Managing/analyzing the Netflix data
Russ Lenth
22S:295 HPC Seminar
October 25, 2007

1 Background
The Netflix prize
- For details: www.netflixprize.com
- $1 million prize for beating Cinematch program for predicting movie ratings by 10%
- Annual progress prize of $50K.
- Cinematch RMSE is 0.9525; $1M goal 0.8572
- Contest begins October 2, 2006 and continues through at least October 2, 2011
- Current leaders (as of Oct. 19): “BellKor” team (Bob Bell, Yehudi Koren, AT&T Research), RMSE = 0.8709

2 Data
The data
- Training data variables: Movie ID, Customer ID, Date, Rating (1–5)
- About 18,000 movies, 480,000 customers, and over 100 million observations
- Packaged as 17,770 separate text files, one for each movie
- These files are saved (gzip format) and available to all in /space/yoyo/data/Netflix/training_data

mv_0012345.txt
0012345 0365262 5 2005-05-04 1076294 3 2005-03-07
. . . 2209921 4 2006-12-23

To read a movie file in R
read.movie = function(movieno) {
  fname = sprintf("/space/yoyo/data/Netflix/training_set/mv_%07d.txt.gz", movieno)
  con = gzfile(fname, "rb")
  lst = scan( con, skip=1, sep="",
             what = list(cust=0, rating=0, date="" )
     )
  close(con)
  lst$rating = as.Date(lst$rating)
  lst
}

2.1 Movie summary statistics
Movie summaries
> mv.summ = function(movieno) {
+  dat = read.movie(movieno)
+  c(length(dat$rating), mean(dat$rating), sd(dat$rating))
+ }
> # Using cluster with 8 processors ...
> system.time(snumm <<- parLapply(cl, 1:17770, mv.summ))
user system elapsed
0.026 0.003 0.050
> mstats = matrix(unlist(snumm), nrow=3)
> sum(mstats[1,])
[1] 100480507
> sum(mstats[1,]*mstats[2,]) / .Last.value
[1] 3.60429

More movie summaries
> summary(mstats[1,])
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
 3.000000 192.00000  561.000000  5655.0000 2668.0000  232900.0000
> summary(mstats[2,])
     Min.  1st Qu.   Median      Mean   3rd Qu.      Max.
1.2880 2.8970  3.2550       3.2280 3.6160  4.7230
> summary(mstats[3,])
     Min. 1st Qu.   Median      Mean   3rd Qu.     Max.
0.5865 1.0100  1.0910       1.1010 1.1820  1.6480

More movie summaries
> hist(mstats[2,], xlab="Mean movie rating")

Histogram of mstats[2, ]

Mean movie rating
Frequency
0 500 1000 2000
0 1 2 3 4 5
1.5 2.0 2.5 3.0 3.5 4.0 4.5
Is it worth it to make native R files?

```r
makeR = function(movieno) {
  attach(read.movie(movieno))
  fname = sprintf("/space/yoyo/data/Netflix/training_set/mv_%07d.RData", movieno)
  save(list=c("cust","rating","date"), file=fname)
  detach()
}
```

```r
> system.time(parLapply(cl, 1:17770, makeR))
user system elapsed
0.012 0.003 231.125
```

```r
newmv.summ = function(movieno) {
  fname = sprintf("/space/yoyo/data/Netflix/training_set/mv_%07d.RData", movieno)
  load(fname)
  c(length(rating), mean(rating), sd(rating))
}
```

```r
> system.time(nmsumm <<- parLapply(cl, 1:17770, newmv.summ))
user system elapsed
0.039 0.002 15.072
```

Yes!!—It takes less than 1/9 the time to do the same thing

2.2 Rearranging the data

Rearranging the data

- Provided data is fine for computing mean ratings per movie and other movie-specific quantities
- Far less convenient for computing customer effects
- To do this, we need to create a new set of files, each with all the data for just a handful of customers.
- (One file per customer would be too many files)
- How to accomplish this without reading/sorting all 17,770 movie files together?

