Matrix Factorization Methods

Inspiration: Content Based Preference

User content preferences:

	Action	Strategy	Story	Exploration	Collection
Matt	3	3	1	4	2
Caitlyn	1	4	4	2	5

Call this matrix

U

Item content attributes

	Action	Strategy	Story	Exploration	Collection
Zelda	4	3	2	5	3
Mario	5	2	1	3	3
Animal Crossing	1	2	3	2	5

Call this matrix

V

Overall preference of user for items is a dot-product

$$\text{pref}(\text{Matt}, \text{Zelda}) = 3\times4 + 3\times3 + 1\times2 + 4\times5 + 2\times3 = 49$$

$$\text{pref}(\text{Caitlyn}, \text{Animal Crossing}) = 1\times1 + 4\times2 + 4\times3 + 2\times2 + 5\times5 = 50$$

$$\text{pref}(\text{Matt}, \text{Animal Crossing}) = 3\times1 + 3\times2 + 1\times3 + 4\times2 + 2\times5 = 33$$

Idea: Is it possible to learn U and V when we take ratings as an expression of preferences?

Matrix Factorization For Explicit Ratings

If we take ratings as an expression of preferences, the our content based setup results in the matrix equation:

$$\hat{R} = UV^t$$

U is (# users) $\times$ K

V is (# items) $\times$ K

K is a <u>hyperparameter</u>

So each predicted rating is a dot product

$$\hat{r}_{ij} = \sum_{K} u_{iK} v^t_{Kj} = \sum_{K} u_{iK} v_{jK}$$

To <u>learn</u> U and V, we want this to accurately reproduce the ratings we <u>know</u>:

$$R \approx UV^t$$

← Remember, a lot of R is <u>missing</u>, so this equation only applies to the non-missing values.

The next step is familiar, we need to measure the quality of our predictions, and we use least squares:

$$\hat{U}, \hat{V} = \underset{U,V}{\text{argmin}} \left\{ \sum_{r_{ij} \text{ ratings in } R} \left(r_{ij} - \sum_{K} u_{iK} v_{jK} \right)^2 \right\}$$

This problem is easily solved with gradient descent, which has very simple update rules:

$$\frac{\partial L}{\partial u_{iK}} = 2 \sum_{r_{ij}} (r_{ij} - \hat{r}_{ij}) v_{jK}$$

$$\frac{\partial L}{\partial v_{iK}} = 2 \sum_{r_{ij}} (r_{ij} - \hat{r}_{ij}) u_{iK}$$

These are the components of the gradient of L.

Comments

① We are estimating (# users + # items) × K parameters, which is a lot.

So, regularization is useful:

$$\hat{U}, \hat{V} = \underset{U,V}{\operatorname{argmin}} \left\{ \sum_{r_{ij}} \left(r_{ij} - \vec{u}_i \cdot \vec{v}_j \right)^2 + \lambda \left(\sum_{i,k} u_{ik}^2 + \sum_{j,k} v_{jk}^2 \right) \right\}$$

This doesn't affect the difficulty of fitting the model with gradient descent.

② K is a hyper parameter, it must be tuned with cross validation.

But, be careful about removing <u>all</u> ratings for a user or item!

③ Similarly, matrix factorization cannot provide ratings for new users or items. You need a fallback methodology for these cases!

④ Some users/items have different ranges for ratings

- Some users rate everything 4 or 5 stars
- Some products are garbage, and are always rated 1 or 2 stars.

You can account for this with user and item level parameters:

$$\hat{r}_{ij} = \underbrace{\vec{u}_i \cdot \vec{v}_j}_{} + b_i^u + b_j^v + \mu$$

overall average rating

item j's deviation from the average

user i's deviation from the average

The additional signal of user i's preference for item j.