Recommendation Systems

Tool: Clustering, Collaborative and content filtering

The Analytics Edge:

Recommendation systems build models around users preferences to help personalize the user experience. Examples include Netflix that provides recommendation on movies to users, Amazon that provides recommendation of items of almost any type to users, eHarmony that provides recommendations on compatible matches for single men and women. The large number of choices available, the massive amount of data available through online transactions help companies use tools of analytics such as clustering and collaborative and content filtering to find the "right" items to recommend to users.

The challenges in recommendation systems pertain to obtaining real-time recommendations, making good recommendations since poor recommendations can imply frustrated customers, and dealing with users for which very little information is available to users with high activity.

The Netflix Story

Netflix is a company that provides DVD by mail service and on-demand streaming of films and TV seriés. The company was founded in 1997 and has over 60 million subscribers globally with annual revenues of roughly $5.5 billion in 2014. Netflix is now available in Singapore too. The company Started with Hastings (one of the Co-founders of Netflix) who was forced to pay $40 in fines when he returned the Apollo 13 DVD well past its due date. As of April 201 Netflix has 81 million subscribers worldwide.

Netflix provides a monthly flat ~~fee~~ service rate for rental of DVDs. A subscriber creates an ordered list of films to rent. Films are delivered by US mail to subscribers who can keep the rented discs as long as they desire with a limit on how many discs that a subscriber can have on loan simultaneously. To rent a new film, the subscriber must mail back the current DVD upon the receipt of which Netflix will ship the next disc in the rental queue. (Flat rate DVD by mail service)

Now increasingly, Netflix is moving more towards the video on demand streaming services on the Internet.

Netflix prize

In 2006, Netflix held the Netflix prize competition to predict the user preferences and beat its existing Netflix movie recommendation engine known as Cinematch by atleast 10%.

Netflix provided a training data set of 100 480 507 ratings that 480 189 users gave
(over 100 million)
to 17770 movies. Each movie was rated from 1 to 5 by the users expressing how much they liked or disliked it.

($\approx$ 100 million ratings
$\approx$ 480 thousand users
$\approx$ 18 thousand movies)

The data was collected from 1998 to 2005. In addition it withheld the ratings of 2817131 ($\approx$ 3 million ratings) data points from the same subscribers over the same set of movies as a "qualifying" dataset. Any participating team had to predict the ratings on the entire qualifying data set but were informed of the accuracy only for roughly half the data (a "quiz" set of 1408342 ratings). This was used to calculate the leaderboard. The other half (a "test" set of 1408789) of ratings was used to determine the ultimate winners. Only the judges knew the quiz & test set partition in the qualifying dataset.

Note that in the training dataset, the average user rated over 200 movies and the average movie was rated by over 5000 users. However some movies had only 3 ratings in the training set while one user had rated over 17000 movies (wide variance in the dataset)

The predictions (which could be any number,
not necessarily in the set {1, 2, 3, 4, 5}) were
Scored against the true ratings in terms
of RMSE (root mean squared error). The
goal was to minimize the Root mean squared
error. The trivial algorithm that used the
average rating for each movie from the traning
set produced a RMSE of 1.0540 on the
quiz set. The algorithm used by Netflix
at that time called Cinematch scored an RMSE
of 0.9514 on the quiz data using the
traning set to build the model (roughly
10% improvement). On the test set, Cinematch
had a RMSE of 0.9525 (close to quiz dataset
To win the 1 million $ prize, the team had
to beat Cinematch by another 10%, to
achieve 0.8572 on the test set. Once
the RMSE on the quiz set was 0.8572 or lesser,
30 days was provided to submit additional candidate
predictions. As long as no team won the grand prize
$ 50,000 was awarded as long as for each year they
improved on the previous winner by atleast 1%..
Teams could submit one attempt per day.

Netflix prize

Oct 2, 2006	Competition launched
Oct 8, 2006	WXYZ Consulting beats Cinematc
June 2007	Over 20000 teams registered
Nov 13, 2007	Bell Kor (Team of 3 AT&T lab researchers) won 50000$ Progres Prize with 8.43% improvement over Cinematch
2008	Teams Bell Kor & BigChaos joine together to win 5000$ prize wi 9.4 % improvement
June 26, 2009	Team BellKor Pragmatic Chaos achieved 10.05 % improvement (quiz test RMSE of 0.8558) Netflix entered last call for Grand Prize.
Last Call (30 days)	
July 25, 2009	Team Ensemble achieved 10.09%
July 26, 2009	Netflix stopped gathering submissions
Final standing on leaderboard	Ensemble 10.10% improvement BellKor Pragmatic Chaos 10.09% quiz set results RMSE = 0.8554
Sep 18, 2009	On the test set, both teams had a test RMSE of 0.8567. But BellKor submitted it 20 min earlier making them the winners of the 1M$ Netflix prize

The team Bell Kor's Pragmatic Chaos consisted of two researchers from AT & T Labs, two researchers from Austrian at the Commendo Research & Consulting, one researcher from Yahoo and two researchers from Pragmatic Theory.

The team published a description of the algorithm as required by the competition

A variety of methods was used in the final winning predictive algorithm. Successful analytics often needs a combination of a variety of approaches to be successful. These approaches include collaborative filtering, matrix factorization, regression models, LASSO technique all combined into ensemble methods.

Movie Lens

Movie Lens is a recommendation system for movie set up by Grouplens research, a research lab at the University of Minnesota.

Grouplens collects and makes data on movie ratings from Movie Lens available for research.

Clustering

In supervised learning, typically one has access to a set of p features (or predictor variables) $X_1, \ldots, X_p$ and an output (or response variable) Y. The goal is to predict Y from $X_1, \ldots, X_p$. We have seen several examples of this including linear regression, logistic regression, discrete choice models, classification and regression trees, random forests.

In unsupervised learning, one has access to a set of p features $X_1, \ldots, X_p$ but no associated response variable. Clustering is an example of such an approach that is used to discover subgroups within the data.

One of the main challenges with unsupervised learning is that it is far more subjective since there is no clear way to perform cross-validation or validate results on a test set.

However these techniques are important in several domains such as clustering groups of shoppers based on their browsing and purchasing histories on say Amazon. Then an individual shopper can be preferentially shown items based on the purchase histories of other shoppers in the cluster (Build a model for each cluster)

Cluster observations into distinct groups such that.