Slice and dice algorithm

First pass

1. Combine the data for 10 movies
   (a) Extract all the data for customer IDs that start with 0 and save to a new file
   (b) Extract all the data for customer IDs that start with 1 and save to a new file
   (c) ...

2. Repeat this operation for 1,769 other sets of 10 movies

Second pass Do the same using sets of 10 (or so) result files, extracting new files based on the second digits of the customer IDs

Eventually If we manage it right, we consolidate all data for each customer into one file (a few customers per file)

Bookkeeping for slicing/dicing

- Use filenames cu CC ... MM ... to keep track of information, stripping off last digit each iteration

  1. mv-0012340, mv-0012341, ..., mv-0012349
     → cu_0-001234, cu_1-001234, ..., cu_9-001234

  2. cu-2-001230, cu-2-001231, ..., cu-2-001239
     → cu_20-00123, cu_21-00123, ..., cu_29-00123

  3. cu_25-00120, cu_25-00121, ..., cu_25-00129
     → cu_250-0012, cu_251-0012, ..., cu_259-0012

  4. ...  

  5. ...  

     → cu_25430-00, cu_25431-00, ..., cu_25439-00

At this stage, all suffixes are -00, and no customer’s data exists in more than one file.

0th step (using 4 processors)

```r
> system.time(parNFSetup(cl))
Farming out the job for 178 patterns...
user system elapsed
0.234 0.033 444.328
```

1st step

```r
> system.time(parSD(cl))
We processed 17770 files in 1778 patterns.
user system elapsed
0.369 0.100 279.603
```
2nd step

```r
> system.time(parSD(cl))
We processed 5334 files in 534 patterns.
user system elapsed
 0.104  0.045 235.899
```

```r
> peek()
We have 4806 files in all...
[1] "cu_00-0000.RData" "cu_00-0001.RData" "cu_00-0002.RData"...
```

3rd step

```r
> system.time(parSD(cl))
We processed 4806 files in 486 patterns.
user system elapsed
 0.091  0.043 196.213
```

```r
> peek()
We have 4770 files in all...
[1] "cu_00-0000.RData" "cu_00-0001.RData" "cu_00-0002.RData"...
```

4th step

```r
> system.time(parSD(cl))
We processed 4770 files in 530 patterns.
user system elapsed
 0.078  0.048 196.922
```

```r
> peek()
We have 5300 files in all...
[1] "cu_00-0000.RData" "cu_00-0001.RData" "cu_00-0002.RData"...
```

5th step

```r
> system.time(parSD(cl))
We processed 5300 files in 2650 patterns.
user system elapsed
 0.092  0.044 388.795
```

```r
> peek()
We have 26495 files in all...
[1] "cu_00000-00.RData" "cu_00001-00.RData" "cu_00002-00.RData"...
```

6th step—NOT

```r
> system.time(parSD(cl))
No more slicing/dicing is necessary. Files have been renamed
user system elapsed
 2.632  1.778 220.862
```

2.3 Customer summary statistics

Customer summaries

```r
> cu.summ = function(file) {
+ load(paste(NFpath,file,sep="/"))
+ tapply(rating, cust, function(r) c(length(r),mean(r),sd(r)))
+ }
```

```r
> system.time(csumm <<- parLapply(cl, dir(path=NFpath,pat="cu_*"),
cu.summ))
```

```r
user system elapsed
 6.065  0.468 49.854
```

```r
cstats = matrix(unlist(csumm), nrow=3)
cust=as.integer(unlist(lapply(csumm, names)))
```

```r
sum(cstats[1,])
[1] 100480507
```

```r
sum(cstats[1,]*cstats[2,]) / sum(cstats[1,])
[1] 3.60429
```

These results confirm that we have the same data as from the movie files

More customer stats

```r
> length(cust)
[1] 480189
```

```r
> summary(cust)
   Min. 1st Qu.  Median    Mean 3rd Qu.   Max. NA's
6     659100 1323000 1323000 1986000 2649000  13999
```

```r
> summary(cstats[2,])
   Min. 1st Qu.  Median    Mean 3rd Qu.   Max.   NA's
1.000 3.380   3.676   3.674   3.980   5.000 1269.000
```

```r
> summary(cstats[1,])
   Min. 1st Qu.  Median    Mean 3rd Qu.   Max.   NA's
1.0   39.0    96.0    209.3  259.0   17650.0 1269.000
```

