Latent Factor Models in Recommender System and Market Segmentation Through Clustering

Joy Zeng

Ohio State University

zeng.341@osu.edu

March 16, 2017

Overview

Introduction to Recommender System

General Settings of Recommender Systems:

- Recommender System is introduced to predict user preference and recommend personalized items
- A typical database usually contains ratings given by n users to m items
- User ratings are then translated to a rating matrix $\mathbf{R}$, where each entry r_{ij} represents the rating user i gave to product j

Table: Small Example Showing the Ratings of Three Users on Four Items

User (i)	Item (j)	ID	Rating (1-5)	Reviews
Kate	Apple	1	3	They taste ok
Jenny	Apple	2	4	I love apples
Jenny	Chocolate	3	2	awful
Alex	Banana	4	5	Great!
Alex	Yogurt	5	4	I like it

Introduction to Recommender System (Continue)

Table: An Example User-Item Rating Matrix

	Apple (v_1)	Chocolate (v_2)	Banana (v_3)	Yogurt (v_4)
Kate (u_1)	3	missing	missing	missing
Jenny (u_2)	4	2	missing	missing
Alex (u_3)	missing	missing	5	4

- The rating matrix is sparse
- We can recommend items to each user if we can estimate the missing ratings

Models for Collaborative Filtering

The Baseline Model, $\mathcal{M}_1$

$\hat{r}_{ij} = \mu + \alpha_i + \beta_j$
where μ is the overall average rating, and $\hat{r}_{ij}$ are the predicted ratings

The Latent Factor Model, $\mathcal{M}_2$

$\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{v}_j$
$L = \sum_{(i,j)\in\mathcal{K}} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda(||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2)$
where λ is shrinkage penalty, or regularization coefficients

Latent Factor Model With Intercept And Bias Terms, $\mathcal{M}_3$

$\hat{r}_{ij} = \mu + \alpha_i + \beta_j + \mathbf{u}_i^T \mathbf{v}_j$
$L = \sum_{(i,j)\in\mathcal{K}} (r_{ij} - \hat{r}_{ij})^2 + \lambda(||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2 + \alpha_i^2 + \beta_j^2)$
where μ is the overall average rating

Models for Collaborative Filtering ($\mathcal{M}_1$)

- A baseline model, $\mathcal{M}_1$ (a two-way ANOVA model), is introduced to establish a basis for comparisons of model performance.

$$\hat{r}_{ij} = \mu + \alpha_i + \beta_j \tag{1}$$

  where μ is the overall average rating, and $\hat{r}_{ij}$ is the predicted ratings. More specifically, α_i and β_j are the effect of user i and the effect of item j, respectively.

- The parameter estimates are:

$$\hat{\mu} = \bar{r}, \hat{\alpha}_i = \bar{r}_{i.} - \bar{r}, \hat{\beta}_j = \bar{r}_{.j} - \bar{r} \tag{2}$$

Models for Collaborative Filtering (continued)

- Generally, we will estimate our ratings $\hat{r}_{ij}$ using a training set, $\mathcal{K}$, and evaluate model performance on a test set $\mathcal{T}$
- The Root-Mean-Square Error (RMSE) is a frequently used measurement of the performance of models, defined as:

$$RMSE = \sqrt{\frac{\sum_{(i,j)\in\mathcal{T}} I_{ij}(r_{ij} - \hat{r}_{ij})^2}{\sum_{(i,j)\in\mathcal{T}} I_{ij}}}, \tag{3}$$

where $\mathcal{T}$ is the test set, which is a subset of the full dataset used to test the model performance. I_{ij} is an indicator function such that $I_{ij} = 1$ if user i rated product j; and $I_{ij} = 0$ otherwise

Models for Collaborative Filtering ($\mathcal{M}_2$)

- Latent factor approaches relate users and items in a low-dimensional latent feature space of dimension d
- The models assume that each user i and each item j are associated with a user feature vector $\mathbf{u}_i \in \mathbb{R}^d$ and an item feature vector $\mathbf{v}_j \in \mathbb{R}^d$
- The model captures the latent relationship between users and items inferred from rating patterns

Models for Collaborative Filtering ($\mathcal{M}_2$)

Latent Factor Model $\mathcal{M}_2$:

$$\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{v}_j \tag{4}$$

Loss function over the observed ratings on the training set is defined as:

$$L = \sum_{(i,j)\in\mathcal{K}} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda(||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2) \tag{5}$$

where λ is a shrinkage penalty, or regularization coefficient.

