Personalized Trading Partner Recos on Bitcoin OTC

Prof R. Bapna (adapted from Prof Xuan Bi@UMN (mailto:Bi@UMN))

12/2/2020

In this project we will try to unpack who-trusts-whom on the Bitcoin OTC network. Users are anonymous on this platform and this creates financial risk for traders. The goal is see if we can unpack some latent structure and information in trustworthiness ratings given by traders to each other (much in the same way netflix recommends movies, or OkCupid recommends dates based on voting or liking actions), to give personalized ratings to individual traders. In Bitocin OTC members rate other members on a scale of -10 (total distrust) to +10 (total trust). This will be the key data for this project.

Note, many statistical models, such as logistic regression, tell you about the importance and role of various features for the average user's response. In other words they are not individual level models, but rather, are at the aggregate level. In personalization, like at Amazon, different users, get different recommendations when they browse the same product. How can we achieve this?

The GOAL is to match senders and receivers in a personalized manner on Bitcoin OTC.

2. Load the data

Note that all we have here are sender and receiver ids and ratings [-10, 10]. The timestamps of the rating is not important. This is similar to what Netflix has for instance, where senders are equivalent to users and receivers are similar to movies.

```
otc=read.csv("bitcoin_otc.csv",sep=',',header=F)
colnames(otc)=c("sender","receiver","rating","time")
dim(otc) #35592 ratings
```

```
## [1] 35592     4
```

```
n=35592
max(otc[,1]) #6000 senders
```

```
## [1] 6000
```

```
max(otc[,2]) #6005 receivers
```

```
## [1] 6005
```

3. Split the data for training and testing

#4. Running matrix factorization

```r
r = Reco()
#Tuning
opts = r$tune(data_memory(train_set[,1],train_set[,2],train_set[,3],index1=TRUE),
              opts = list(loss="l2",dim = c(1, 2, 3), lrate = 0.1,nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 3
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.1
##
## $min$costq_l1
## [1] 0.1
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 3.042447
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate loss_fun
## 1    1      0.0     0.01      0.0     0.01   0.1 3.189616
## 2    2      0.0     0.01      0.0     0.01   0.1 3.160889
## 3    3      0.0     0.01      0.0     0.01   0.1 3.104914
## 4    1      0.1     0.01      0.0     0.01   0.1 3.192142
## 5    2      0.1     0.01      0.0     0.01   0.1 3.075817
## 6    3      0.1     0.01      0.0     0.01   0.1 3.156134
## 7    1      0.0     0.10      0.0     0.01   0.1 3.238111
## 8    2      0.0     0.10      0.0     0.01   0.1 3.116456
## 9    3      0.0     0.10      0.0     0.01   0.1 3.054336
## 10   1      0.1     0.10      0.0     0.01   0.1 3.264756
## 11   2      0.1     0.10      0.0     0.01   0.1 3.174722
## 12   3      0.1     0.10      0.0     0.01   0.1 3.134868
## 13   1      0.0     0.01      0.1     0.01   0.1 3.236647
## 14   2      0.0     0.01      0.1     0.01   0.1 3.233854
## 15   3      0.0     0.01      0.1     0.01   0.1 3.111926
## 16   1      0.1     0.01      0.1     0.01   0.1 3.267639
## 17   2      0.1     0.01      0.1     0.01   0.1 3.102820
## 18   3      0.1     0.01      0.1     0.01   0.1 3.148482
## 19   1      0.0     0.10      0.1     0.01   0.1 3.249356
## 20   2      0.0     0.10      0.1     0.01   0.1 3.160505
## 21   3      0.0     0.10      0.1     0.01   0.1 3.120920
## 22   1      0.1     0.10      0.1     0.01   0.1 3.284817
## 23   2      0.1     0.10      0.1     0.01   0.1 3.077244
## 24   3      0.1     0.10      0.1     0.01   0.1 3.116106
## 25   1      0.0     0.01      0.0     0.10   0.1 3.277262
## 26   2      0.0     0.01      0.0     0.10   0.1 3.145951
## 27   3      0.0     0.01      0.0     0.10   0.1 3.100986
## 28   1      0.1     0.01      0.0     0.10   0.1 3.241553
## 29   2      0.1     0.01      0.0     0.10   0.1 3.123467
## 30   3      0.1     0.01      0.0     0.10   0.1 3.088077
## 31   1      0.0     0.10      0.0     0.10   0.1 3.318845
## 32   2      0.0     0.10      0.0     0.10   0.1 3.225906
## 33   3      0.0     0.10      0.0     0.10   0.1 3.078981
## 34   1      0.1     0.10      0.0     0.10   0.1 3.211654
## 35   2      0.1     0.10      0.0     0.10   0.1 3.057475
## 36   3      0.1     0.10      0.0     0.10   0.1 3.084274
## 37   1      0.0     0.01      0.1     0.10   0.1 3.297735
## 38   2      0.0     0.01      0.1     0.10   0.1 3.085921
## 39   3      0.0     0.01      0.1     0.10   0.1 3.119149
## 40   1      0.1     0.01      0.1     0.10   0.1 3.301551
## 41   2      0.1     0.01      0.1     0.10   0.1 3.097191
## 42   3      0.1     0.01      0.1     0.10   0.1 3.132408
## 43   1      0.0     0.10      0.1     0.10   0.1 3.201713
## 44   2      0.0     0.10      0.1     0.10   0.1 3.152183
## 45   3      0.0     0.10      0.1     0.10   0.1 3.042447
## 46   1      0.1     0.10      0.1     0.10   0.1 3.224663
## 47   2      0.1     0.10      0.1     0.10   0.1 3.115639
## 48   3      0.1     0.10      0.1     0.10   0.1 3.097312
```

```
#Training: using the tuned parameters
set.seed(123)
RS3=r$train(data_memory(train_set[,1],train_set[,2],train_set[,3],index1=TRUE), opts = c(loss="l
2",opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       3.3920   3.4181e+05
##    1       3.0597   2.8349e+05
##    2       2.7703   2.3769e+05
##    3       2.5623   2.0799e+05
##    4       2.4001   1.8646e+05
##    5       2.2756   1.7098e+05
##    6       2.1750   1.5902e+05
##    7       2.0929   1.4970e+05
##    8       2.0229   1.4212e+05
##    9       1.9621   1.3567e+05
##   10       1.9101   1.3046e+05
##   11       1.8630   1.2569e+05
##   12       1.8235   1.2192e+05
##   13       1.7884   1.1861e+05
##   14       1.7573   1.1574e+05
##   15       1.7296   1.1321e+05
##   16       1.7051   1.1102e+05
##   17       1.6836   1.0915e+05
##   18       1.6639   1.0741e+05
##   19       1.6463   1.0591e+05
```

```
#Predicting
pred_rvec = r$predict(data_memory(test_set[,1],test_set[,2],test_set[,3],index1=TRUE), out_memor
y())
#Evaluating the performance of the prediction with RMSE
sqrt(mean((test_set[,3]-pred_rvec)^2)) #2.84; a very small RMSE, given the rating range is -10 t
o 10
```

```
## [1] 2.88796
```