1) Data within a group are similar to each other

2) Data in different groups are different from each other

K-means clustering

given n observations $\bar{x}_i = (x_{i1}, \ldots, x_{ip})$ for $i = 1, \ldots, n$, partition it into K clusters $C_1, \ldots, C_K$ where:

$$C_1 \cup C_2 \cup \ldots \cup C_K = \{1, \ldots, n\}$$

$$C_k \cap C_{k'} = \phi \quad \text{for } k \neq k'$$

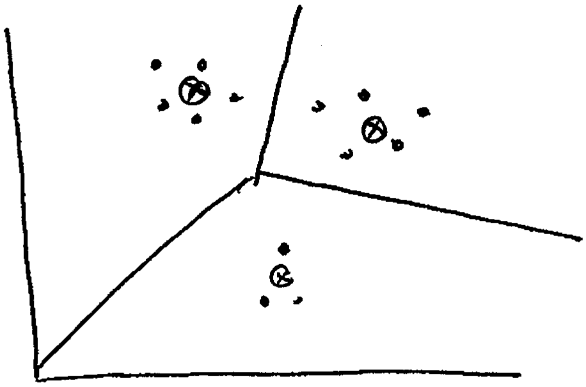
The clusters are chosen so as to minimize the within cluster sum of squares:

$$\min_{\substack{C_1, \ldots, C_K \\ \mu_1, \ldots, \mu_K}} \sum_{k=1}^{K} \sum_{i \in C_k} \| \bar{x}_i - \bar{\mu}_k \|^2$$

(where $\|x_i - \mu_k\| = \sqrt{(x_{i1} - \mu_{k1})^2 + \ldots + (x_{ip} - \mu_{kp})^2}$)

Here the optimal $\mu_1, \ldots, \mu_K$ are the mean of the observations in clusters $C_1, \ldots, C_K$, namely $\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$. The observations are partitioned into clusters in which each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster.

K = 3



Partition the Space of data into Voronoi cells

(Region where all the points in that cell are closer to the ⊗ than any other)

Given K clusters and n observations there are K^n ways to partition which grows rapidly as n increases. Solving the K-means problem exactly is challenging but efficient heuristic algorithms that quickly converge to the local optimum exist.

<u>Algorithm</u> (Lloyd's algorithm)

1) Start with a random set of K means. For example this could be K observations from the data set

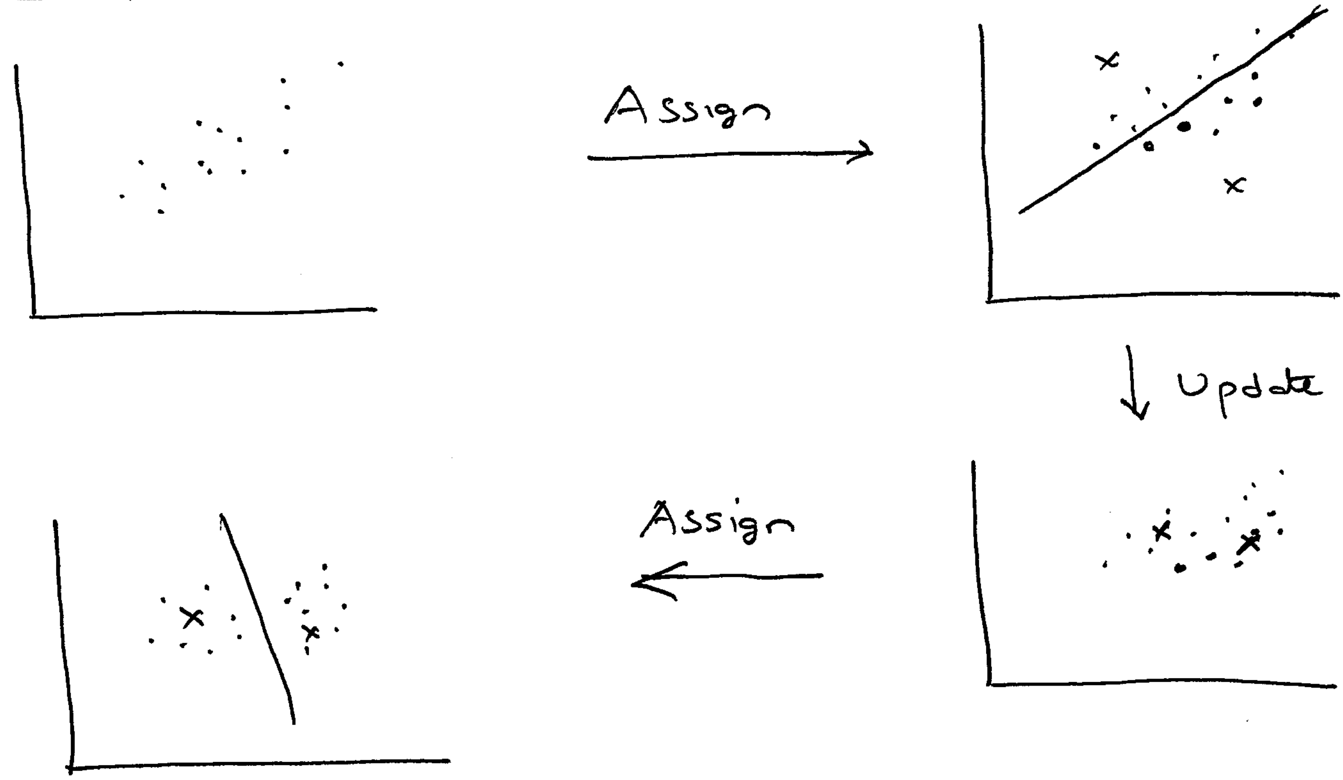
2)

a) <u>Assignment step</u>: Assign each observation to the cluster whose centroid (mean) is the closest.

b) <u>Update step</u>: Calculate the new centroid (mean) of the observations in the new cluster

Repeat steps a) and b) till the assignment no longer changes.

Example : $K = 2$



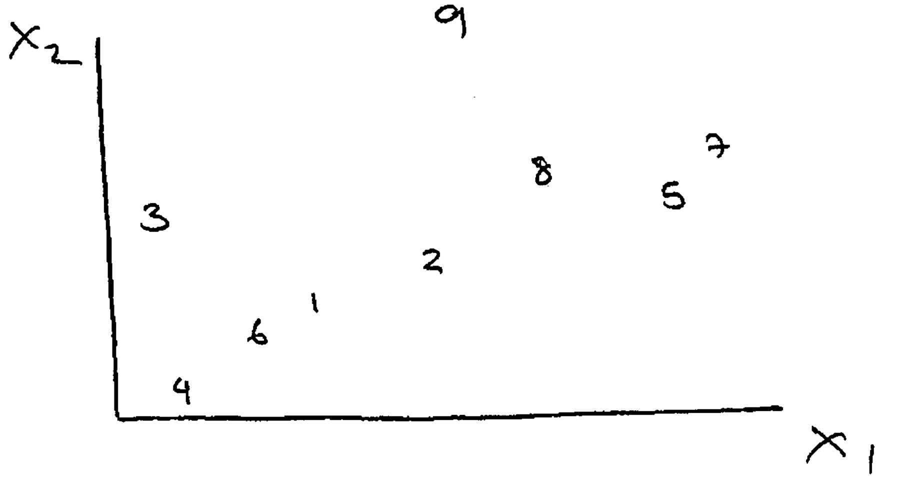
Continue till convergence to local optimum

Alternate approach is start by randomly allocating each observation to a cluster and then using the update and assignment steps iteratively. The number of clusters (K) has to be decided before running the algorithm. The K-means algorithm works well for both small & large datasets. Since randomization is involved one can also try running the algorithm multiple times to try & choose the best clusters.