Models for Collaborative Filtering ($\mathcal{M}_3$)

Taking user bias and item bias into account, we assume that each user has an associated user bias α_i and each item has an associated item bias β_j:

$$\hat{r}_{ij} = \mu + \alpha_i + \beta_j + \mathbf{u}_i^T \mathbf{v}_j \tag{6}$$

where μ is the overall average rating.
The loss function is defined as:

$$L = \sum_{(i,j)\in\mathcal{K}} (r_{ij} - \hat{r}_{ij})^2 + \lambda(||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2 + \alpha_i^2 + \beta_j^2) \tag{7}$$

Models for Collaborative Filtering (Model Training)

Model training:

- Learning latent factor models requires finding $\mathbf{u}_i$ and $\mathbf{v}_j$ to minimize the loss
- Closed-form solutions for the estimates of $\mathbf{u}_i$ and $\mathbf{v}_j$ do not exist $\Rightarrow$ numerical approaches are needed to find an approximate numerical solution for the estimate feature vectors

The difficulties in model training:

- computationally expensive
- many parameters to estimate
- tuning λ

Gradient Descent (GD) and Stochastic Gradient Descent (SGD) are among the most widely used iterative methods in minimizing loss functions, and can be employed in training the models

Gradient Descent(GD)

Each training sample is a tuple $(\mathbf{x}^{(t)}, y^{(t)})$ and the loss is a measure of the distance between the model predictions $g_\theta(\mathbf{x})$ and the truth y, defined as $L_\theta = L(g_\theta(\mathbf{x}), y) = \frac{1}{T}\sum_{t=1}^{T} l(g_\theta(\mathbf{x}^{(t)}), y^{(t)})$, where $\theta = \{\theta_1, \theta_2, ..., \theta_p\}$ are model parameters and T is the number of training samples

GD

$$\theta_s \leftarrow \theta_s - h[\frac{1}{T}\sum_{t=1}^{T} \nabla_{\theta_s} l(g_\theta(\mathbf{x}^{(t)}), y^{(t)})], s = 1, ..., p \tag{8}$$

where h is the step size

Stochastic Gradient Descent(SGD)

Each training sample is a tuple $(\mathbf{x}^{(t)}, y^{(t)})$ and the loss is a measure of the distance between the model predictions $g_\theta(\mathbf{x})$ and the truth y, defined as $L_\theta = L(g_\theta(\mathbf{x}), y) = \frac{1}{T}\sum_{t=1}^{T} l(g_\theta(\mathbf{x}^{(t)}), y^{(t)})$, where $\theta = \{\theta_1, \theta_2, ..., \theta_p\}$ are model parameters and T is the number of training samples

SGD

$$\theta_s \leftarrow \theta_s - h\nabla_{\theta_s} l(g_\theta(\mathbf{x}^{(t^*)}), y^{(t^*)}), s = 1, ..., p \tag{9}$$

where t^* is the randomly selected training sample in each iteration

GD and SGD in Simple Linear Regression

Example: $y = a + bx$

Goal: find the values of a and b that minimize $L = \sum_{t=1}^{T}(y^{(t)} - a - bx^{(t)})^2$

$\nabla_a = \frac{\partial L}{\partial a} = -2\sum_{t=1}^{T}(y^{(t)} - a - bx^{(t)})$

$\nabla_b = \frac{\partial L}{\partial b} = -2\sum_{t=1}^{T}[(y^{(t)} - a - bx^{(t)})x^{(t)}]$

GD Algorithm

```
repeat until convergence {
$a = a - h\nabla_a;$
$b = b - h\nabla_b;$ }
```

SGD Algorithm

```
repeat until convergence {
$a = a - h[-(y^{(t^*)} - a - bx^{(t^*)})];$
$b = b - h[-(y^{(t^*)} - a - bx^{(t^*)})x^{(t^*)}];$ }
```

