Reconstructing phenotype-specific multi-omics networks with SmCCNet

***W.Jenny Shi*[*1], *Laura Saba*[1], *and Katerina Kechris*[1]**

[1]University of Colorado Denver|Anschutz Medical Campus

[*]wjennyshi@gmail.com

June 4, 2018

Abstract

Sparse multiple canonical correlation network analysis (SmCCNet) is a machine learning technique for integrating multiple omics data on the same subjects, along with a quantitative phenotype of interest, and reconstructing multi-omics networks that are specific to the phenotype. While the current version integrates two omics data types in addition to a phenotype, the framework can be easily generalized to more than two omics data types and multiple quantitative phenotypes. In this document, we illustrate a standard workflow of SmCCNet with a synthetic miRNA, mRNA expression dataset.

Contents

1 SmCCNet overview

Note: if you use SmCCNet in published research, please cite:

> Shi, W.J., Y. Zhuang, P.H. Russell, B.D. Hobbs, M.M. Parker,P.J. Castaldi, P. Rudra, B. Vestal, C.P. Hersh, L.M. Saba, and K. Kechris, "Unsupervised Discovery of PhenotypeSpecific Multi-Omics Networks." (*Submitted*)

1.1 Workflow

SmCCNet is a canonical correlation based integration method that reconstructs phenotype-specific multi-omics networks (Shi et al., *submitted*). The algorithm is based on sparse multiple canonical analysis (SmCCA) for two omics data X_1, X_2 and a quantitative phenotype Y measured on the same subjects. SmCCA finds the canonical weights w_1, w_2 that maximize the (weighted or unweighted) sum of pairwise canonical correlations between X_1, X_2 and Y, under some constraints (Equation 1). In SmCCNet, the sparsity constraint functions $P_s(\cdot), s = 1, 2$, are the least absolute shrinkage and selection operators (LASSO). The weighted version corresponds to a, b, c not all equal; the unweighted version corresponds to $a = b = c = 1$.

$$(w_1, w_2) = \arg\max_{\tilde{w}_1, \tilde{w}_2} \left(a\tilde{w}_1^T X_1^T X_2 \tilde{w}_2 + b\tilde{w}_1^T X_1^T Y + c\tilde{w}_2^T X_2^T Y\right), \quad \text{subject to } \|\tilde{w}_s\|^2 = 1, P_s(\tilde{w}_s) \le c_s, s = 1, 2. \tag{1}$$

The sparsity penalties c_1, c_2 influence how many features will be included in each subnetwork. With pre-selected sparsity penalties, the SmCCNet algorithm creates a network similarity matrix based on SmCCA canonical weights from repeated subsampled omics data and the phenotype, and then finds multi-omics modules that are relevant to the phenotype. The subsampling scheme analyzes a subset of omics features at a time and enables simultaneous identification of multiple subnetworks with different connection strength. Forming final similarity matrix by aggregating results from subsamples also improves network robustness. The general workflow (Figure 1) involves three steps:

- Step I: Determine SmCCA sparsity penalties c_1, c_2. The user can select the penalties for omics feature selection based on the study purpose and/or prior knowledge. Alternatively, one can pick sparsity penalties based on a K-fold cross validation (CV) procedure that minimizes the total prediction error (Figure 2). The K-fold CV procedure ensures selected penalties to be generalizable to similar independent data sets and prevents over-fitting.
- Step II: Randomly subsample omics features without replacement, apply SmCCA with chosen penalties and compute a feature relationship matrix for each subset. Repeat the process many times and define the similarity matrix to be the average of all feature relationship matrices.
- Step III: Apply a hierarchical tree cutting to the similarity matrix to find the multi-omics networks. This step simultaneously identifies multiple subnetworks.

1.2 SmCCNet package

The SmCCNet package has the following dependencies:

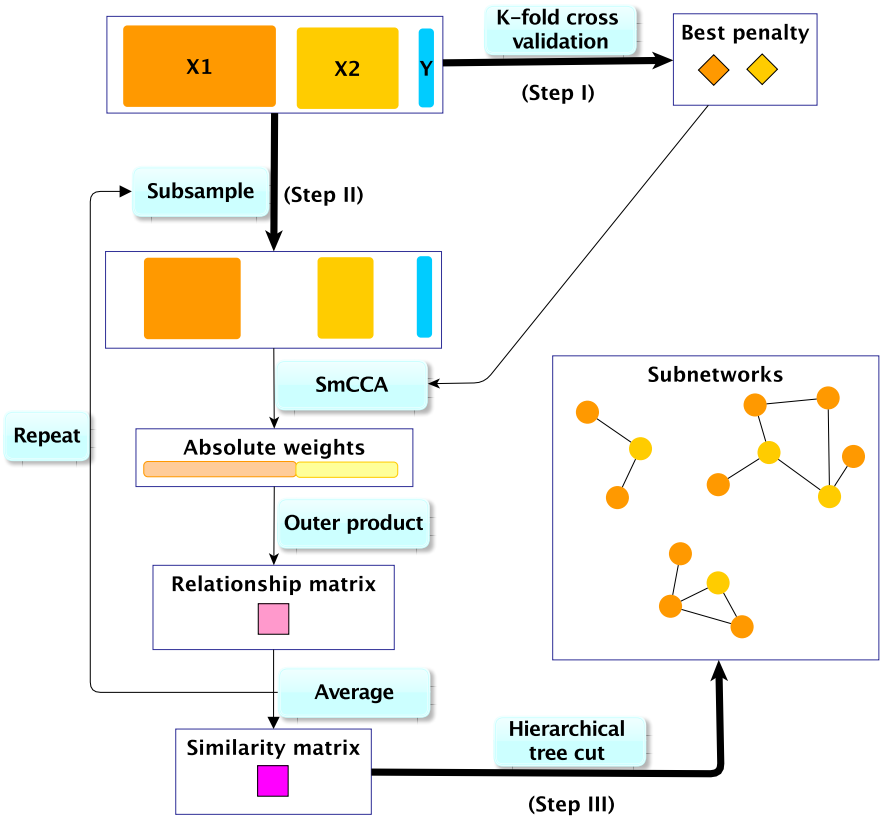

Figure 1: SmCCNet workflow overview
X1 and X2 are two omics data types for the same set of n subjects. Y indicates a quantitative phenotype measure for those n subjects.

```
library(PMA)
library(pbapply)
library(Matrix)
library(igraph)
```

The SmCCNet package consists of two R scripts:

```
source("../R/ModifiedPMA.R")
source("../R/SmCCNetSource.R")
```

The current version of the SmCCNet package includes four (external) functions:

- **getRobustPseudoWeights()**: Compute aggregated (SmCCA) canonical weights.
- **getAbar()**: Calculate similarity matrix based on canonical weights.
- **getMultiOmicsModules()**: Perform hierarchical tree cutting on the similarity matrix and extract clades with multi-omics features.
- **plotMultiOmicsNetwork()**: Plot (trimmed or full) multi-omics subnetworks.

More details on above functions can be found in a separate package description document.

2 SmCCNet workflow with a synthetic dataset

2.1 Synthetic dataset

For the illustration we consider a synthetic data set with 500 genes (X_1) and 100 miRNAs (X_2) expression levels measured for 358 subjects, along with a quantitative phenotype (Y).

```
load("../data/ExampleData.Rdata")
head(X1[ , 1:6])
##          Gene_1   Gene_2   Gene_3   Gene_4   Gene_5   Gene_6
## Samp_1 22.48570 40.35372 31.02575 20.84721 26.69729 30.20545
## Samp_2 37.05885 34.05223 33.48702 23.53146 26.75463 31.73594
## Samp_3 20.53077 31.66962 35.18957 20.95254 25.01883 32.15723
## Samp_4 33.18689 38.48088 18.89710 31.82330 34.04938 38.79989
## Samp_5 28.96198 41.06049 28.49496 18.37449 30.81524 24.00454
## Samp_6 18.05983 29.55471 32.54002 29.68452 26.19996 26.76684
head(X2[ , 1:6])
##           Mir_1    Mir_2    Mir_3    Mir_4    Mir_5     Mir_6
## Samp_1 15.22391 17.54583 15.78472 14.89198 10.34821  9.689755
## Samp_2 16.30697 16.67283 13.36153 14.48855 12.66090 11.333613
## Samp_3 16.54512 16.73501 14.61747 17.84527 13.82279 11.329333
## Samp_4 13.98690 16.20743 16.29308 17.72529 12.30056  9.844108
## Samp_5 16.33833 17.39387 16.39792 15.85373 13.38767 10.599414
## Samp_6 14.54110 16.51999 14.73958 15.87504 13.21359 10.922393
head(Y)
##           Pheno
## Samp_1 235.0674
## Samp_2 253.5450
## Samp_3 234.2050
## Samp_4 281.0354
## Samp_5 245.4478
## Samp_6 189.6231
```

Denote the number of features in $X_1 \& X_2$ as $p_1 \& p_2$ respectively, and the number of subjects as n.

```
p1 <- ncol(X1)
p2 <- ncol(X2)
n <- nrow(X1)
Y.scaled <- scale(Y, center = TRUE, scale = TRUE)
AbarLabel <- c(colnames(cbind(X1, X2)))
```

Although SmCCNet does not require normality, it calculates the Pearson correlation between linear combinations of omics features and the phenotype, which assumes finite variances and finite covariance. It is necessary to include a transformation if the data are skewed. The algorithm also requires standardization (center and scale) of the data.