One however needs to check if the clusters are reasonable - looking at statistic of each cluster or looking at an outcome variable which is not used to cluster but clustering can help prediction then

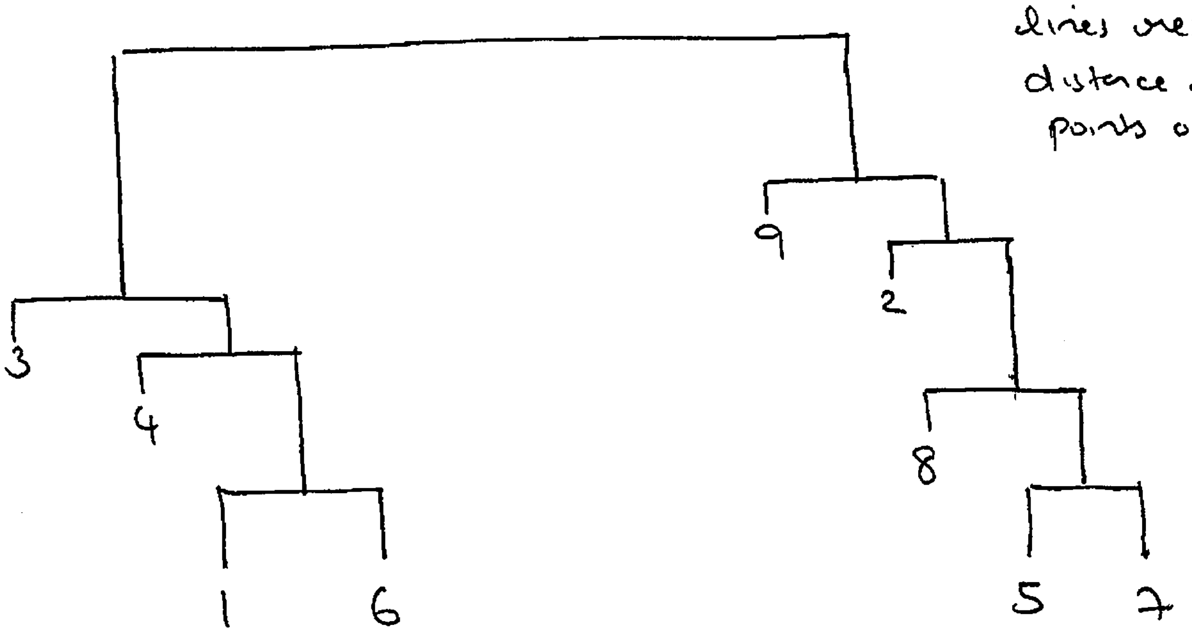
Hierarchical Clustering

This method does not need K to be prespecified and provides a tree based representation of the observations (called a dendrogram).



$n = 9$ observations ; $p = 2$

Dendrogram representation



Heights of vertical lines represents distance between points or clusters

Each leaf of dendrogram is an observation.
Fusion of two leaves into a branch corresponds to similar observations.
Observation that fuse at bottom of tree are very similar while fuse at top tends to be quite different
Height of fusion indicates how different observations are.

Note that similarity of observations should be based on the location on the vertical axis where branches containing two observations are fused first.
Similarity of two observations cannot be based on their proximity along horizontal axis

For example 9 is not closer to 2 any more than it is to 8, 5 or 7.

To identify clusters from a dendrogram, make a horizontal cut across the dendrogram. The number of vertical lines that it crosses will be the number of clusters.
The farthest this horizontal line can move up and down without hitting one of the horizontal lines the better. Also depending on the application, you might have some sense on how many clusters you want.

The term hierarchical refers to the observation that the clusters obtained by cutting the dendrogram at a particular height is nested within clusters obtained by cutting it at a greater height.

Algorithm (Hierarchical clustering)

1) Begin with n observations and a measure of pairwise dissimilarities (a total of $\frac{n(n-1)}{2}$ measures). Treat each of the observation as its own cluster.

2) For $i = n, n-1, \ldots, 2$

   a) Examine all pairwise inter cluster dissimilarities among the i clusters and identify pairs of clusters that are most similar. Fuse these clusters. The dissimilarity between clusters helps indic height of dendrogram where fusion should be placed.

   b) Compute pairwise inter cluster dissimilarit among $i-1$ remaining clusters.

Note that measuring distances (dissimilarities) between clusters has severaly choices. Say a Euclidean distance metric is used. Then one can define the linkage (dissimilarity between two groups of observations) for example as Complete (Maximal distance between two points in clusters A and B), Average (Average of distance between any two observations, one in A and the other in B)

Ward's criterion is another such method that minimizes the total within cluster variance. Here the pair of clusters that are merged such that it leads to minimum increase in total within cluster variance after merging.

$$\text{Within cluster variance} = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{d=1}^{p} (x_{id} - x_{i'd})^2$$

$$= 2 \sum_{i \in C_k} \sum_{d=1}^{p} (x_{ij} - \mu_{kj})^2$$

$$\text{where } \mu_{kj} = \frac{\sum_{i \in C_k} x_{ij}}{|C_k|}$$

<u>Note</u>:

$$\left(\text{where } \mu = \frac{\sum_i x_i}{n}\right)$$

$$\sum_i \|x_i - y\|^2 = \sum_i \|x_i - \mu + \mu - y\|^2$$

$$= \sum_i \|x_i - \mu\|^2 + \sum_i \|\mu - y\|^2 + 2 \sum_i (x_i - \mu)'(\mu - y)$$

$$= \sum_i \|x_i - \mu\|^2 + n \|\mu - y\|^2$$

Hence

$$\sum_{i,i'} \|x_i - x_{i'}\|^2 = n \sum_i \|x_i - \mu\|^2 + n \sum_j \|\mu - x_j\|^2$$

$$= 2n \sum_i \|x_i - \mu\|^2$$

k-means clustering typically runs much faster than ~~hierarchical~~ hierarchical clustering

(linear versus quadratic computational time in number of observations)

k means will provide different results in different runs due to randomization while hclust will give repeatable solutions

Analytics on Movie Lens dataset

```
movies <- read.csv("movies.csv", stringsAsFactors = FALSE)
```

The movies dataframe consists of two variables with 8569 movies with movie ID and title

To read in movie genres, we use the following commands to check we are reading it in properly.

```
Countfields <- count.fields("genres.csv", sep = "|")
```

This helps to count the number of fields as seperated by "|" in each row of genres.csv

```
Countfields[1]
```

This gives value of 5 which means first movie has 5 genres listed

```
min(countfields)
max(countfields)
```

Each movie has between 1 and 7 genres

```
genres <- read.csv("genres.csv", header = FALSE,
    sep = "|", col.names = c("X1", "X2", "X3",
    "X4", "X5", "X6", "X7"))
```

This reads in a data frame with 8569 observations & 7 variables with column names "X1" to "X7".

header = FALSE helps validate variable names are not provided in top row.