SGD Algorithm for $\mathcal{M}_2$

Algorithm 3 Learning algorithm for $\mathcal{M}_2$ using SGD. Each training sample is a tuple (UserID, ProductID, rating), (i_t, j_t) denotes the user i and item j in the training sample t, r_{i_t,j_t} denotes the rating of user i on product j for the training sample t, n_u is the number of unique users in the dataset, m_v is the number of unique items in the dataset

```
 1: Initialize $[\mathbf{U}]_{ik}^T := \sqrt{\frac{\bar{R}}{d}} + \epsilon(r)$, $i = 1, ..n_u$, $k = 1, ..., d$,        ▷ $\epsilon(r) \sim U[-0.1, 0.1]$
 2: Initialize $[\mathbf{V}]_{kj} := \sqrt{\frac{\bar{R}}{d}} + \epsilon(r)$, $j = 1, ..m_v$, $k = 1, ..., d$
 3: for $T = 1$ to number of epoch do
 4:     Permute the training sample index
 5:     for each training sample t in the permuted order do
 6:         $e := r_{i_t,j_t} - \mathbf{u}_{i_t}^T \mathbf{v}_{j_t}$
 7:         Update:
 8:         $\mathbf{u}_{i_t} \leftarrow \mathbf{u}_{i_t} - h[-e\mathbf{v}_{j_t} + \lambda \mathbf{u}_{i_t}]$        ▷ $h$ is the step size
 9:         $\mathbf{v}_{j_t} \leftarrow \mathbf{v}_{j_t} - h[-e\mathbf{u}_{i_t} + \lambda \mathbf{v}_{j_t}]$        ▷ update $\mathbf{u}_{i_t}$, $\mathbf{v}_{j_t}$
10:     end for
11:     Compute training loss and training RMSE based on all the observed ratings
        in the training sample
12: end for
```

SGD Algorithm for $\mathcal{M}_3$

Algorithm 4 Learning algorithm for $\mathcal{M}_3$ using SGD.

```
1:  Initialize $[\mathbf{U}]^T_{ik} := \sqrt{\frac{\bar{R}}{d}} + \epsilon(r)$, $i = 1, ..n_u$, $k = 1, ..., d$,    ▷ $\epsilon(r) \sim U[-0.1, 0.1]$
2:  Initialize $[\mathbf{V}]_{kj} := \sqrt{\frac{\bar{R}}{d}} + \epsilon(r)$, $j = 1, ..m_v$, $k = 1, ..., d$
3:  Initialize $\alpha_i \sim U[-0.1, 0.1]$, $i = 1, ..n_u$
4:  Initialize $\beta_j \sim U[-0.1, 0.1]$, $j = 1, ..m_v$
5:  Initialize $\mu \sim U[-0.1, 0.1]$
6:  for $T = 1$ to *number of epoch* do
7:      Permute the training sample index
8:      for each training sample t in the permuted order do
9:          $e := r_{i_t, j_t} - \mathbf{u}^T_{i_t}\mathbf{v}_{j_t}$
10:         Update:
11:         $\mathbf{u}_{i_t} \leftarrow \mathbf{u}_{i_t} - h[-e\mathbf{v}_{j_t} + \lambda\mathbf{u}_{i_t}]$    ▷ $h$ is the step size
12:         $\mathbf{v}_{j_t} \leftarrow \mathbf{v}_{j_t} - h[-e\mathbf{u}_{i_t} + \lambda\mathbf{v}_{j_t}]$    ▷ update $\mathbf{u}_{i_t}$, $\mathbf{v}_{j_t}$
13:         $\alpha_{i_t} \leftarrow \alpha_{i_t} - h[-e + \lambda\alpha_{i_t}]$
14:         $\beta_{j_t} \leftarrow \beta_{j_t} - h[-e + \lambda\beta_{j_t}]$
15:         $\mu \leftarrow \mu + h * e$
16:     end for
17:     Compute *training loss* and *training RMSE* based on all of the observed ratings
    in the training sample
18: end for
```

Returning to our small example illustrating $\mathcal{M}_3$

$$\mathbf{R} := \begin{bmatrix} 3 & \textit{missing} & \textit{missing} & \textit{missing} \\ 4 & 2 & \textit{missing} & \textit{missing} \\ \textit{missing} & \textit{missing} & 5 & 4 \end{bmatrix}$$

- $n_u = 3$ and $m_v = 4$
- $\mathbf{U}^T$ is a $3 \times d$ matrix and $\mathbf{V}$ is a $d \times 4$ matrix, where d is the number of latent dimensions
- The overall average rating is $\bar{R} = 3.6$

Returning to our small example illustrating $\mathcal{M}_3$ (continued)

Suppose the number of latent dimensions $d = 1$. Our initialization is:

$$\mathbf{U}^T := \begin{bmatrix} \sqrt{\frac{\bar{R}}{1}} + U[-0.1, 0.1] \\ \sqrt{\frac{\bar{R}}{1}} + U[-0.1, 0.1] \\ \sqrt{\frac{\bar{R}}{1}} + U[-0.1, 0.1] \end{bmatrix}_{3\times 1} := \begin{bmatrix} \sqrt{3.6} + U[-0.1, 0.1] \\ \sqrt{3.6} + U[-0.1, 0.1] \\ \sqrt{3.6} + U[-0.1, 0.1] \end{bmatrix},$$

