

1 Module 19: Recommendation Systems

Contents

1	Module 19: Recommendation Systems	1
1.1	Content-based Filtering	1
1.2	Collaborative Filtering	1
1.2.1	Funk SVM	1
1.3	The SURPRISE Library	2
1.4	Hybrid Recommender Systems	3

1.1 Content-based Filtering

Leverage similarities in *products, services or content*.

Filtering: used predicted ratings as a way to filter out (unwanted) choices.

Content-based: filtering is performed on content items i.e. music, shows, movies, etc.

Usually applied as a categorical filter. e.g. with two music genres Rock/EDM, 5 users' ratings (usually data is incomplete) and a list of a band's proximity to their genre – **item factor** – (i.e. Odesza is .95 correlation with EDM, etc.). Then, we can simply fit a linear regression.

The major disadvantage is that item factors must be manually collected (i.e. use just "Rock", "EDM" which misses nuance/hidden features) for each item of content.

1.2 Collaborative Filtering






Leverage *similar users' preferences* to serve appropriate content.

Generate N random item factors. Fit a linreg to the N factors to each user (will be garbage). Use those predictions to fit the item factors themselves. Then repeat steps 2,3 iteratively to minimize loss.

This is essentially inefficient SQD. In which case, it's more efficient to do a single gradient descent. This will be matrix multiplication.

1.2.1 Funk SVM

(is technically not actually SVD; normal SVD does not work with missing values)

$Q: Z \times N$						$P: M \times Z$		
	Ommazh	Melt-Banana	BTS	Zhou Shen	Sanam		F1	F2
								
F1	-1.53	-1.77	1.16	1.09	0.73	An	-1.52	1.35
F2	1.77	1.88	1.45	1.74	1.69	Bhavana	-1.51	1.11
						Cordelia	1.15	2.16
						Diego	1.19	2.15

e.g. with this example, N (number of content items) = 5, M (number of users) = 4, Z (somewhat arbitrary number of item factors) = 2

Predictions for any given user \rightarrow any given item $\hat{r}_{i,j} = \text{row}_i(P) \cdot \text{col}_j(Q)$. Allow error against ground truth $r_{i,j}$ to be $e_{i,j}^2 = (\hat{r}_{i,j} - r_{i,j})^2$.

e.g. Cordelia's rating of Melt Banana:

$$\hat{r}_{i,j} = (1.15, 2.16) \cdot (-1.77, 1.88) = 2.0253$$

$$e_{i,j}^2 = (2 - 2.0253)^2 = \text{smol}$$

Of course, we would need this for all combinations. Not all users rate every item, so we allow columns j to subset R_i , the available user ratings of an artist.

$$\text{MSE} = \sum_{i=1}^M \sum_{j \in R_i}^N e_{i,j}$$

We want to pick values $Q_{a,b}$ and $P_{a,b}$ in the matrices P, Q which minimize the MSE.

For each gradient descent step with some learning rate α :

$$P_{a,b} = P_{a,b} - \alpha \sum_{j \in R_a}^N e_{a,j} Q_{b,j}$$

Making use of the fact $\frac{\partial e_{i,j}^2}{\partial_{a,b}} = 2e_{a,j}Q_{b,j}$

1.3 The SURPRISE Library

```
from surprise import Dataset, Reader
from surprise import SVD
```

```
# Instantiate reader/dataset
reader = Reader(rating_scale = (0.5))
```

```
sf = Dataset.load_from_df(df[['UserID', 'itemID', 'rating']], reader)

# Build training data
training_data = sf.build_full_dataset()

# Create/fit model
model = SVD(n_factors, n_epochs = 10_000, biased = False)
model.fit(training_data)

# Single prediction
model.predict('an', 'ommazh')

# Multi-prediction
test_data = training_data.build_testset()
predictions_list = model.test(test_data)

# Get all predictions with
model.pu @ model.qi.T
```

Other matrix factorization strategies

- SVD++: optimizes with implicit feedback to improve prediction accuracy
- Regularized SVD: probability-based algorithm to compute near-optimal low-rank SVD of large amts of data with high accuracy (treat data as compressed)
- Iterative SVD: Estimate SVD of an incomplete matrix using first-order optimization over orthogonal manifolds
- Probabilistic Matrix Factorization (PMF): linearly combine item factor vectors with user-specific factors in a linear factor model (best used on huge, sparse, imbalanced datasets)
- Non-negative Matrix Factorization (NMF): extract principal components of a set of non-negative data vectors automatically
- Bayesian Probabilistic Matrix Factorization (BPMF): integrate all model parameters/hyperparameters to avoid tuning and extend to recommendations where top N queries are recommended to users

1.4 Hybrid Recommender Systems

Combine multiple strategies, using advantages in different ways.

Recommendation strategies

- Predict items that are given high ratings
- Similar/Dissimilar to previously viewed/rated items

- Promoted by advertisers
- Popular/Recently added to service
- Regular purchases that the user will need soon

The matrix Q (`model.qi`) is the item factors (columns) which can be shown N-dimensionally. Just compute the distance of the individual objects (rows) from each other.