2.2 Step I: Determine optimal sparsity penalties through CV (optional)

To find the optimal sparsity penalties c_1, c_2, we apply a K-fold CV on the synthetic data (Figure 2). Note that under LASSO constraints, $1 \leq c_1 \leq \sqrt{p_1 s_1}, 1 \leq c_2 \leq \sqrt{p_2 s_2}$, where p_1, p_2 denote the number of features in omics data X_1, X_2 respectively, and s_1, s_2 are the proportions of X_1, X_2 features to be sampled every time. The sparse penalties c_1, c_2 can be re-parametrized as $0 < l_1, l_2 \leq 1$, such that $c_1 = \max\{1, l_1\sqrt{p_1 s_1}\}, c_2 = \max\{1, l_2\sqrt{p_2 s_2}\}$. Large penalty values correspond to more features in each subnetwork, while small penalties correspond to fewer features. Here is the list of parameters need to be specified:

- K: Number of folds in CV.
- $CCcoef$: Optional coefficients, (a, b, c) in Equation 1, for the weighted SmCCA. If CCcoef = NULL (default), then $a = b = c = 1$, and the objective function is the unweighted total sum of all pairwise canonical correlations.
- s_1, s_2: Proportions of feature subsampling from X_1, X_2.
- $numSubsamp$: Number of subsamples.
- P_1P_2: A penalty option matrix for X_1, X_2. Each row of P_1P_2 is a pair of penalty options (l_1, l_2).

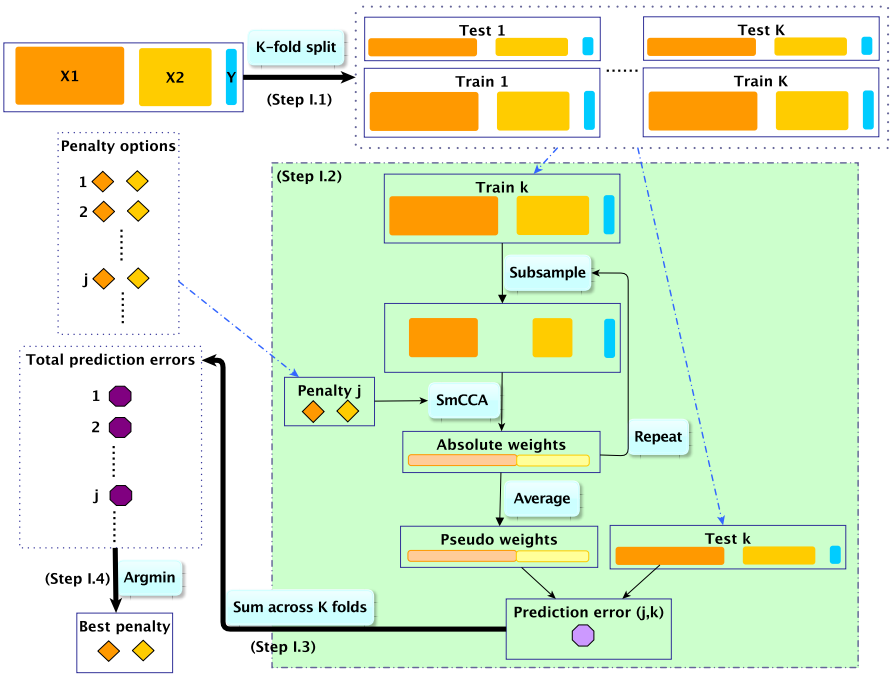

Figure 2: SmCCNet K-fold CV
The best penalty pairs are chosen based on the smallest total prediction error.

```
K <- 3 # Number of folds in K-fold CV.
CCcoef <- NULL # Unweighted version of SmCCNet.
s1 <- 0.7; s2 <- 0.9 # Feature sampling proportions.
numSubsamp <- 500 # Number of subsamples.

# Create sparsity penalty options.
```

```
pen1 <- seq(.05, .3, by = .05)
pen2 <- seq(.05, .3, by = .05)
P1P2 <- expand.grid(pen1, pen2)
# Map (l1, l2) to (c1, c2).
c1 <- sqrt(p1 * s1) * P1P2[ , 1]; c1[c1] <- 1
c2 <- sqrt(p2 * s2) * P1P2[ , 2]; c2[c2 < 1] <- 1
# Based on prior knowledge we may assume that there are at least as many genes
# as miRNAs in each network.
P1P2 <- P1P2[which(c1>c2), ]

# Set a CV directory.
CVDir <- "Example3foldCV/"
dir.create(CVDir)
```