Note that each variable has different number of factor levels. To obtain the overall set, we use:

```
fac <- union(union(union(union(union(union(
        levels(g$X1), levels(g$X2)), levels(g$X3)), levels(g$X4)),
        levels(g$X5)), levels(g$X6)), levels(g$X7))
```

fac has a total of 20 categories from Action to IMAX where " " simply illustrates missing category. (or not provided)

To standardize across all variables:

```
g$X1 <- factor(g$X1, fac)
    ⋮
g$X7 <- factor(g$X7, fac)

levels(g$X1)
```

"Action" "Adventure" "Animation" "Children"

"Comedy" "Crime" "Documentary" "Drama"

"Fantasy" "Film-Noir" "Horror" "Musical"

"Mystery" "Romance" "Sci-Fi" "Thriller"

"War" "Western" " " "IMAX"

```
M <- matrix(0, nrow = 8569, ncol = 20)
```

Creates a matrix with 8569 rows (movies) and 20 columns (categories)

```
colnames(M) <- fac
```

Assign columnnames to matrix as genres

```
for(i in 1:8569) {
  M[i, genres[i,"X1"]] <- 1
   :
  M[i, genres[i,"X2"]] <- 1 }
```

Creates a matrix with entry = 1 where a movie is of a particular genre & 0 otherwise.

```
Data <- as.data.frame(M)
Data$title <- movies$title
```

Creates a data frame & introduces the movie title into the data frame.

```
Data <- Data[,-19]
```

Drops the 19 th column which corresponds to "" category.

Hierarchical clustering in R

```
distances <- dist(Data[,1:19], method = "euclidean")
```

The function dist(.) computes distances between movies using the first 19 columns (genres of movies) with Euclidean distances.

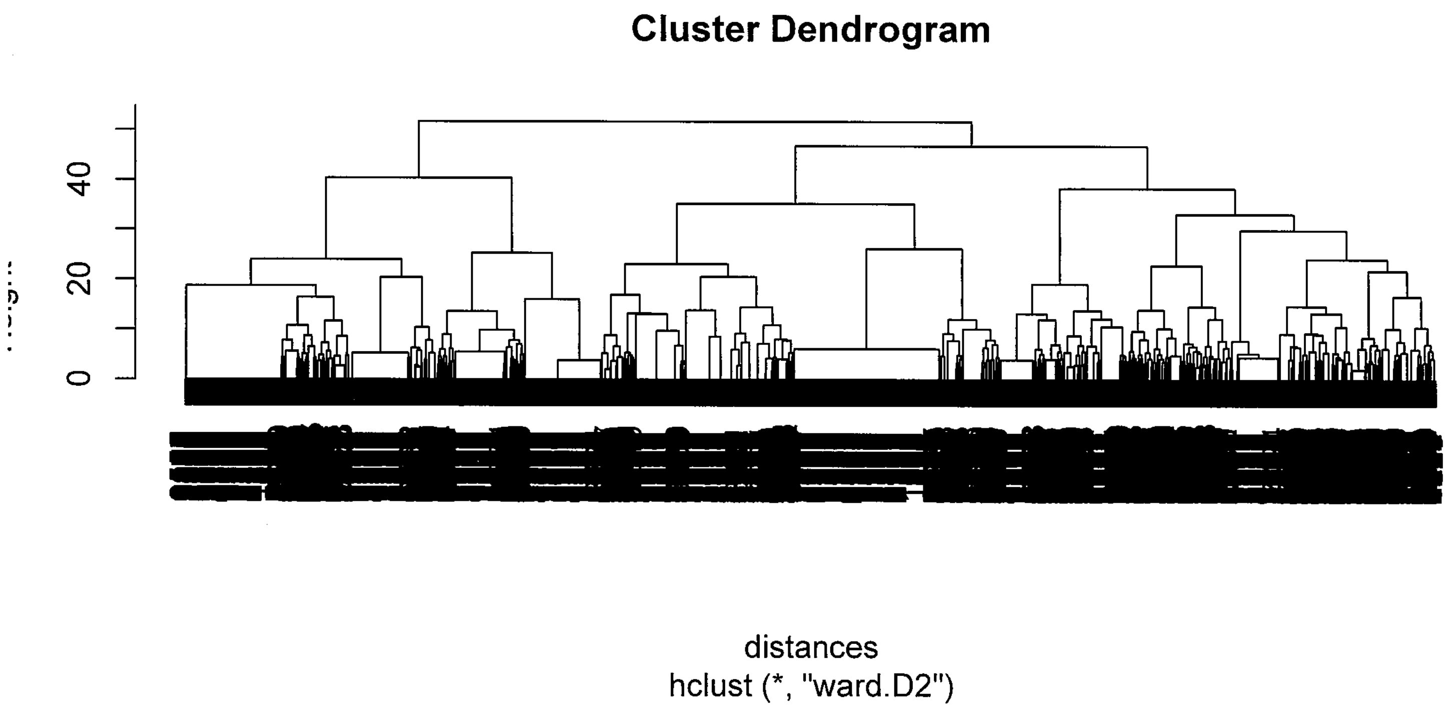
A total of $\frac{8568 \times 8569}{2} = 36709596$ distances are computed.

```
clusterMovies1 <- hclust(distances, method = "ward.D2")
```

Performs hierarchical cluster analysis using the distances. The Ward's method aims to find compact, spherical clusters.

```
plot(clusterMovies1)
```

Plots a dendogram of clusters. This lists all points at the bottom making it hard to read in this example.


Cluster Dendrogram
40
20
0
distances
hclust (*, "ward.D2")

```
clusterGroups1 <- cutree(clusterMovies1, k = 10)
```

Cuts a tree resulting from the hclust(.) into 10 groups.

```
tapply(Data[,1], clusterGroups1, mean)
```

Computes the average value across the cluster groups for the Action variable. Higher value indicates many movies in the cluster are action movies

```
Cat1 <- matrix(0, nrow = 19, ncol = 10)
for (i in 1:19) {
  Cat1[i,] <- tapply(Data[,i], clusterGroups1,
                     mean) }

rownames(Cat1) <- colnames(Data)[1:19]
```

This helps create a matrix Cat where rows denote categories and columns indicate clusters

Cluster 1 : Fantasy, Comedy, Children, Adventure

Cluster 2 : Romance, Comedy, Drama

Cluster 3 : Comedy, Drama

Cluster 4 : Drama, Thriller, Crime

Cluster 5 : Sci-Fi, Adventure, Action

Cluster 6 : Horror, Thriller

Cluster 7 : Drama

Cluster 8 : Romance, Drama

Cluster 9 : Documentary

Cluster 10 : War, Drama, Action

Subset (Data $ title, cluster Groups1 = = 6)

lists out all movies in category 6

(horror, thriller)

Subset (Data, movies $ title == "Grand Budapest Hotel, The (2014)"

This is row number 8418 in the data frame

Cluster Groups1[8418]

This is the comedy, drama cluster (3)

Subset (Data, movies $ title == "Moneyball (2011)")

This is row number 7925 in the data frame

Cluster Groups1[7925]

This is in the drama cluster (7)

Subset (Data, movies $ title == "X-Men : Frst Class (2011)"

Row number 7849

Cluster Groups1[7849]

This is put in cluster (10).

k-means clustering in R

set.seed(1)

ClusterMovies2 <- kmeans(Data[,1:19],
centers = 10, nstart = 20)

This performs a k-means clustering using k = 10 and 20 random initial configurations. Here a random set of rows are chosen as initial centers.