#5. Manually check a few predictions

```
#Calculate P and Q (latent-factor matrices)
P_file = out_file(tempfile())
Q_file = out_file(tempfile())
r$output(P_file, Q_file)
```

```
## P matrix generated at C:\Users\rbapna\AppData\Local\Temp\RtmpaSg1GD\file6a3878ef1589
## Q matrix generated at C:\Users\rbapna\AppData\Local\Temp\RtmpaSg1GD\file6a386ed056e
```

```
## $P
## NULL
##
## $Q
## NULL
```

```
P0=read.table(P_file@dest, header = FALSE, sep = " ")
Q0=read.table(Q_file@dest, header = FALSE, sep = " ")
P=matrix(as.numeric(unlist(P0)),dim(P0))
Q=matrix(as.numeric(unlist(Q0)),dim(Q0))
#Manually check predictions with P and Q
test_set[1:11,]
```

```
##    sender receiver rating       time
## 3       1       15      1 1289243140
## 9       2       20      5 1289370622
## 14     21        3      7 1289441526
## 18     10        6      7 1289555731
## 30      6        4      2 1289770700
## 34     17       13      2 1289873368
## 39      4       31      1 1290197549
## 43      7       34      1 1290644458
## 44     34        7      1 1290644477
## 46      1       32      1 1290666953
## 52      7        6      3 1290826591
```

```
#what are sender 1's weights of the 3 latent factors?
P[1,]
```

```
## [1] 1.75056 0.16166 1.79244
```

```
#what are receiver 15's weights of the 3 latent factors?
Q[15,]
```

```
## [1] -0.0485172  0.6977410  0.3027140
```

```
# we can multiply and sum these to get the predicted rating of 1 to 15
sum(P[1,]*Q[15,]) #Compare this value with the actual rating: 1
```

```
## [1] 0.5704612
```

```
#lets try 2 to 20
sum(P[2,]*Q[20,]) #Compare this value with the actual rating: 5
```

```
## [1] 6.116316
```

Conclusion: We were able to estimate two connected 3 dimensional latent spaces that have weights such that their multiplications give the ratings generated in training set. Think of these as three factors that influence trust in the real world, but we don’t know what these factor are. By the end of this algorithm we estimate each sender and each receivers weights for these 3 factors! This will allow us to now take any two people and multiply and add their weights to estimate the level of trust between them. If its high, say above 5, then we can suggest them to trade with each other!