```
> library(DataRemix)
```

1 A Toy Example

In this section, we define a simple objective function called *eval*() which calculates the sum of a penalty term and the squared error between the DataRemix reconstruction and the original input matrix. The input matrix is a 100-by-9 matrix with random values. In this case, we know that when k=9,p=1 or $\mu = 1$, p=1, DataRemix reconstruction is the same as the orginial matrix and the objective function achieves the minimal value which is qual to the penalty term we add.

```
> eval <- function(X_reconstruct, X, penalty){
+   return(-sum((X-X_reconstruct)^2)+penalty)
+ }#eval
>
```

First we genrate a random matrix with dimension 100-by-9 and perform the SVD decomposition.

```
> set.seed(1)
> num_of_row <- 100
> num_of_col <- 9
> X <- matrix(rnorm(num_of_row*num_of_col), nrow = num_of_row, ncol = num_of_col)
> svdres <- svd(X)
```

Set mt to be 2000.

```
> basis_short <- omega[1:2000,]
```

Infer the optimal combinations of k, p and μ. Here X and *penalty* are additional inputs for the *eval*() function.

```
> DataRemix.res <- DataRemix(svdres, eval, k_limits = c(1, length(svdres$d)),
+                   p_limits = c(-1,1), mu_limits = c(1e-12,1),
+                   num_of_initialization = 5, num_of_thompson = 50,
+                   basis = basis_short, xi = 0.1, full = T, verbose = F,
+                   X = X, penalty = 100)
> knitr::kable(cbind(1:55,DataRemix.res$para), align = "l",
+              col.names = c("Iteration","k","p","mu","Eval"))
```

```
|Iteration |k  |p          |mu        |Eval        |
|:---------|:--|:----------|:---------|:-----------|
|1         |8  |0.9343941  |0.8669163 |80.133470   |
|2         |4  |-0.6161244 |0.0822944 |-774.549343 |
|3         |6  |-0.8592770 |0.5276627 |-674.508131 |
|4         |5  |-0.9036173 |0.5945408 |-595.209680 |
|5         |8  |0.1977374  |0.0279159 |-608.454077 |
```

6	2	0.1096308	0.0000000	-798.448854
7	8	1.0000000	0.0232085	41.940176
8	9	0.9908897	0.0057159	99.565228
9	8	-0.2288323	0.0000000	-764.657856
10	8	0.6938650	0.0000000	-201.177694
11	8	0.8569239	0.0000000	-35.687731
12	9	0.7270715	0.0216316	-116.110954
13	8	0.9125185	0.6508176	60.823593
14	7	0.9938341	0.0138750	-33.246379
15	7	1.0000000	0.9835948	99.963172
16	9	0.9992828	0.1334226	99.997252
17	6	1.0000000	0.0027111	-116.012604
18	6	1.0000000	0.0000000	-117.188654
19	9	1.0000000	0.0000000	100.000000
20	7	0.5919201	0.0304955	-348.951951
21	6	0.9495708	0.0000141	-126.990264
22	8	1.0000000	0.3028856	70.428038
23	4	1.0000000	0.0000000	-312.285772
24	9	1.0000000	0.0000000	100.000000
25	9	0.4570922	0.0000000	-399.458625
26	9	0.6942318	1.0000000	-153.172296
27	8	1.0000000	0.0000000	39.148404
28	9	1.0000000	0.0000006	100.000000
29	1	0.7604649	0.0478726	-658.475340
30	7	0.5703131	0.0000000	-376.926859
31	8	0.9392489	0.0024607	23.153371
32	7	1.0000000	0.0000000	-36.839791
33	9	0.8396668	0.0000000	4.800500
34	9	-0.7023136	0.0000013	-824.460170
35	4	0.3708370	0.5039367	-341.185194
36	8	0.9957501	0.0033676	39.466448
37	6	0.5357809	0.0000000	-455.120758
38	8	1.0000000	0.0003565	39.191779
39	8	0.8999003	0.0000588	-1.233873
40	9	0.8910134	0.0000000	50.636469
41	8	1.0000000	0.0000000	39.148404
42	3	1.0000000	0.0000005	-430.786462
43	9	0.9699995	0.0000000	95.511739
44	6	0.2058344	0.0000093	-657.873571
45	9	0.9886725	0.0000001	99.331375
46	9	0.9568334	0.0000000	90.989670
47	8	0.9863858	0.0000000	38.234068
48	9	0.9813945	0.0000000	98.226869
49	9	0.9796213	0.0000058	97.881664
50	9	1.0000000	0.0000001	100.000000
51	9	0.9392123	0.0000000	82.849057

```
|52         |9  |1.0000000  |0.0000000 |100.000000  |
|53         |9  |0.9296450  |0.0000000 |77.527953   |
|54         |9  |1.0000000  |1.0000000 |100.000000  |
|55         |8  |0.7535881  |0.0000005 |-137.631862 |
```