ClusterMovies2$tot.withinss

Total within cluster sum of squares (we want this number to be small)

set.seed(1)

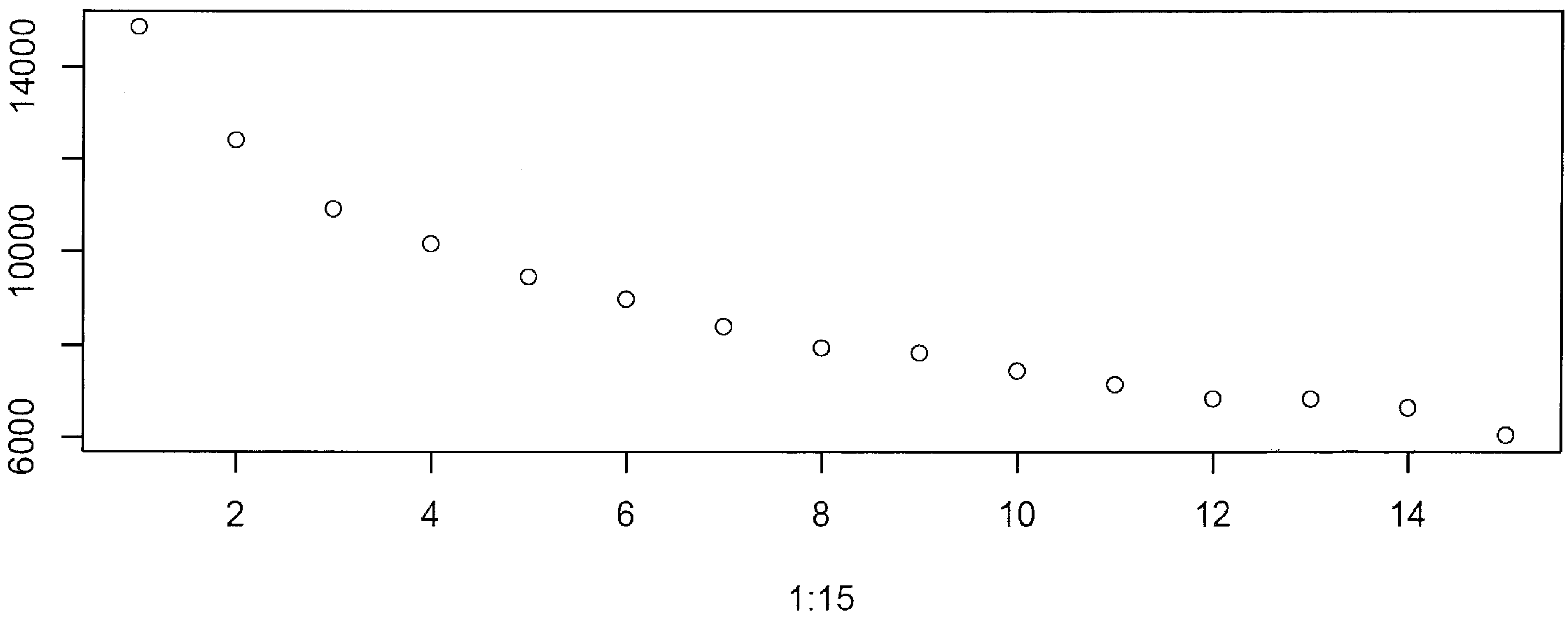
ClusterMovies3 <- kmeans(Data[,1:19], centers = 10,
nstart = 1)

ClusterMovies3$tot.withinss

Comparing 7601.8 with 7324.7 it is clear that picking more starting points help provides better solutions

```
set.seed(1)
fit <- 0

for (k in 1:15) {

ClusterMovies4 <- kmeans(Data[,1:19], Centers = k,
                         nstart = 20)

fit[k] <- clusterMovies4$tot.withinss }

plot(1:15, fit)
```

This plots the error in the cluster fit as the number of clusters increases


14000
10000
6000
2
4
6
8
10
12
14
1:15

```
Cat2 <- matrix(0, nrow= 19, ncol = 10)
for (i in 1:19) {
    Cat2[i,] <- tapply(Date[,i], clusterMovies2$cluster,
                       mean)
}
rownames(Cat2) <- colnames(Date)[1:19]
```

Cluster 1 : Horror

Cluster 2 : Crime, Drama, Thriller, Action

Cluster 3 : Drama

Cluster 4 : Documentary

Cluster 5 : Action, Adventure, Sci-Fi, Thriller

Cluster 6 : Comedy

Cluster 7 : Thriller, Horror

Cluster 8 : Children, Adventure, Animation, Fantasy, Comedy

Cluster 9 : War, Drama

Cluster 10 : Comedy, Drama

Recommendation Systems

There are two main types of recommendation systems:

1) Collaborative filtering : Recommendations are made based on attributes of users.

Each user is represented by a vector of items where the entry gives the customer's rating of the item. This vector will typically have many empty entries (only a small fraction is rated or purchased)

Last.fm creates a list of recommended songs for you by observing what the user listens to and likes and comparing against other users

2) Content filtering : Recommendations are made based on attributes of items.

Each item is ~~rep~~ represented by a set of attributes (such as genre of movie, keywords on webpage).

Pandora uses the attributes of a song such as style, artist to seed the station with other songs with similar attributes .

3) Hybrid recommendation Systems :

A combination of both collaborative and Content filtering might be useful. Netflix is such an example. For example Netflix makes recommendations by comparing the watching and searching habits of similar users (collaborative filtering) as well as recommending movies that share similar attributes (content filtering).