$$\mathbf{V} := \begin{bmatrix} \sqrt{\frac{\bar{R}}{1}} + U[-0.1, 0.1] & \sqrt{\frac{\bar{R}}{1}} + U[-0.1, 0.1] & \sqrt{\frac{\bar{R}}{1}} + U[-0.1, 0.1] & \sqrt{\frac{\bar{R}}{1}} + U[-0.1, 0.1] \end{bmatrix}_{1\times 4}$$

$$= \begin{bmatrix} \sqrt{3.6} + U[-0.1, 0.1] & \sqrt{3.6} + U[-0.1, 0.1] & \sqrt{3.6} + U[-0.1, 0.1] & \sqrt{3.6} + U[-0.1, 0.1] \end{bmatrix}_{1\times 4}$$

$$\alpha_i := U[-0.1, 0.1],\ i = 1, 2, 3,$$

$$\beta_j := U[-0.1, 0.1],\ j = 1, ..., 4,$$

$$\mu := U[-0.1, 0.1]$$

Returning to our small example illustrating $\mathcal{M}_3$ (continued)

In each epoch, the following two procedures are performed:

1. Permute the training samples by their unique ID. An example of a permuted dataset is given below:

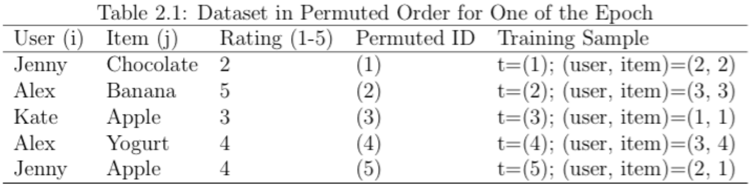


Table 2.1: Dataset in Permuted Order for One of the Epoch

User (i)	Item (j)	Rating (1-5)	Permuted ID	Training Sample
Jenny	Chocolate	2	(1)	t=(1); (user, item)=(2, 2)
Alex	Banana	5	(2)	t=(2); (user, item)=(3, 3)
Kate	Apple	3	(3)	t=(3); (user, item)=(1, 1)
Alex	Yogurt	4	(4)	t=(4); (user, item)=(3, 4)
Jenny	Apple	4	(5)	t=(5); (user, item)=(2, 1)

2. Then for each of the training sample $(u_i, v_j)^{(t)}$ in the permuted order t, where $t = (1), ..., (5)$ in the example above, the algorithm automatically updates its corresponding $\mathbf{u}_i$, $\mathbf{v}_j$, α_i, β_j and μ.
3. For instance, when $t = (1)$, the training sample is (Jenny, Chocolate) $= (2, 2)$. It is used to compute the gradients for updating parameters $\mathbf{u}_2$, $\mathbf{v}_2$, α_2, β_2 and μ.

Returning to our small example illustrating $\mathcal{M}_3$ (continued)

By setting step size $h = 0.07$, $\lambda = 0.001$, after 5 epochs, the predicted matrix is:

Table: Recovered Rating Matrix

	Apple	Chololate	Banana	Yogurt
Kate	3.000352 (3)	1.266848	3.119489	2.363782
Jenny	3.998818 (4)	2.000702 (2)	4.156005	3.273350
Alex	4.806945	2.564952	4.999202 (5)	3.999546 (4)

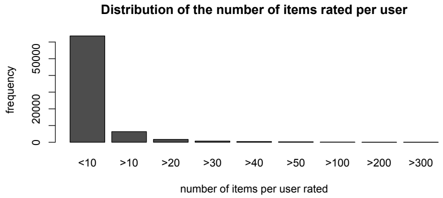
The Amazon Fine Food Reviews Dataset

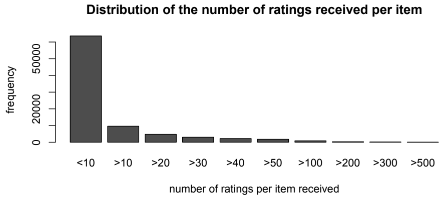
- Dataset consists of 568,454 user ratings and text reviews of food products from Amazon

Column Index	Variable Name	Variable Description
1	Id	unique identifier
2	ProductId	unique identifier for the product
3	UserId	unique identifier for the user
7	Score	rating scaled from 1 to 5
9	Summary	brief summary of the text review
10	Text	text review

- There are $256,059$ unique users and $74,258$ unique products in the dataset
- Converting the dataset to a user-item rating matrix, the matrix is highly sparse since only a very small fraction of entries have ratings $(568,454/19,014,429,222 = 0.002989593\%)$

The Amazon Fine Food Reviews Dataset (continued)





we have very little information about how each user rated the items as most of the users only rated less than 10 items among the list of items. Thus, recovering the whole rating matrix is a very difficult task

What I've investigated

What I've investigated when training the models:

- Different initialization rules
- Sensitivity analysis
- Data splitting
- Tuning λ

Experiments on initialization rules for $\mathcal{M}_3$

We compare the results of two different initialization rules using model $\mathcal{M}_3$:

1. All the entries in $[\mathbf{U}]^T_{ik}$, $[\mathbf{V}]_{kj}$ are initialized by $\sqrt{\frac{\bar{R}}{d}}$, all the α_i, β_j and μ are originally assigned to be zero
2. Same as (i) but adding a random noise $\epsilon(r) \sim U[-0.1, 0.1]$ to each term in the first rule

We trained model $\mathcal{M}_3$ on the first 20000 data points using 1000 epochs with step size $h = 0.02$ and see whether the algorithm converges to a minimum.

Sensitivity analysis for different starting values on $\mathcal{M}_3$

Sensitivity analysis on different starting values is needed to provide evidence for the robustness of the training algorithm.
3 parallel experiments:

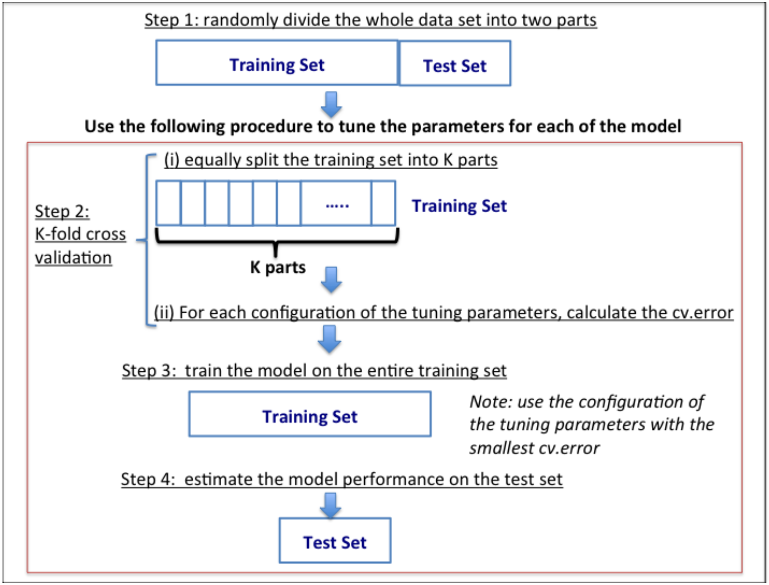
1. 10 experiments starting from $\bar{R} - 1$: for the initial values of **U** and **V**, we assign every entry in $[\mathbf{U}]_{ik}^T$, $[\mathbf{V}]_{kj}$ to be $\sqrt{\frac{\bar{R}-1}{d}} + \epsilon(r)$, where $\epsilon(r)$ is random noise chosen from $U[-0.1, 0.1]$.
2. 10 experiments starting from around $\bar{R}$: $[\mathbf{U}]_{ik}^T$ and $[\mathbf{V}]_{kj}$ are both initialized by $\sqrt{\frac{\bar{R}}{d}} + \epsilon(r)$
3. 10 experiments starting from $\bar{R} + 1$: both $[\mathbf{U}]_{ik}^T$ and $[\mathbf{V}]_{kj}$ start from $\sqrt{\frac{\bar{R}+1}{d}} + \epsilon(r)$

In all 30 experiments α_i, β_j, and μ are initialized with small random values randomly sampled from the uniform distribution on the interval $[-0.1, 0.1]$.

Data splitting

- Randomly select 60% of the data from the entire dataset to be in the training set and assign the rest to be in the test set
- Users and items that do not appear in the training set are removed from the test set
- We first fit $\mathcal{M}_1$ using the training set and compute the test RMSE to get a performance baseline

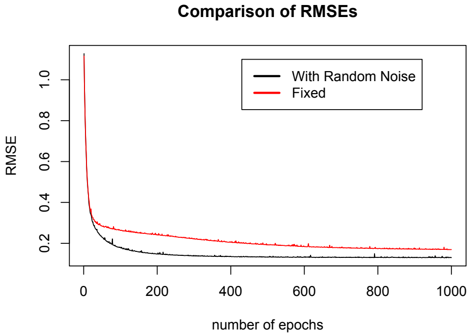
Models for Collaborative Filtering (Model training)


Step 1: randomly divide the whole data set into two parts
Training Set
Test Set
Use the following procedure to tune the parameters for each of the model
Step 2: K-fold cross validation
(i) equally split the training set into K parts
......
Training Set
K parts
(ii) For each configuration of the tuning parameters, calculate the cv.error
Step 3: train the model on the entire training set
Training Set
Note: use the configuration of the tuning parameters with the smallest cv.error
Step 4: estimate the model performance on the test set
Test Set

