

Recommender Systems

Jack Bennetto

October 13, 2016

Objectives

Today's objectives:

- Describe primary approaches to recommender systems
- Build a recommender using collaborative filtering and similarity
- Build a recommender using collaborative filtering and matrix factorization

Agenda

Today's plan:

1. Overview of types of recommender systems
2. Content-based recommender systems
3. Collaborative filtering with similarity
4. Collaborative filtering with matrix factorization
5. Best practices
6. Appendix

References

A couple references, from the machine learning perspective:

3

- [Mining of Massive Datasets](#)
- [Recommender Systems: An Introduction](#)
- [Music Recommendation and Discovery: The Long Tail, Long Tail, and Long Play in the Digital Music Space](#)
- [Matrix Factorization Techniques for Recommender Systems](#)
- [Amazon.com recommendations: Item-to-Item Collaborative Filtering](#)
- [Dato/GraphLab documentation & blog](#)

Introduction

4

5

Recommendation business problem

Recommendation problem takes several forms:

- Goal of recommender:
 - predict missing ratings
 - May be sufficient to just predict a subset of items with high expected rankings
 - May be sufficient to just predict general trends, such as *trending* news
- Long-tail:
 - Scarcity \Rightarrow brick & mortar stocks items based on average user
 - Online \Rightarrow cater to individual, not average user \Rightarrow stock everything, both popular and long tail
- Often described as *personalization*
- Examples: Movies (Netflix), Products (Amazon), Music (Pandora), and News articles (CNN)

Approaches to recommender systems

6

There are several approaches to building a recommender:

- Content-based: recommend based on properties/characteristics
- Collaborative filtering (CF): recommend based on similarity
- Hybrid: Content-based + Collaborative filtering
- Applications:
 - Product recommendations
 - Movie recommendations
 - News articles

Data

Typically, data is a *utility (rating)* matrix, which captures user preferences/well-being:

- User rating of items
- User purchase decisions for items
- Most items are unrated \Rightarrow matrix is sparse
- Unrated are coded as 0 or missing
- Use recommender:
 - Determine which attributes users think are important
 - Predict ratings for unrated items
 - Better than trusting 'expert' opinion

Types of data

Data can be:

8

- *Explicit:*
 - User provided ratings (1 to 5 stars)
 - User like/non-like
- *Implicit:*
 - Infer user-item relationships from behavior
 - More common
 - Example: buy/not-buy; view/not-view
- To convert implicit to explicit, create a matrix of 1s (yes) and 0s (no)

Example: explicit utility matrix

Example 9.1 in [Mining of Massive Datasets](#):

9

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3		5	1		3

Example: implicit utility matrix

Based on example 9.1 in [Mining of Massive Datasets](#):

10

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	1			1	1		
B	1	1	1				
C				1	1	1	
D		1		1	1		1

Content-Based Recommenders

11

12

Overview of content-based recommenders

Use features to determine similarity:

- Recommend based on item properties/characteristics
- Construct item *profile* of characteristics
- Construct item features:
 - Text: use TF-IDF and use top N features or features over a cutoff
 - Images: use tags -- only works if tags are frequent & accurate
- Compute document similarity: Jaccard, Cosine
- Construct user profile

Item profile

- Consists of (feature, value) pairs
- Consider setting feature to 0 or 1
- Consider how to scale non-Boolean features

13

14

User profile

- Describes user preferences (utility matrix)
- Consider how to aggregate item features per user:
 - Compute "weight" a user puts on each feature
 - E.g., "Julia Roberts" feature = average rating for films with "Julia Roberts"
- Normalize: subtract average utility per user
 - E.g., "Julia Roberts" feature = average rating for films with "Julia Roberts" - average rating

Content-based recommendations

- Compute (cosine) distance between user profile and item profiles
- May want to bucket items first using random-hyperplane and locality-sensitivity-hashing (LSH)
- ML approach:
 - Use random forest or equivalent to predict on a per-user basis
 - Computationally intensive -- usually only feasible for small problems

Collaborative filtering using similarity

Overview of CF using similarity

Use similarity to recommend items:

- Make recommendations based on similarity:
 - Between users
 - Between items
- Similarity measures:
 - Pearson
 - Cosine
 - Jaccard

Types of collaborative filtering

18

Two types of similarity-based CF:

- *User-based*: predict based on similarities between users
 - Performs well, but slow if many users
 - Use item-based CF if $|Users| \gg |Items|$
- *Item-based*: predict based on similarities between items
 - Faster if you precompute item-item similarity
 - Usually $|Users| \gg |Items| \Rightarrow$ item-based CF is most popular
 - Items tend to be more stable:
 - Items often only in one category (e.g., action films)
 - Stable over time
 - Users may like variety or change preferences over time
 - Items usually have more ratings than users \Rightarrow items have more stable average ratings than users

Collaborative filtering recipe

Compute predictions by similarity:

19

1. Normalize (demean) utility matrix
2. Reduce dimensionality: SVD, NMF, or UV (optional)
3. Compute similarity of users or items
4. Predict ratings for unrated items
5. Add prediction to average rating of user/item

20

Review: measuring similarity

Example 9.1 in [Mining of Massive Datasets](#):

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3		5	1		3

- What is the Jaccard distance between A & B? A & C?
- What is the Cosine distance between A & B? A & C?
- See text for examples with normalization and rounding

Choosing a similarity measure

21

Chose the appropriate similarity measure for your data:

- Cosine:
 - Use for ratings (non-Boolean) data
 - Treat missing ratings as 0
 - Cosine + de-meanned data is the same as Pearson
- Jaccard:
 - Use only Boolean (e.g., buy/not buy) data
 - Loses information with ratings data

Then compute *similarity matrix* of pair-wise similarities between items (users)

Predict ratings from similarity

22

Predict using a similarity-weighted average of ratings:

$$\hat{r}_{ui} = \frac{\sum_{j \in I_u} \text{similarity}(i, j) \cdot R_{uj}}{\sum_{j \in I_u} \text{similarity}(i, j)}$$

where

- \hat{r}_{ui} is user u 's predicted rating for item i
- $I_u \equiv$ set of items rated by u
- R_{uj} is utility matrix, i.e., $R_{uj} \equiv$ user u 's rating of item j

\Rightarrow Compute similarity between items!

Check for mastery

How would you modify the prediction formula below for a user-based recommender?

23

$$\hat{r}_{ui} = \frac{\sum_{j \in I_u} \text{similarity}(i, j) \cdot R_{uj}}{\sum_{j \in I_u} \text{similarity}(i, j)}$$

Hint: should you compute similarity between users or items?

Recommend best items

24

Recommend items with highest predicted rating:

- Sort predicted ratings \hat{r}_{ui}
- Optimize by only searching a neighborhood which contains the n items most similar to i
- Beware:
 - Consumers like variety
 - Don't recommend every Star Trek film to someone who liked first film
 - Best to offer several different types of item

Dimensionality reduction (optional)

May use SVD or similar method to reduce dimension:

25

```
U, Sigma, VT = np.linalg.svd(m_ratings)
# Set n_top_eig to capture most of the variance
m_sigma = np.mat(np.eye(n_top_eig) * Sigma[:n_top_eig])
m_new_ratings = m_ratings.T * U[:, :n_top_eig] * m_sigma.I
```

See [Application of Dimensionality Reduction in Recommender System -- A Case Study](#)

26

Collaborative filtering using matrix factorization

Collaborative filtering using matrix factorization

Predict ratings from *latent factors*:

- Compute latent factors q_i and p_u via matrix factorization
- *Latent factors* are unobserved user or item attributes:
 - Describe some user or item concept
 - Affect behavior
 - Example: escapist vs. serious, male vs. female films
- Predict rating: $\hat{r}_{ui} = q_i^T p_u$
- Assumes:
 - Utility matrix is product of two simpler matrices (long, thin):
 - \exists small set of users & items which characterize behavior
 - Small set of features determines behavior of most users
- Can use NMF, UV , or SVD

Review: SVD

28

Q: What is SVD?

Q: How do you compute it? (optional)

Q: How do you compute the variance in the data that a factor explains?

Q: What do the different matrices in decomposition represent?

Q: How can you use it to reduce dimensions?

Review: NMF

Q: What is NMF?

29

Q: How do you compute it?

Q: What do the different matrices in decomposition represent?