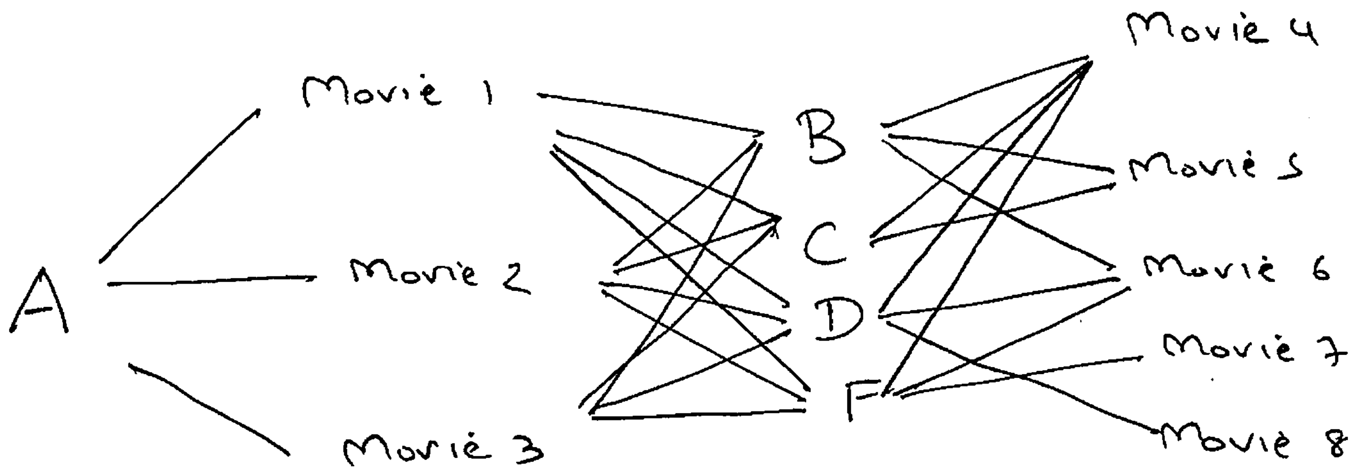
Content filtering often works better than Collaborative filtering when the user has not rated or purchased many items. As soon as one item is liked, content filtering can be used to make recommendations.

However if a user has rated many items, it is hard for content filtering to make recommendations since there might be many similar items to liked items. Furthermore in collaborative filtering it is possible to recommend items not previously considered in user's history. Thus it is possible to get them to purchase items they might not consider

Collaborative filtering

In this approach the data of ratings of users for items is used to predict missing ratings Our create a top-N recommendation list for an active user.

Note that collaborative filtering will suffer from a cold start problem in comparison to content filtering since it will be unable to address new products and users. On the other hand it is domain free (does not depend on the item attributes).



Say person A likes movies 1, 2 & 3

Suppose persons B, C, D & F also liked movies 1, 2 & 3

All of them also liked movie 4 followed by movie 6 which can be recommend to A in order

Techniques for collaborative filtering

Let $r_{u,i}$ = Rating of user u for item i

Baseline model

A simple baseline model is to predict the average rating based on item popularity average:

$$\underbrace{b_{u,i}}_{\substack{\text{Baseline prediction} \\ \text{for user } u \text{ for item } i}} = \bar{r}_i \quad \left(\begin{array}{c}\text{Average rating} \\ \text{for item } i\end{array}\right)$$

across all users who rated it

Another approach is based on random recommendations or item popularity (no. of users who have rated it)

User based collaborative filtering

This is based on identifying other users whose ratings are similar to that of the active user and use their rating on other items to predict what the active (current) user will like.

To measure the similarity of user to user, the Pearson correlation can be used

$$\underbrace{S(u,v)}_{\substack{\text{Similarity between} \\ \text{users } u \text{ and } v}} = \frac{\sum_{i \in I_u \cap I_v} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{ui} - \bar{r}_u)^2 \sum_{i \in I_u \cap I_v} (r_{vi} - \bar{r}_v)^2}}$$

Here I_u, I_v are items rated by users u and v and $\bar{r}_u$, $\bar{r}_v$ are average ratings of users u and v.

This similarity metric was used in Netflix.

Alternative similarity measures include the cosine similarity:

$$S(u,v) = \frac{\sum_{i \in I_u \cap I_v} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u \cap I_v} r_{ui}^2} \sqrt{\sum_{i \in I_u \cap I_v} r_{vi}^2}}$$

To predict the rating the simplest method is to choose a set of neighbors of user u, denoted by N_u (say k nearest neighbors or atleast a certain level of similarity):

$$\underbrace{P_{ui}}_{\substack{\text{Predicted rating} \\ \text{for user } u \text{ for item } i}} = \frac{\sum_{v \in N_u} r_{vi}}{|N_u|}$$

We can also use the observation that some users are more similar to u using the similarity metric.

$$P_{ui} = \frac{\sum_{v \in N_u} S(u,v)\, r_{vi}}{\sum_{v \in N_u} S(u,v)}$$

While all users could be used in the set N_u, it helps to restrict to a smaller number of neighbors (in the range of 20 to 250 typically)

Item based collaborative filtering

This is based on identifying items for users that are similar to the items they like. If two items have similar rating patterns across users the items are similar. Users will prefer items that are similar where the similarity metric is measured as say:

$$S(i,j) = \frac{\sum_{u \in U_i \cap U_j} r_{ui} r_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} r_{ui}^2} \sqrt{\sum_{u \in U_i \cap U_j} r_{ui}^2}}$$

To predict the rating, use $S(i,j)$ to identify a neighborhood N_i of items for item i:

$$P_{ui} = \frac{\sum_{j \in N_u} S_{ij} r_{uj}}{\sum_{j \in N_u} S_{ij}}$$

The performance of recommender algorithms can be improved by normalizing the ratings so as to compensate for user's differences on rating scales. For example, there could be bias caused by users who consistently rate higher than (or lower than) other users.

Let $\hat{r}_{ui} = r_{ui} - \bar{r}_u$. This centers the scores. After applying collaborative filtering and normalizing back to original scale gives:

$$\underbrace{p_{ui}}_{\text{Predicted rating for user } u \text{ for item } i} = \bar{r}_u + \frac{\sum_{v \in N_u} (s_{uv})(r_{vi} - \bar{r}_v)}{\sum_{v \in N_u} (s_{uv})}$$

Other transformations include taking the rating variance into account.

To measure the quality of predicted ratings

$$\underset{\text{(root mean squared error)}}{\text{RMSE}} = \sqrt{\frac{\sum_{u,i} (p_{ui} - r_{ui})^2}{n}}$$

where p_{ui} = predicted rating for user u for item i, r_{ui} is real rating and n is the total number of ratings that are predicted.