More customer stats

```r
> hist(cstats[2,], xlab="Mean customer rating")
```

Histogram of cstats[2,]

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
3 Analysis

3.1 Time trends

Time trends

Do ratings change systematically over time? A simple analysis we can do is find the slopes of the regression lines for each movie.

```r
> date.trend
function(movieno) {
  read.movie(movieno)
  d.dev = as.integer(date) - mean(as.integer(date))
  365.25 * sum(d.dev*rating) / sum(d.dev*d.dev)
}

> system.time(date.trends <<- parSapply(cl, 1:17770, date.trend))
```

```
user system elapsed
0.065 0.001 14.002
```

```r
> summary(date.trends)
```

```r
Min. 1st Qu. Median Mean 3rd Qu. Max.
-11.85000 0.01564 0.09913 0.09450 0.20230 15.19000
```

```r
> hist(date.trends[abs(date.trends)<.5], main="")
```

Histogram of inlying slopes

3.2 ANCOVA model

An analysis-of-covariance model

If we take a traditional linear-models approach, we might want to fit a model of the form

\[ E(r_{ij}) = \beta_0 + \mu_i + \beta_i(x_{ij} - \bar{x}_i) + \kappa_j \]

where \( r_{ij} \) is the rating of the \( i \)th movie by the \( j \)th customer and \( x_{ij} \) is the \( (i,j) \)th date, \( i = 1, 2, \ldots, 17770 \), \( j = 1, 2, \ldots, 480189 \), subject to the constraints

\[ \sum_{i=1}^{17770} \mu_i = 0 \]
\[ \sum_{j=1}^{480189} \kappa_j = 0 \]

- With appropriate indicator variables, etc., the \( X \) matrix for this model has 100, 480, 507 rows and 515, 728 columns. and \( X'X \) has 2.66 \times 10^{11} \) elements.
- Maybe we should find a different approach...

3.3 Iterative method

Iterative method

Here is an approach dating back to the “old days” (but not unlike the ideas behind Gibbs sampling)

1. Start with initial guesses for parameter estimates
2. Loop:
   (a) Estimate the \( \mu_i \) after adjusting for the \( \beta_i \) and \( \kappa_j \)
   (b) Estimate the \( \beta_i \) after adjusting for the new \( \mu_i \) and \( \kappa_j \)
   (c) Estimate the \( \kappa_j \) after adjusting for the new \( \mu_i \) and new \( \beta_i \)
3. Repeat (2) until estimates stabilize

3.4 R functions

R functions for iterative analysis

We’ll need each movie’s mean date

```r
> get.mean.date = function(movieno) {
+  read.movie(movieno)
+  mean(as.integer(date))
+ }

> mean.date = parSapply(cl, 1:17700, get.mean.date)
```

And we need some initial values

```r
> cu.eff = cstats[2,] - 3.6
> mv.eff = matrix(rep(0,2*17770), nrow=2)
```

Code for movie effects

```r
est.mv.effs = function (movieno, lambda0=0, lambda1=0) {
  read.movie(movieno)
  xdev = as.integer(date) - mean.date[movieno]
  ydev = rating - 3.6
  - sapply(cust, function(c) cu.eff[cu.pos[c]])
  avg = sum(ydev) / (lambda0 + length(ydev))
  slope = sum(xdev*ydev) / (lambda1 + sum(xdev*xdev))
  c(avg, slope)
}
```

```r
mv.eff <<- mv.eff
```