Experimental Results(data splitting and convergence test)

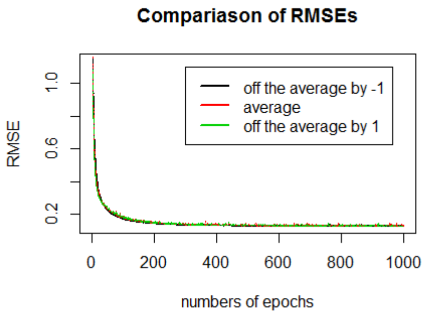
- Training set: $269,268$ observations(47.37% of the whole dataset)
- Test set: $165,880$ observations(29.20% of the whole dataset)
- Based on the test RMSE of the baseline model ($\mathcal{M}_1$), the baseline score for this study is 1.041752

Experimental Results(Convergence test)



- using the fixed initialization: the RMSE was still decreasing after 1000th epochs. The RMSE at the 1000th epoch was 0.169536179.
- when initializing with a random noise term, the RMSE reached a lower level using only 100 epochs and at the 1000th epoch, RMSE was only 0.1298694.

Experimental Results(Sensitivity Analysis of the SGD Algorithm in Collaborative Filtering)



In conclusion, both the convergence assessment and sensitivity analysis on different starting values provide support for the robustness of the SGD algorithm. Also, different starting values do not play a central role in convergence of the algorithm.

Experimental Results(step size and starting points)

According to the evidence from these experimental results, the step size and starting points used in our later experiments are chosen to be

- $h = 0.02$
- $[\mathbf{U}]_{ik}^{T} := \sqrt{\frac{\bar{R}}{d}} + \epsilon(r)$
- $[\mathbf{V}]_{kj} := \sqrt{\frac{\bar{R}}{d}} + \epsilon(r)$
- μ, $\{\alpha_i\}_{i=1}^{n_u}$, $\{\beta_j\}_{j=1}^{n_m}$ chosen to be initialized with random noise $\epsilon(r)$, where $\epsilon(r) \sim U[-0.1, 0.1]$.

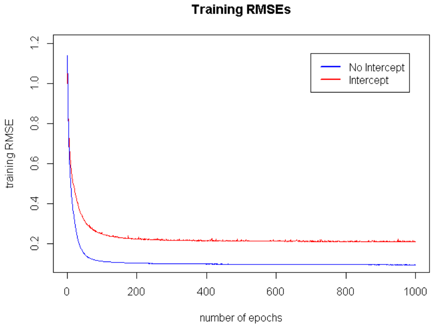
Experimental Results(Cross-Validation RMSE and Test RMSE for Each Model)

- 5-fold cross validation was used for each model $\mathcal{M}_2$ and $\mathcal{M}_3$, and the number of epochs used in the cross-validation was 300
- For $\mathcal{M}_2$, the best configuration was $(\lambda^*, d^*) = (0.05, 35)$ with a cross-validation RMSE of 0.906529
- For $\mathcal{M}_3$, the best configuration was $(\lambda^*, d^*) = (0.16, 30)$ with a cross-validation RMSE of 0.832056922.

Experimental Results(training on the whole training set)

Then we trained each model on the whole training set with 1000 epochs using (λ^*, d^*):

	training RMSE	test RMSE
$\mathcal{M}_2$	0.09294333	0.9023706
$\mathcal{M}_3$	0.2087268	0.8096657



Experimental Results(training on the whole training set continued)

- the error rate when predicting using model $\mathcal{M}_2$ is 13.38% better than the baseline score
- the error rate when predicting using model $\mathcal{M}_3$ is 22.28% better than the baseline score.
- Model $\mathcal{M}_3$ outperforms model $\mathcal{M}_2$ by only 8.9% for this particular dataset.

Market Segmentation through Clustering (K-means Clustering)

- What can we do with our trained models?
- One heuristic approach is to cluster users and items based on their extracted feature vectors
- K-means Clustering: given a set of n unlabeled observations $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, K-means clustering is a technique for partitioning the set into K disjoint clusters $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$, where $\mathcal{S}_k \cap \mathcal{S}_{k^*} = \emptyset$ if $k \neq k^*$, in a such way that minimizes the total within-cluster variation

$$\arg\min_{\mathcal{S}} \sum_{k=1}^{K} \sum_{\mathbf{x} \in \mathcal{S}_k} ||\mathbf{x} - \bar{\mathbf{x}}_k||^2$$

where $\bar{\mathbf{x}}_k$ is the cluster center, i.e the mean of the observations in $\mathcal{S}_k$.