Analytics on MovieLens data set

```
ratings <- read.csv("ratings.csv")
str(ratings)
```

The ratings dataframe consists of 100023 Observations of 3 variables where UserId (Identity of user), movieId (Identity of movie), rating (0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)

706 users, 8552 movies

```
max(table(ratings$userId))
min(table(ratings$userId))
```

Users have rated from 20 movies up to 2268 movies (User 16, User 516 17,.., 684

```
which(table(ratings$userId) == 20)
max(table(ratings$movieId))
min(table(ratings$movieId))
```

Movies have from 1 up to 337 ratings

```
Data <- matrix (nrow = length(unique (ratings $ userId))
                ncol = length(unique (ratings $ movie Id))
```

This creates an empty matrix with 706 rows and 8552 columns

```
rownames (Data) <- unique (ratings $ user Id )
colnames (Data) <- unique (ratings $ movie Id)
```

This names the rows and columns with the unique users and movies in the data set

```
for (i in 1: nrow (ratings)) {
Data [as. character (ratings $ user Id [i]),
      as. character (ratings $ movie Id [i]) ] <- ratings $
                                                    rating [i]

}
```

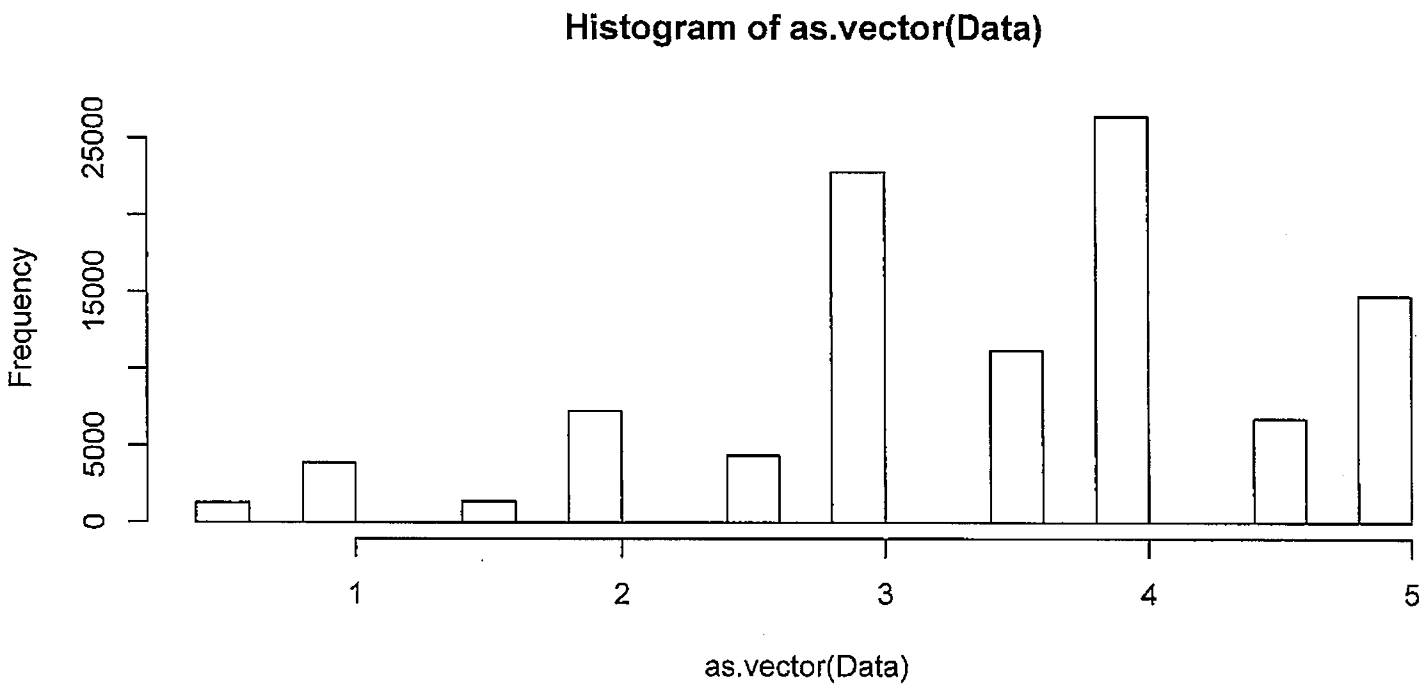
This fills in the entries of the Data matrix with movie ratings

```
Data norm <- Data - rowMeans (Data, na.rm = TRUE)
```

Normalize the ratings of each user so that bias is reduced.

```
hist (as. vector (Data ))
hist (as. vector (Data norm))
```

Histograms of raw & normalised ratings indicate greater skew in raw ratings & more symmetric in normalised ratings

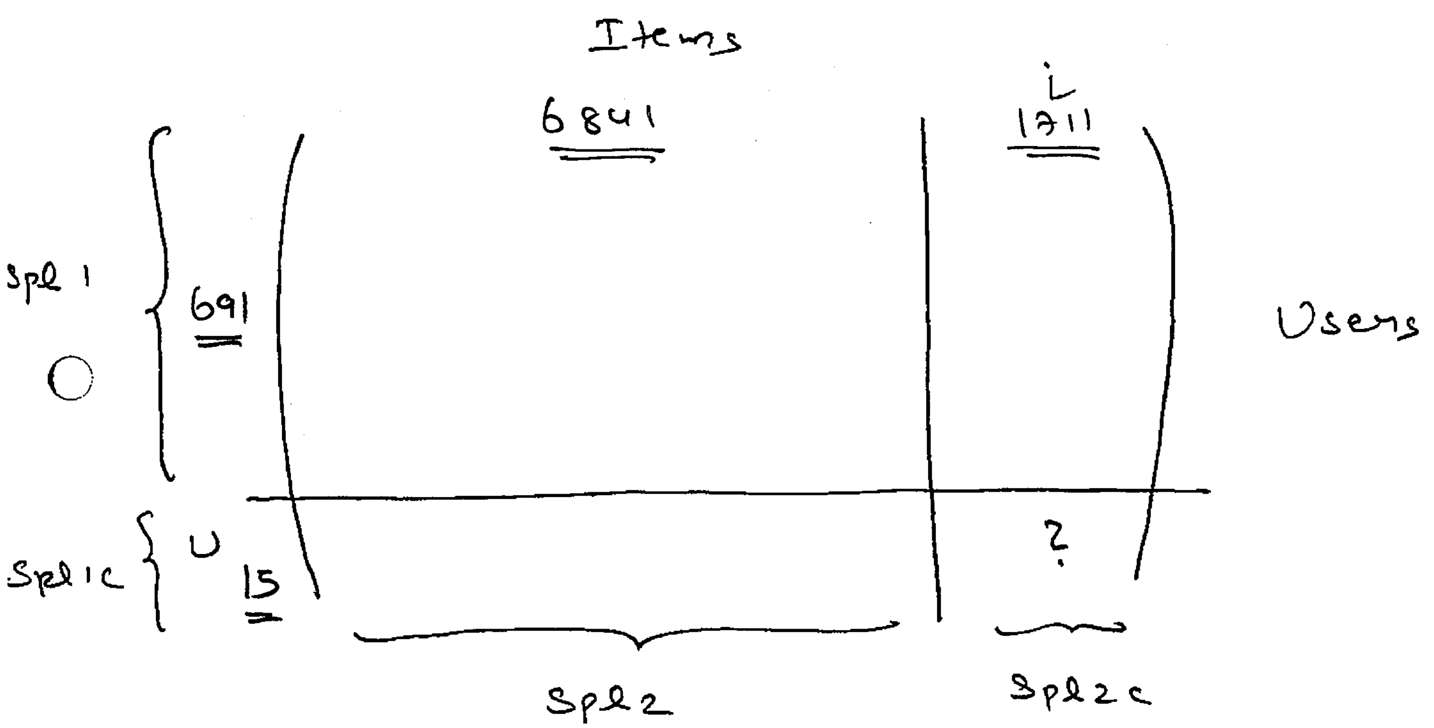


Histogram of as.vector(Data)
Frequency
0
5000
15000
25000
1
2
3
4
5
as.vector(Data)