30

SVD vs. NMF

SVD:

- Must know all ratings -- i.e., no unrated items
- Assumes can minimize squared Frobenius norm
- Very slow if matrix is large & dense

NMF:

- Can estimate via alternating least squares (ALS) or stochastic gradient descent (SGD)
- Must regularize
- Can handle big data, biases, interactions, and time dynamics

Using NMF in recommendation systems

NMF is a 'best in class' option for many recommendation problems:

31

- Includes overall, user, & item bias as well as latent factor interactions
- Can fit via SGD or ALS
- No need to impute missing ratings
- Use regularization to avoid overfitting
- Can handle time dynamics, e.g., changes in user preferences
- Used by winning entry in Netflix challenge

NMF problem formulation

32

To factor the utility matrix:

$$\sum_{\{q_i, p_u\} \quad (u,i) \in \mathcal{R}} (r_{ui} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2)$$

where

- $\mathcal{R} \equiv$ all (u, i) in the training set with known ratings
- λ is amount of regularization
- r_{ui} is user u 's rating of item i
- p_u is latent factor for user u
- q_i is latent factor for item i

NMF problem formulation with bias

33

Should account for bias:

$$\sum_{\{q_i, p_u, \mu, b_u, b_i\} \mid (u,i) \in \mathcal{R}} (r_{ui} - \mu - b_u - b_i - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2)$$

where

- μ : overall bias (average rating)
- b_u : user bias
- b_i : item bias

Estimating NMF

34

Two methods to estimate NMF factors:

- Stochastic gradient descent (SGD):
 - Easier and faster than ALS
 - Must tune learning rate
 - Sometimes called 'Funk SGD' after originator
- Alternating least squares (ALS):
 - Use least squares, alternate between fixing q_i and p_u
 - Available in Spark/MLib
 - Fast if you can parallelize
 - Better for implicit (non-sparse) data
- Beware of local optima!

NMF ProTips

To get best performance with NMF:

- Model bias (overall, user, and item)
- Model time dynamics, such as changes in user preferences
- Add side or implicit information to handle cold-start
- See [Matrix Factorization Techniques for Recommender Systems](#)

35

36

Building a recommender with NMF

Use [GraphLab](#):

- Supports many types of recommenders
- Provides (near) best in class performance
- Reasonable licensing terms
- To improve performance, focus on:
 - Data collection and quality
 - Cold-start problem
 - Feature engineering

Best practices

37

38

Overview:

Will discuss:

- Cold-start problem
- Evaluation
- GraphLab ProTips
- (GraphLab) model selection

The cold-start problem

Difficult to build a recommender without ratings:

- *Cold-start* problem:
 - Need utility matrix to recommend
 - Can ask users to rate items
 - Infer ratings from behavior, e.g., viewing an item
- Must also handle new users and new items
- Approaches:
 - Use ensemble of (bad) recommenders until you have enough ratings
 - Use content-based recommender
 - Exploit implicit, tag, and other side data
 - Use `ItemSimilarityModel` until you have enough rating data

Evaluation issues

40

Choose right evaluation criteria:

- Historically, used RMSE or MAE
- But, only care about predicting top n items
 - Should you compute metric over all missing ratings in test set?
 - No need to predict items undesirable items well
- *Precision at n* : percentage of top n predicted ratings that are 'relevant'
- *Recall at n* : percentage of relevant items in top n predictions
- Lift or hit rate are more relevant to business

Evaluation issues

41

Evaluation is difficult:

- Performance of recommender should be viewed in context of *user experience* (UX)
- \Rightarrow run A/B test on entire system
- Cross validation is hard:
 - What do you use for labels because of missing data?
 - Users choose to rate only some items \Rightarrow selection bias
 - Not clear how to fix this bias, which is always present
- Beware of local optima \Rightarrow use multiple starts

Cross-validation

Cross-validation (for item-based recommender):

42

- Randomly sample ratings to use in training set
- Split on users
- Be careful if you split temporally
- Do not split on items

43

Recommender issues

Building a production recommender is also challenging:

- Part of entire UX
- Should consider:
 - Diversity of recommendations
 - Privacy of personal information
 - Security against attacks on recommender
 - Social effects
 - Provide explanations
- See [Recommender systems: from algorithms to user experience](#)

GraphLab ProTips

44

GraphLab provides best in class performance:

- Start with `MatrixFactorizationModel`:
 - Switch to `LinearRegressionModel` if too slow
 - Switch to `FactorizationModel` if need interactions
- Focus on cold-start and side information to obtain best performance
- Tune settings with
`graphlab.toolkits.model_params_search()`
- Compare models with
`graphlab.recommender.util.compare_models()`

Select model based on data and business metric:

- For best ranking performance:
 - Use `ItemSimilarityModel`, `MatrixFactorizationModel`, or `FactorizationModel`
 - Set `ranking_regularization` $\in (0, 1)$
 - With implicit data, add rating column of 1s and set `unobserved_rating_value=0`
- For best ratings prediction with real ratings:
 - Use `MatrixFactorizationModel`, `FactorizationModel`, or `LinearRegressionModel`
 - `LinearRegressionModel` uses user & item features and user & item popularity bias
 - Matrix models add user & item latent factors
 - `FactorizationModel` adds interaction between latent and side features

Dato documentation

Dato's documentation is excellent:

- [Documentation](#)
- [Basic example](#)
- [Million song example](#):

46

47

Computational

Computation tips:

- Compute offline:
 - Matrix factorization
 - Similarity matrix
 - User/item neighborhoods (via clustering)
- Compute predicted ratings/rankings live

Summary

You should now be able to explain:

- Content-based vs. collaborative filtering recommenders?
- Item-based vs. user-based CF?
- Compute measures of similarity (Jaccard, Pearson, cosine)?
- State which GraphLab recommender model is right for which problem?
- Describe how to tune and evaluate a recommender?
- Explain how to overcome the cold-start problem

Appendix: similarity measures

49

50

Similarity measures

Recommenders use distance to quantify similarity:

- Cosine similarity:

- $\text{cosine}(\mathbf{x}, \mathbf{y}) = \cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$
- $\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{1}{2} + \frac{1}{2} \cdot \text{cosine}(\mathbf{x}, \mathbf{y})$
- Same as Pearson if you de-mean data
- Treat blanks as 0

- Jaccard distance:

- Jaccard index: $J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$
- Jaccard distance: $d_J(A, B) = 1 - J(A, B)$
- Use for binary data
- Loses information with non-Boolean data
- Example:
 - Let $U_k \equiv \{i \in \text{Users} \mid R_{ik} \neq 0\}$, i.e. user i rated item k
 - $\text{similarity}(a, b) = J(U_a, U_b) = \frac{|U_a \cap U_b|}{|U_a \cup U_b|}$

Other distance measures

Two other measures of similarity:

- Similarity:
 - Constructed from Euclidean distance so $\text{similarity}(x, y) \in (0, 1)$
 - $\text{similarity}(x, y) = \frac{1}{1 + \| \mathbf{x} - \mathbf{y} \|}$
- Pearson correlation: $\text{pearson}(x, y) = \frac{(x, y)}{\sigma(x) \cdot \sigma(y)}$
 - Renormalize to be in (0,1):
 $\text{similarity}(x, y) = \frac{1}{2} + \frac{1}{2} \cdot \text{pearson}(x, y)$
 - Use Numpy `corrcoef()`

Appendix: matrix factorization

52

53

Review: matrix factorization (1/4)

Use matrix factorization to predict ratings:

- Discover *latent factors*, unobserved characteristics which determine behavior
- Reduce dimension
- Consider: SVD, UV, or NMF

Review: SVD (2/4)

Decompose rating matrix, M , into $U \cdot \Sigma \cdot V^T$

- U : $m \times d$ unitary matrix, represents user latent factors
- Σ :
 - $d \times d$ diagonal matrix of singular values
 - Σ^2 is the variance of each factors
- V^T :
 - $d \times n$ matrix
 - Transpose of item latent factors
- Keep only factors which explain the top ~90% of variance
- Caveat: doesn't work with missing values

Review: UV (3/4)

Decompose rating matrix, M , into $U \cdot V$

55

- U : $m \times d$ unitary matrix, represents user latent factors
- V : $d \times n$ matrix
- Can fit via stochastic gradient descent or alternating least squares (ALS)
- Use regularization to avoid overfitting

56

U · V decomposition

M is an m by n matrix

- $M \approx U \cdot V$, U is m by d and V is d by n
- Use entries from $U \cdot V$ to predict missing ratings
- Fit by minimizing RMSE of $M - U \cdot V$:
 - Has multiple local optima
 - Use multiple starts & algorithms
 - Start from $\sqrt{\frac{(\{m_{ij} \in M | m_{ij} \neq 0\})}{d}}$
 - Perturb for other starts
 - Vary path for visiting elements during optimization
 - Compute via ALS or update rule
 - Minimize RMSE of $\sum (m_{ij} - (U \cdot V)_{ij})^2$
 - Overfitting
 - Use (stochastic) gradient descent to optimize

Review: NMF (4/4)

57

Non-negative matrix factorization :

- Includes overall, user & item bias as well as latent factor interactions
- Can fit via stochastic gradient descent or alternating least squares (ALS)
- Use regularization to avoid overfitting
- Used by winning entry in Netflix challenge

58