Code for customer effects

```r
est.cu.effs = function (filename, lambda=0) {
  load(paste(NFpath,filename,sep="/"))
  deff = as.integer(date) - sapply(movie, function(m) mean.date[m])
  deff = deff * sapply(movie, function(m) mv.eff[2,m])
  ydev = rating - 3.6 - deff
  - sapply(movie, function(m) mv.eff[1,m])
  tapply(ydev, cust, function(e) sum(e) / (lambda + length(e)))
}
```

```r
update.cu = function(cl) {
  clusterExport(cl, "cu.eff")
  ce = unlist(parLapply(cl, custfiles, est.cu.effs))
  chg = c(max=max(ce - cu.eff), RMS=sqrt(mean((ce-cu.eff)^2)))
  cu.eff <<- ce
  chg
}
```

4
3.5 Results

Iterations
> update.mv(cl)
  max.eff RMS.eff max.slope RMS.slope
  2.146194510 0.522287975 0.037305680 0.001179864
> update.cu(cl)
  max RMS
  1.4802265 0.1243077

> update.mv(cl)
  max.eff RMS.eff max.slope RMS.slope
  0.2349055528 0.0645016692 0.0054959693 0.0001473149
> update.cu(cl)
  max RMS
  0.17133869 0.01897151

> update.mv(cl)
  max.eff RMS.eff max.slope RMS.slope
  4.246874e-02 1.183684e-02 1.386837e-03 4.324022e-05
> update.cu(cl)
  max RMS
  0.039378870 0.007066787

Iterations (cont’d)
> update.mv(cl)
  max.eff RMS.eff max.slope RMS.slope
  1.521279e-02 3.48561e-03 4.119564e-04 1.889038e-05
> update.cu(cl)
  max RMS
  0.02063310 0.004243885

> update.mv(cl)
  max.eff RMS.eff max.slope RMS.slope
  9.900898e-03 1.802297e-03 1.697929e-04 1.072775e-06
> update.cu(cl)
  max RMS
  0.01381076 0.002765469

• Pretty close after 5 times around.

• Computation time (10 nodes): Around 75 seconds for each update.mv and 175 seconds for each update.cu run.

Summaries
> summary(cu.eff)
 Min. 1st Qu. Median Mean 3rd Qu. Max.
-3.480000  -0.224700  0.058611  0.068900  0.358100  2.445000

> apply(mv.eff, 1, summary)
 [,1] [,2]
Min.  -2.302000  -3.486002
1st Qu.  -0.60850  -1.235004
Median  -0.249600  8.504000
Mean    -0.289200  -8.195006
3rd Qu.  0.085230  2.896004
Max.    1.077000  4.046002

3.6 Ridge regression

Ridge regression
• Substantial risk of over-fitting
• Especially considering spareseness of data
• Ridge-regression idea: essentially pretend that we have \( \lambda \) additional zero values for each movie (or customer)
• Shrinks estimates towards zero — especially those with small denominators

Modified code
# Save old estimates for comparison
> CU.eff = cu.eff
> MV.eff = mv.eff
>
> fix(update.cu)
> update.cu
function(cl, lambda=50) {
  clusterExport(cl, "mv.eff")
  ce = unlist(pardapply(cl, custfiles, est.cu.effs, lambda))
  chg = c(maxmax(ce - cu.eff), RMS=sqrt(mean((ce-cu.eff)^2)))
  cu.eff <<- ce
}

etc.

Iterations
First round
> update.mv(cl)
  max.eff RMS.eff max.slope RMS.slope
  1.606790689 0.321115611 0.001000710
> update.cu(cl)
  max RMS
  3.4119180 0.2481984

Fourth round
> update.mv(cl)
  max.eff RMS.eff max.slope RMS.slope
  1.109536e-02 5.890785e-03 1.940836e-05 4.303194e-06
> update.cu(cl)
  max RMS
  0.0004275699 0.0060686696

Comparisons of two estimates

• A plot of 480,000 customer effects is a bit messy. I took a random sample of 1,000; same for the movie effects.

• The reference line is the identity line.

3.7 Conclusions

Conclusions
• Learning experience
• Parallel computing really helps!
• Snow really helps!
• It is actually possible to fit a multiple regression model with \( n = 10^8 \) and \( p = 5 \times 10^5 \)—and get it done in an hour