2 GTex

In this section, we show a different task of recovering known pathways based on the GTex gene expression data. $corMatToAUC()$ is the main objective function with two inputs: *data* and *GS*. Here *data* is the correlation matrix across all genes and *GS* is a binary matrix with indicates the canonical mSigDB pathways. We formally define the object as the average AUC across pathways and we also keep track of the average AUPR value.

```
> library(DataRemix)
> library(ROCR)
> library(caTools)
> auc_pr<-function(obs, pred) {
+   xx.df <- prediction(pred, obs)
+   perf  <- performance(xx.df, "prec", "rec")
+   xy    <- data.frame(recall=perf@x.values[[1]], precision=perf@y.values[[1]])
+   xy <- subset(xy, !is.nan(xy$precision))
+   #add a point at 0
+   xy <- rbind(c(0, xy[1,2]), xy)
+   res   <- trapz(xy$recall, xy$precision)
+   return(res)
+ }#auc_pr
> simpleAUC<-function(lab, value){
+   value=as.numeric(rank(value)-1)
+   posn=as.numeric(sum(lab==1))
+   negn=as.numeric(sum(lab!=1))
+   stat=sum(value[lab==1])-posn*(posn+1)/2
+   return(stat/(posn*negn))
+ }#simpleAUC
> perPathPR<-function(pathPredict, GS){
+   length=ncol(pathPredict)
+   pathPR=double(length)
+   for(i in 1:length){
+     x=pathPredict[,i]
+     y=GS[,i]
+     auc=auc_pr(y, x)
+     pathPR[i]=auc
+   }
+   return(pathPR)
+ }#perPathPR
> perPathAUC<-function(pathPredict, GS){
```

```
+   length=ncol(pathPredict)
+   pathPR=double(length)
+   for(i in 1:length){
+     x=pathPredict[,i]
+     y=GS[,i]
+     auc=simpleAUC(y, x)
+     pathPR[i]=auc
+   }
+   return(pathPR)
+ }#perPathAUC
> corMatToAUC=function(data, GS){
+   #self-correlation is 0
+   diag(data)=0
+   data[is.na(data)]=0
+   pathPredict=data%*%GS
+   #fix values for genes that are in the pathway since they only get n-1 correlations
+   nGenes=colSums(GS)
+   #factor to multiply by
+   pathwayF=unlist(lapply(nGenes, function(x){x/(x-1)}))
+   for(i in 1:ncol(GS)){
+     iigenes=which(GS[,i]==1)
+     pathPredict[iigenes,i]=  pathPredict[iigenes,i]*pathwayF[i]
+   }
+
+   PATHAUPR=perPathPR(pathPredict, GS)
+   PATHAUC=perPathAUC(pathPredict, GS)
+   return(c(mean(PATHAUPR),mean(PATHAUC)))
+ }#corMatToAUC
```

Load the data. $GTex_cc$ stands for the correlation matrix and *canonical* represents the pathway matrix. It takes time to decompose $GTex_cc$, thus we precompute the SVD decomposition of $GTex_cc$ and load it as $GTex_svdres$.

```
> load(url("https://www.dropbox.com/s/o949wkg76k0ccaw/GTex_cc.rdata?dl=1"))
> load(url("https://www.dropbox.com/s/wsuze8w2rp0syqg/GTex_svdres.rdata?dl=1"))
> load(url("https://github.com/wgmao/DataRemix/blob/master/inst/extdata/canonical.rdata?raw=
> #svdres <- svd(GTex_cc)
```

Run corMatToAUC() on the default correlation matrix $GTex_cc$.

```
> GTex_default <- corMatToAUC(GTex_cc, canonical)
> GTex_default
```

```
[1] 0.0450869 0.7238648
```

Set mt to be 2000.

```
> basis_short <- omega[1:2000,]
```

Infer the optimal combinations of k, p and μ. Here GS is the additional input for the $corMatToAUC()$ function.

```
> DataRemix.res <- DataRemix(GTex_svdres, corMatToAUC,
+                            k_limits = c(1, length(svdres$d)%/%2),
+                            p_limits = c(-1,1), mu_limits = c(1e-12,1),
+                            num_of_initialization = 5, num_of_thompson = 150,
+                            basis = basis_short, xi = 0.1, full = T, verbose = F,
+                            GS = canonical)
> knitr::kable(cbind(1:15,DataRemix.res$full[order(DataRemix.res$para[,4],decreasing = T)
+              [1:15],]), align = "l", col.names = c("Iteration","k","p","mu",
+              "mean AUPR", "mean AUC"))
```

Iteration	k	p	mu	mean AUPR	mean AUC
1	4	0.5381316	0.1386310	0.0550813	0.7379277
2	4	0.4863602	0.0907230	0.0548421	0.7378860
3	4	0.8423897	1.0000000	0.0552223	0.7377778
4	4	0.6020969	0.1812639	0.0547284	0.7376984
5	4	0.6169342	0.2563718	0.0552114	0.7376680
6	4	0.8645731	1.0000000	0.0549222	0.7375448
7	4	0.8097002	1.0000000	0.0551748	0.7370142
8	4	0.7540819	0.7011658	0.0551882	0.7370007
9	4	0.8085108	1.0000000	0.0551745	0.7369641
10	4	0.3236219	0.0256532	0.0537234	0.7368677
11	4	0.7882419	1.0000000	0.0549466	0.7358457
12	4	0.3906252	0.0347955	0.0529104	0.7355947
13	4	0.7134235	0.6411006	0.0548220	0.7354232
14	4	0.4074656	0.0379034	0.0526801	0.7353147
15	4	0.7442313	0.8183563	0.0546631	0.7349044