Managing/analyzing the Netflix data

Russ Lenth

Department of Statistics & Actuarial Science
The University of Iowa

22S:295 HPC Seminar
October 25, 2007

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 $= 0.8709$

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$date = as.Date(lst$date)
  lst
}
```

Movie summaries

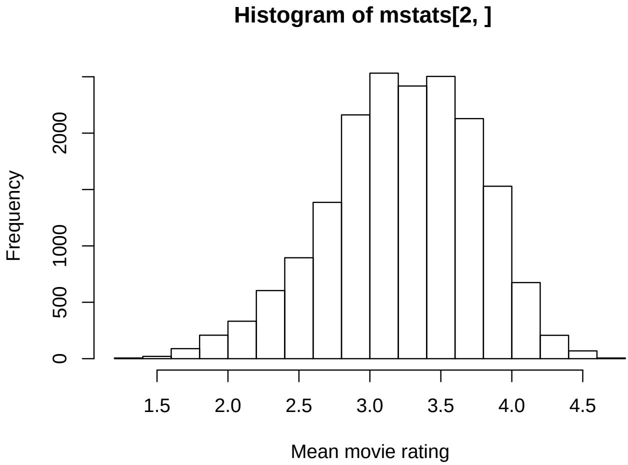
```
> mv.summ = function(movieno) {
+   dat = read.movie(movieno)
+   c(length(dat$rating), mean(dat$rating), sd(dat$rating))
+ }

> # Using cluster with 8 processors ...
> system.time(msumm <<- parLapply(cl, 1:17770, mv.summ))
   user  system elapsed
  0.026   0.003 130.134

> mstats = matrix(unlist(msumm), 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     192     561    5655    2668  232900

> summary(mstats[2,])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.288   2.897   3.255   3.228   3.616   4.723

> 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")
```



Is it worth it to make native R files?

```
> makeR = function(movieno) {
+   attach(read.movie(movieno))
+   fname = sprintf("/space/yoyo/data/Netflix/training_set/mv_%07d.RData",
+   save(list=c("cust","rating","date"), file=fname)
+   detach()
}
> system.time(parLapply(cl, 1:17770, makeR))
   user  system elapsed
  0.012   0.003 231.125
> newmv.summ = function(movieno) {
+   fname = sprintf("/space/yoyo/data/Netflix/training_set/mv_%07d.RData",
+   load(fname)
+   c(length(rating), mean(rating), sd(rating))
+ }
> 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

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
   1. Extract all the data for customer IDs that start with 0 and save to a new file
   2. Extract all the data for customer IDs that start with 1 and save to a new file
   3. ...
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)


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

> peek()
We have 17770 files in all...
 [1] "cu_-0000001.RData" "cu_-0000002.RData" "cu_-0000003.RData"
 [4] "cu_-0000004.RData" "cu_-0000005.RData" "cu_-0000006.RData"
 [7] "cu_-0000007.RData" "cu_-0000008.RData" "cu_-0000009.RData"
[10] "cu_-0000010.RData" "..."               "cu_-0017761.RData"
[13] "cu_-0017762.RData" "cu_-0017763.RData" "cu_-0017764.RData"
[16] "cu_-0017765.RData" "cu_-0017766.RData" "cu_-0017767.RData"
[19] "cu_-0017768.RData" "cu_-0017769.RData" "cu_-0017770.RData"
```

1st step

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

> peek()
We have 5334 files in all...
 [1] "cu_0-000000.RData" "cu_0-000001.RData" "cu_0-000002.RData"
 [4] "cu_0-000003.RData" "cu_0-000004.RData" "cu_0-000005.RData"
 [7] "cu_0-000006.RData" "cu_0-000007.RData" "cu_0-000008.RData"
[10] "cu_0-000009.RData" "..."               "cu_2-001768.RData"
[13] "cu_2-001769.RData" "cu_2-001770.RData" "cu_2-001771.RData"
[16] "cu_2-001772.RData" "cu_2-001773.RData" "cu_2-001774.RData"
[19] "cu_2-001775.RData" "cu_2-001776.RData" "cu_2-001777.RData"
```

2nd step

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

> peek()
We have 4806 files in all...
 [1] "cu_00-00000.RData" "cu_00-00001.RData" "cu_00-00002.RData"
 [4] "cu_00-00003.RData" "cu_00-00004.RData" "cu_00-00005.RData"
 [7] "cu_00-00006.RData" "cu_00-00007.RData" "cu_00-00008.RData"
[10] "cu_00-00009.RData" "..."               "cu_26-00168.RData"
[13] "cu_26-00169.RData" "cu_26-00170.RData" "cu_26-00171.RData"
[16] "cu_26-00172.RData" "cu_26-00173.RData" "cu_26-00174.RData"
[19] "cu_26-00175.RData" "cu_26-00176.RData" "cu_26-00177.RData"
```

3rd step


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

> peek()
We have 4770 files in all...
 [1] "cu_000-0000.RData" "cu_000-0001.RData" "cu_000-0002.RData"
 [4] "cu_000-0003.RData" "cu_000-0004.RData" "cu_000-0005.RData"
 [7] "cu_000-0006.RData" "cu_000-0007.RData" "cu_000-0008.RData"
[10] "cu_000-0009.RData" "..."               "cu_264-0008.RData"
[13] "cu_264-0009.RData" "cu_264-0010.RData" "cu_264-0011.RData"
[16] "cu_264-0012.RData" "cu_264-0013.RData" "cu_264-0014.RData"
[19] "cu_264-0015.RData" "cu_264-0016.RData" "cu_264-0017.RData"
```

