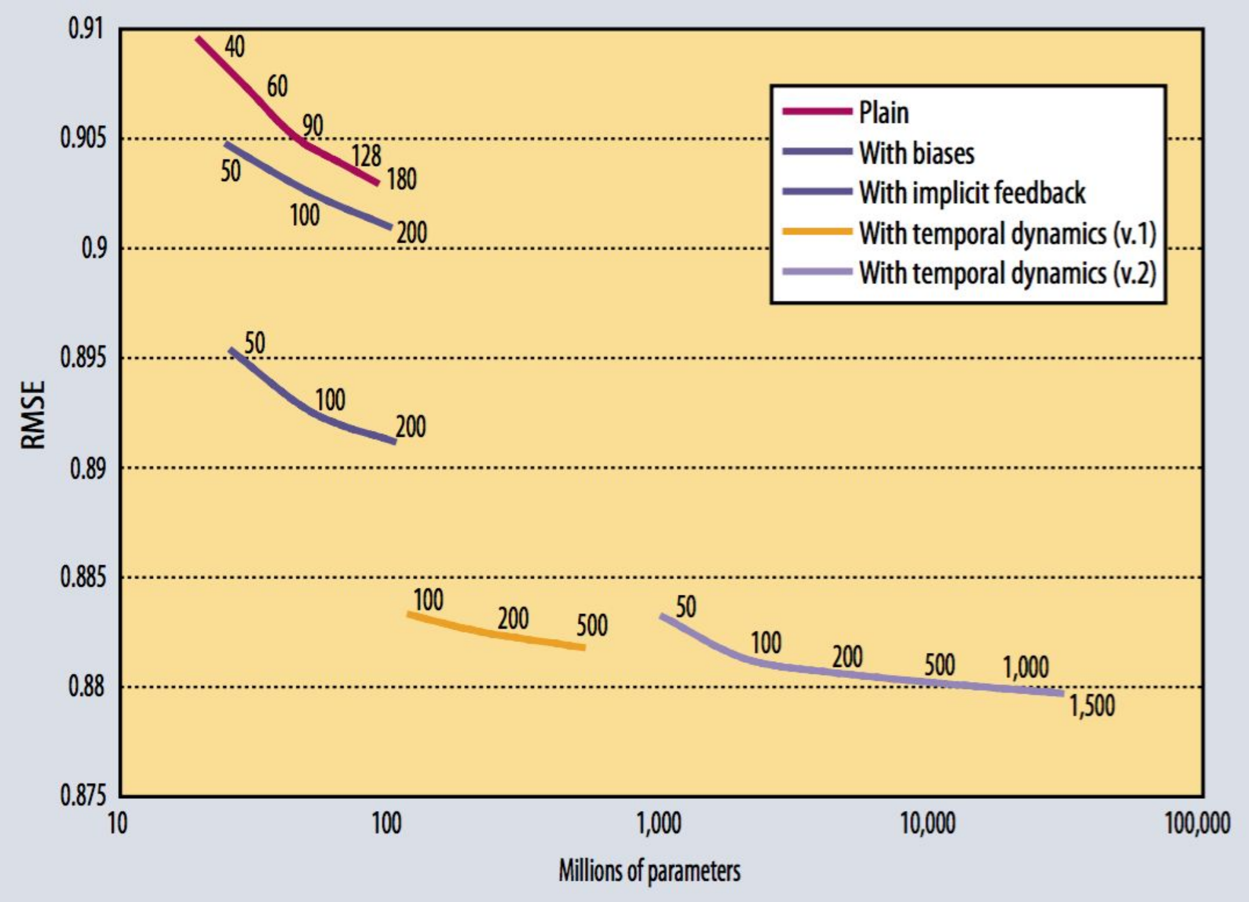
Ratings are now estimated as:

$$r_{ij} \approx \mu + b_i^* + b_j' + u_{i:} \cdot v_{:j}$$

The new cost function, with the biases included:

$$\arg\min_{U,V,b^*,b'} \sum_{i,j\in\mathcal{K}} (r_{ij} \underbrace{- \mu - b_i^* - b_j'}_{\text{New part!}} - u_{i:} \cdot v_{:j})^2 + \lambda_1(||u_{i:}||^2 + ||v_{:j}||^2) \underbrace{+ \lambda_2((b_i^*)^2 + (b_j')^2)}_{\text{New part!}}$$

From the paper: “Matrix Factorization Techniques for Recommender Systems”



Root mean square error over the Netflix dataset using various matrix factorization models.

Numbers on the chart denote each model’s dimensionality (k).

The more refined models perform better (have lower error).

Netflix’s inhouse model performs at RMSE=0.9514 on this dataset, so even the simple matrix factorization models are beating it!

Read the paper for details; it’s a good read!