Datanorm
Histogram of as.vector( )
Frequency
0
5000
15000
-4
-2
0
2
4

```
Set.seed(1)

Spl1 ← sample(1: nrow(Data), 0.98 * nrow(Data))

Set.seed(2)

Spl2 ← sample(1: ncol(Data), 0.8 * ncol(Data))

Spl1c ← setdiff(1: nrow(Data), spl1)

Spl2c ← setdiff(1: ncol(Data), Spl2)
```

- This splits the data into the following format so as to apply recommendation techniques by randomly selecting rows & columns



```
User Pred <- matrix(nrow = length(spl1c), ncol =
                                  length(spl2c))

Base1 <- matrix(nrow = length(spl1c),
                          ncol = length(spl2c))

Base2 <- matrix(nrow = length(spl1c),
                          ncol = length(spl2c))
```

Create three different prediction approaches and corresponding Predicted rating matrices)

```
for(i in 1:length(spl1c)) {
  Base1[i,] <- ColMeans(Data[spl1, spl2c],
                         na.rm = TRUE) }
```

This computes predicted ratings by averaging out the item rating for all the users in the training set (spl1)

```
for (j in 1: length(spl2c)) {
  Base2[,j] <- rowMeans(Data[spl1c, spl2],
                         na.rm = TRUE) }
```

This computes predicted ratings by averaging over the user rating for all the items in the training set (spl1)

```
Cor <- matrix (nrow = length (spl1), ncol = 1)

Order <- matrix (nrow = length (spl1c), ncol = length (spl1))
```

The Cor object will keep track of correlations between users while Order will sort users in terms of decreasing correlations

```
for (i in 1: length (spl1c)) {
  for (j in 1: length (spl1)) {
    Cor[j] <- cor (Data [spl1c[i], spl2] ,
                   Data [spl1 [j], spl2] ,
              use = "pairwise.complete.obs") }
   V <- order (Cor, decreasing = TRUE , na.last = NA)
  Order[i,] <- c(V, rep(NA, times = length (spl1) - length(V)))
  }
```

This creates an Order matrix of dimension 15 x 691 where each row corresponds to neighbors of the users in the spl1c in decreasing order of Pearson correlation

The NA's increasing that there are some users who have no common ratings of movies with the user.

```
for (i in 1: length (spl1c)) {

  UserPred [i, ] <- ColMeans ( Data [Spl1 [Order [i, 1:250]
                                Spl 2C], na.rm = TRUE )
```

This computes user predictions by looking at the 250 nearest neighbors & averaging equally over all these user ratings in the items in Spl 2C

```
RMSEUserPred <- Sqrt( mean( (Data [Spl1C, Spl2C]
                              - UserPred )^2,
                        na.rm = TRUE ) )
```

RMSE = 0.898

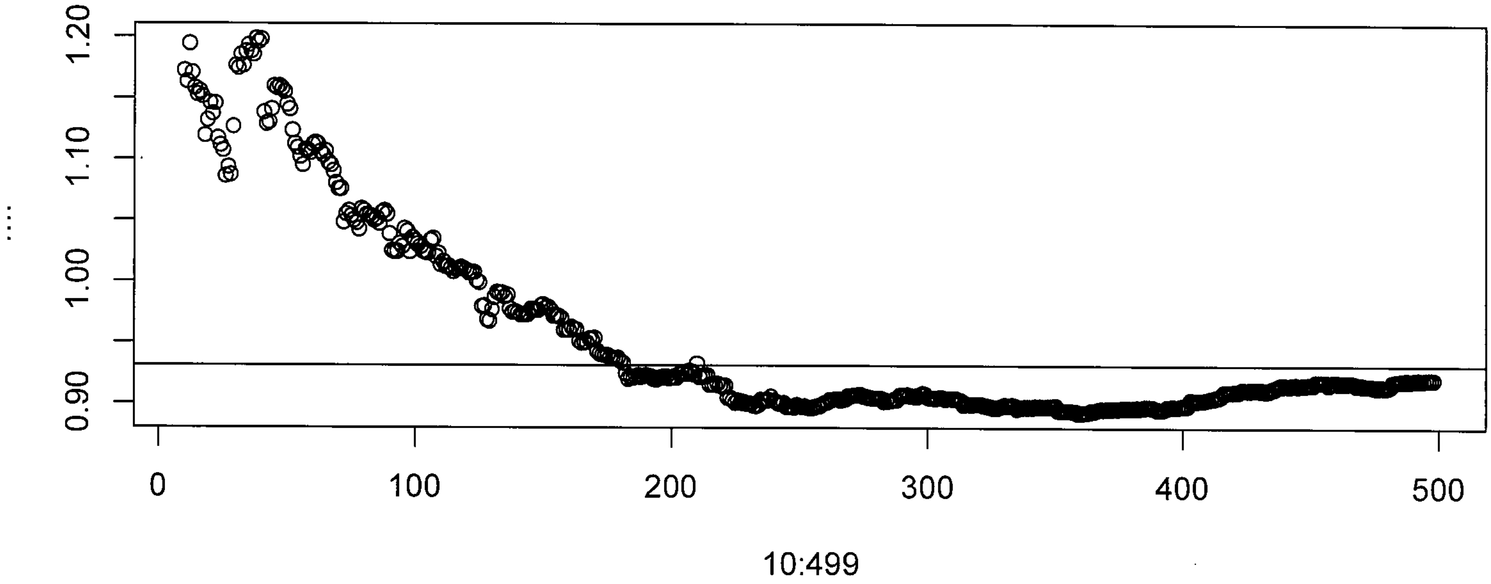
```
RMSEBase1 <- Sqrt( mean( (Data [Spl1C, Spl2C]
                          - Base 1 )^2,
                     na.rm = TRUE ) )

RMSEBase2 <- Sqrt( mean( (Data [Spl1C, Spl2C]
                          - Base 2 )^2,
                     na.rm = TRUE ) )
```

RMSE = 0.931 & 0.995 respectively

We can also vary the neighborhood set to see the effect & also try weighting.

```
RMSE <- rep(NA, times = 490)
for (k in 10 : 499) {
  for (i in 1: length( spl1c)) {
    UserPred [i, ] <- colMeans ( Data [ spl1[order [i,1:k]]
                                  spl2c], na.rm = TRUE)
  }
RMSE (k-10) <- sqrt ( mean( ( (Data [ spl1c, spl2c])
                                   - UserPred )^2
                       na.rm = TRUE
```

```
plot (10 : 499, rm )
abline (h = 0.931)
```


1.20
1.10
1.00
0.90
0
100
200
300
400
500
10:499