4th step


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

> peek()
We have 5300 files in all...
 [1] "cu_0000-000.RData" "cu_0000-001.RData" "cu_0001-000.RData"
 [4] "cu_0001-001.RData" "cu_0002-000.RData" "cu_0002-001.RData"
 [7] "cu_0003-000.RData" "cu_0003-001.RData" "cu_0004-000.RData"
[10] "cu_0004-001.RData" "..."               "cu_2645-000.RData"
[13] "cu_2645-001.RData" "cu_2646-000.RData" "cu_2646-001.RData"
[16] "cu_2647-000.RData" "cu_2647-001.RData" "cu_2648-000.RData"
[19] "cu_2648-001.RData" "cu_2649-000.RData" "cu_2649-001.RData"
```

5th step

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

> peek()
We have 26495 files in all...
 [1] "cu_00000-00.RData" "cu_00001-00.RData" "cu_00002-00.RData"
 [4] "cu_00003-00.RData" "cu_00004-00.RData" "cu_00005-00.RData"
 [7] "cu_00006-00.RData" "cu_00007-00.RData" "cu_00008-00.RData"
[10] "cu_00009-00.RData" "..."               "cu_26485-00.RData"
[13] "cu_26486-00.RData" "cu_26487-00.RData" "cu_26488-00.RData"
[16] "cu_26489-00.RData" "cu_26490-00.RData" "cu_26491-00.RData"
[19] "cu_26492-00.RData" "cu_26493-00.RData" "cu_26494-00.RData"
```

6th step—NOT

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

> peek()
We have 26495 files in all...
 [1] "cu_00000.RData" "cu_00001.RData" "cu_00002.RData"
 [4] "cu_00003.RData" "cu_00004.RData" "cu_00005.RData"
 [7] "cu_00006.RData" "cu_00007.RData" "cu_00008.RData"
[10] "cu_00009.RData" "..."            "cu_26485.RData"
[13] "cu_26486.RData" "cu_26487.RData" "cu_26488.RData"
[16] "cu_26489.RData" "cu_26490.RData" "cu_26491.RData"
[19] "cu_26492.RData" "cu_26493.RData" "cu_26494.RData"
```

Customer summaries


```
> cu.summ = function(file) {
+   load(paste(NFpath,file,sep="/"))
+   tapply(rating, cust, function(r) c(length(r),mean(r),sd(r)))
+ }
> system.time(csumm <<- parLapply(cl, dir(path=NFpath,pat="cu_"),
  cu.summ))
   user  system elapsed
  6.065   0.468  49.854
> cstats = matrix(unlist(csumm), nrow=3)
> cust=as.integer(unlist(lapply(csumm, names)))

> sum(cstats[1,])
[1] 100480507

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

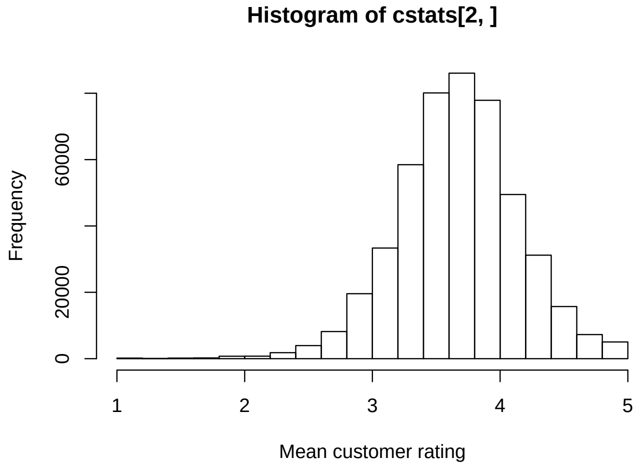
These results confirm that we have the same data as from the movie files

More customer stats


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

> summary(cust)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      6  659100 1323000 1323000 1986000 2649000

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

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

> summary(cstats[3,])
   Min.    1st Qu.    Median      Mean    3rd Qu.      Max.        NA's
 0.0000    0.8406    0.9819    0.9982    1.1410    2.8280  1269.0000
```

More customer stats


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

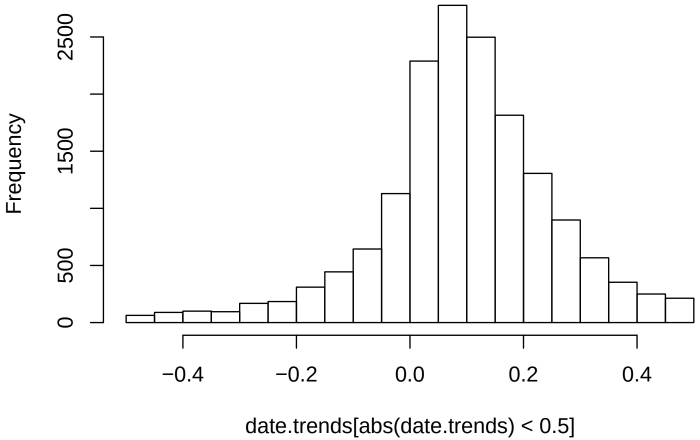


Time trends

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