Market Segmentation through Clustering (K-means Clustering: Elbow Method)

The elbow method determines the optimal number of clusters by performing K-means clustering technique on a range of values of K, and for each K compute the within cluster sum of squares:

$$WCSS(K) = \sum_{j=1}^{K} \sum_{v_i \in cluster_j} ||v_i - \bar{v}_j||^2 \tag{10}$$

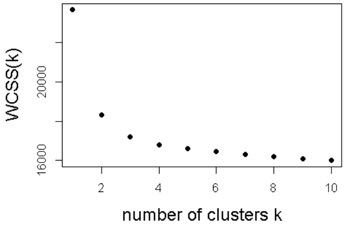
Then, one should choose a K such that adding an extra cluster does not bring much gain in variance explained by the additional cluster.

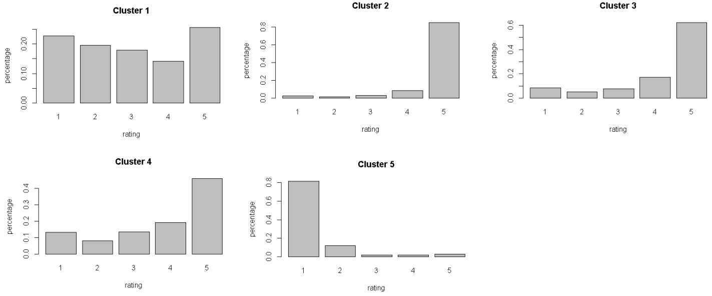
Text Mining on Each of the Clusters

Text mining is performed on each of the text reviews

- Text preprocessing: removing punctuation, converting each word to lower case, removing numbers, removing common word endings such as -ing, -es, -s, and removing stop words, such as are, the, and that, that have no analytic value.
- Each text review is transformed into a text frequency vector

Experimental Results(Clustering items using $\mathcal{M}_2$)



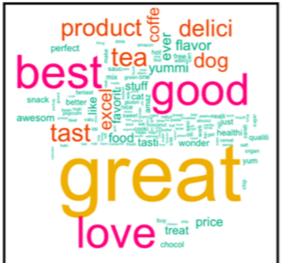


Experimental Results(Clustering items using $\mathcal{M}_2$ continued)

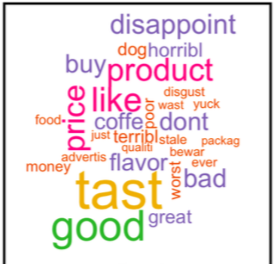
Cluster 2 has the highest observed average rating 4.714273 while cluster 5 has the lowest 1.324111.

Table 4.6: Most Frequent Terms in Cluster 2 and 5

2	great	best	good	love	tea	tast	product	delici	dog
5	tast	good	like	product	price	disappoint	bad	buy	dont

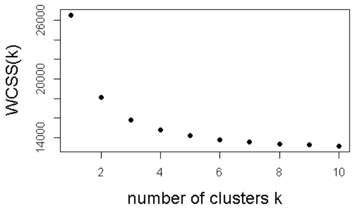


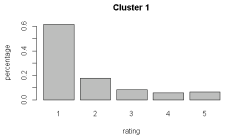
Cluster 2

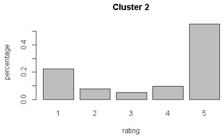


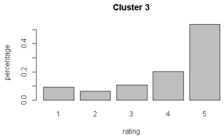
Cluster 5

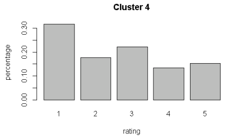
Experimental Results(Clustering items using $\mathcal{M}_3$)

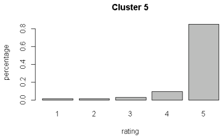










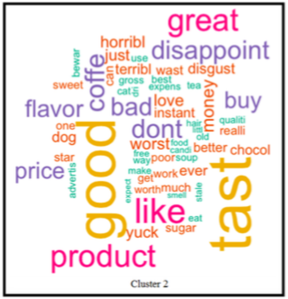


Experimental Results(Clustering items using $\mathcal{M}_3$ continued)

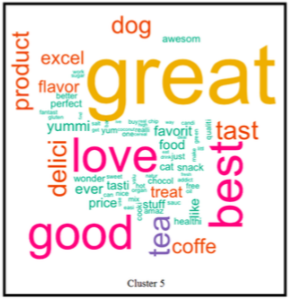
Cluster 1 has the lowest average rating of 1.779356 while cluster 5 has the highest 4.744958