```
> 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

> summary(date.trends)
     Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-11.85000   0.01564    0.09913    0.09450    0.20230   15.19000

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

Histogram of inlying slopes



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 ith movie by the jth customer and x_{ij} is the (i, j)th date, $i = 1, 2, \ldots, 17770$, $j = 1, 2, \ldots, 480189$, subject to the constraints

$$\sum_{i=1}^{17770} \mu_i \;=\; \sum_{j=1}^{480189} \kappa_j \;=\; 0$$

- With appropriate indicator variables, etc., the $\mathbf{X}$ matrix for this model has $100,480,507$ rows and $515,728$ columns. and $\mathbf{X}'\mathbf{X}$ has 2.66×10^{11} elements.

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 ith movie by the jth customer and x_{ij} is the (i,j)th date, $i = 1, 2, \ldots, 17770$, $j = 1, 2, \ldots, 480189$, subject to the constraints

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

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

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:
   1. Estimate the μ_i after adjusting for the β_i and κ_j
   2. Estimate the β_i after adjusting for the new μ_i and κ_j
   3. Estimate the κ_j after adjusting for the new μ_i and new β_i
3. Repeat (2) until estimates stabilize

R functions for iterative analysis

We'll need each movie's mean date

```
> 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

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

Code for movie effects

```
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, slop> mv.eff = matrix(rep(0,2*17770), nrow=2)
}

update.mv = function(cl) {
  clusterExport(cl, "cu.eff")
  me = parSapply(cl, 1:17770, est.mv.effs)
  chg = c(max.eff = max(abs(me[1,]-mv.eff[1,])),
    RMS.eff = sqrt(mean((me[1,]-mv.eff[1,])^2)),
    max.slope = max(abs(me[2,]-mv.eff[2,])),
    RMS.slope = sqrt(mean((me[2,]-mv.eff[2,])^2)) )
  mv.eff <<- me
  chg
}
```

Code for customer effects

```
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)))
}

update.cu = function(cl) {
  clusterExport(cl, "mv.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
}
```

Iterations

```
> update.mv(cl)
    max.eff     RMS.eff   max.slope   RMS.slope
2.146194510 0.522287975 0.037305960 0.001179864
> update.cu(cl)
      max       RMS
1.4802255 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 3.458161e-03 4.119544e-04 1.898038e-05
> update.cu(cl)
        max         RMS
0.020633110 0.004243885

> update.mv(cl)
     max.eff      RMS.eff    max.slope    RMS.slope
9.900898e-03 1.802297e-03 1.697929e-04 1.072775e-05
> update.cu(cl)
        max         RMS
0.013813076 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.
-3.48000 -0.22470  0.05861  0.06890  0.35810  2.44500

> apply(mv.eff, 1, summary)
            [,1]       [,2]
Min.    -2.30200 -3.486e-02
1st Qu. -0.60850 -1.235e-04
Median  -0.24960  8.504e-05
Mean    -0.28920 -8.195e-06
3rd Qu.  0.08523  2.896e-04
Max.     1.07700  4.046e-02
```

Ridge regression

- Substantial risk of over-fitting
- Especially considering spareseness of data
- Ridge-regression idea: essentially pretend that we have λ 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(parLapply(cl, custfiles, est.cu.effs, lambda))
  chg = c(max=max(ce - cu.eff), RMS=sqrt(mean((ce-cu.eff)^2)))
  cu.eff <<- ce
  chg
}

etc.
```

Iterations

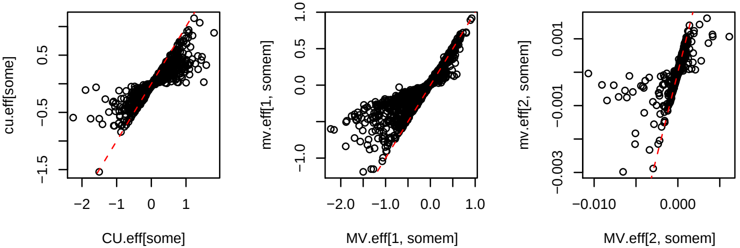
First round

```
> update.mv(cl)
    max.eff     RMS.eff   max.slope   RMS.slope
1.606790689 0.321115611 0.040424673 0.001000710
> update.cu(cl)
      max       RMS
3.4119180 0.2481984
```

Fourth round

```
> update.mv(cl)
     max.eff      RMS.eff    max.slope    RMS.slope
1.105136e-02 5.980754e-03 1.548363e-05 4.951843e-06
> update.cu(cl)
         max          RMS
0.0004277699 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.

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