Table 4.8: Most Frequent Terms in Cluster 1 and 5

1	tast	good	like	product	great	coffe	dont	disappoint	bad
5	great	good	love	best	tea	tast	delici	dog	product



Cluster 2

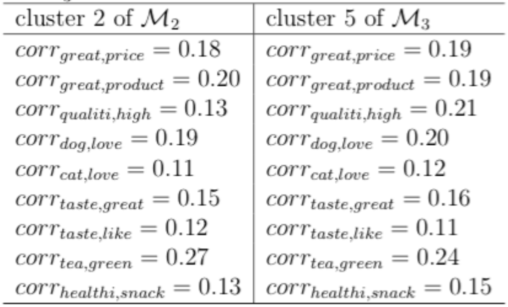


Cluster 5

Experimental Results(Comparison of the clustering results of two models)

No matter which model, negative words like *disappoint* and *bad* are more likely to occur in the cluster with lowest average rating while positive words like *good*, *great*, *best* more frequently appear in the cluster with highest average rating

Table 4.9: Correlations of the Terms in the Cluster with the Highest Average Rating from Model $\mathcal{M}_2$ and $\mathcal{M}_3$



cluster 2 of $\mathcal{M}_2$	cluster 5 of $\mathcal{M}_3$
$corr_{great,price} = 0.18$	$corr_{great,price} = 0.19$
$corr_{great,product} = 0.20$	$corr_{great,product} = 0.19$
$corr_{qualiti,high} = 0.13$	$corr_{qualiti,high} = 0.21$
$corr_{dog,love} = 0.19$	$corr_{dog,love} = 0.20$
$corr_{cat,love} = 0.11$	$corr_{cat,love} = 0.12$
$corr_{taste,great} = 0.15$	$corr_{taste,great} = 0.16$
$corr_{taste,like} = 0.12$	$corr_{taste,like} = 0.11$
$corr_{tea,green} = 0.27$	$corr_{tea,green} = 0.24$
$corr_{healthi,snack} = 0.13$	$corr_{healthi,snack} = 0.15$

Experimental Results(Comparison of the clustering results of two models continued)

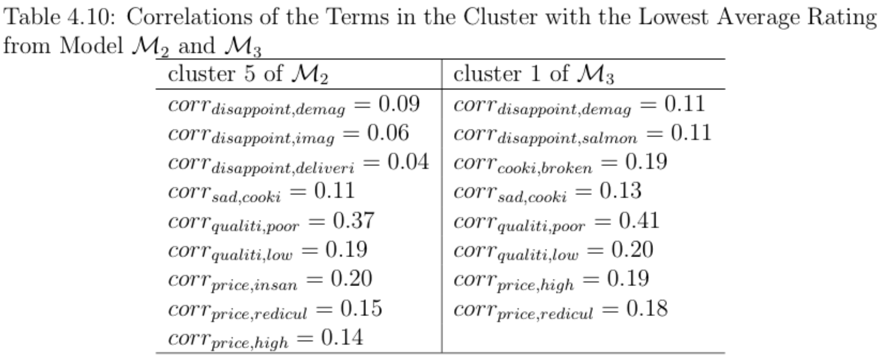


Table 4.10: Correlations of the Terms in the Cluster with the Lowest Average Rating from Model $\mathcal{M}_2$ and $\mathcal{M}_3$

cluster 5 of $\mathcal{M}_2$	cluster 1 of $\mathcal{M}_3$
$corr_{disappoint,demag} = 0.09$	$corr_{disappoint,demag} = 0.11$
$corr_{disappoint,imag} = 0.06$	$corr_{disappoint,salmon} = 0.11$
$corr_{disappoint,deliveri} = 0.04$	$corr_{cooki,broken} = 0.19$
$corr_{sad,cooki} = 0.11$	$corr_{sad,cooki} = 0.13$
$corr_{qualiti,poor} = 0.37$	$corr_{qualiti,poor} = 0.41$
$corr_{qualiti,low} = 0.19$	$corr_{qualiti,low} = 0.20$
$corr_{price,insan} = 0.20$	$corr_{price,high} = 0.19$
$corr_{price,redicul} = 0.15$	$corr_{price,redicul} = 0.18$
$corr_{price,high} = 0.14$	

Discussion

- The model performance for both $\mathcal{M}_2$ and $\mathcal{M}_3$ are pretty good while $\mathcal{M}_3$ slightly outperforms $\mathcal{M}_2$
- The predictions of both latent factor models could potentially be improved by more careful tuning of step size, λ and d
- Using the extracted hidden item features by either model to cluster items gave similar results
- Combining latent factor models with such additional source of information could improve the accuracy of predicting user ratings

The End