2.2.1 Create test and training data sets.

First, we need to split the data (X_1, X_2, Y) into test and training sets (Figure 2, Step I.1). The data sets will be standardized (centered and scaled) by columns. Since data with zero column variance can not be scaled, we require each column variance to be greater than zero for the test and training sets. Valid test and training sets are saved under a CV directory.

```
set.seed(12345) # Set random seed.

foldIdx <- split(1:n, sample(1:n, K))
for(i in 1:K){
  iIdx <- foldIdx[[i]]
  x1.train <- scale(X1[-iIdx, ])
  x2.train <- scale(X2[-iIdx, ])
  yy.train <- scale(Y[-iIdx, ])
  x1.test <- scale(X1[iIdx, ])
  x2.test <- scale(X2[iIdx, ])
  yy.test <- scale(Y[iIdx, ])

  # Check if standardized data sets are valid.
  if(is.na(min(min(x1.train), min(x2.train), min(yy.train), min(x1.test),
               min(x2.test), min(yy.test)))){
    stop("Invalid scaled data. At least one of the data matrices include a
         column with zero variance.")
  }

  subD <- paste0(CVDir, "CV_", i, "/")
  dir.create(subD)
  save(x1.train, x2.train, yy.train, x1.test, x2.test, yy.test,
       s1, s2, P1P2, p1, p2, numSubsamp, CCcoef,
       file = paste0(subD, "Data.Rdata"))
}
```

2.2.2 Run K-fold CV

For each of the K-fold we compute the prediction error for each penalty pair option (Figure 2, Step I.2). For computational efficiency, we recommend utilizing parallel computing for K-fold CV. As an illustration, we will use the R package **parallel**. The R code below can be easily modified into a *for* loop if multiple cords/threads is not available.

```
library(parallel)

cl <- makeCluster(K, type = "FORK") # Create K parallel threads.
clusterExport(cl = cl, "CVDir") # Pass on variable CVDir to each thread.
parSapply(cl, 1:K, function(CVidx){
    # Reload source code files for each thread.
    source("../R/ModifiedPMA.R")
    source("../R/SmCCNetSource.R")

    # Create a result directory for each thread.
    subD <- paste0(CVDir, "CV_", CVidx, "/")
    load(paste0(subD, "Data.Rdata"))
    dir.create(paste0(subD, "SmCCA/"))

    RhoTrain <- RhoTest <- DeltaCor <- rep(0, nrow(P1P2))
    for(idx in 1:nrow(P1P2)){
        # Consider one pair of sparsity penalties at a time.
        l1 <- P1P2[idx, 1]
        l2 <- P1P2[idx, 2]

        # Run SmCCA on the subsamples (Figure 1, Step II)
        Ws <- getRobustPseudoWeights(x1.train, x2.train, yy.train, l1, l2,
                                     s1, s2, NoTrait = FALSE,
                                     FilterByTrait = FALSE, Bipartite = TRUE,
                                     SubsamplingNum = numSubsamp,
                                     CCcoef = CCcoef)

        # Aggregate pseudo-canonical weights from the subsamples.
        meanW <- rowMeans(Ws)
        v <- meanW[1:p1]
        u <- meanW[p1 + 1:p2]

        # Compute the prediction error for given CV fold and sparsity penalties.
        if(is.null(CCcoef)){CCcoef <- rep(1, 3)} # Unweighted SmCCA.
        rho.train <- cor(x1.train %*% v, x2.train %*% u) * CCcoef[1] +
            cor(x1.train %*% v, yy.train) * CCcoef[2] +
            cor(x2.train %*% u, yy.train) * CCcoef[3]
        rho.test <- cor(x1.test %*% v, x2.test %*% u) * CCcoef[1] +
            cor(x1.test %*% v, yy.test) * CCcoef[2] +
            cor(x2.test %*% u, yy.test) * CCcoef[3]
        RhoTrain[idx] <- round(rho.train, digits = 5)
        RhoTest[idx] <- round(rho.test, digits = 5)
        DeltaCor[idx] <- abs(rho.train - rho.test)

        # Periodically save results in a temporary file.
```

```
            if(idx %% 10 == 0){
                save(P1P2, RhoTrain, RhoTest, DeltaCor, idx,
                     file = paste0(subD, "temp.Rdata"))
            }
        }

        # Record prediction errors for given CV fold and all sparsity penalty
        # options.
        DeltaCor.all <- cbind(P1P2, RhoTrain, RhoTest, DeltaCor)
        colnames(DeltaCor.all) <- c("l1", "l2", "Training CC", "Test CC",
                                    "CC Pred. Error")
        write.csv(DeltaCor.all,
                file = paste0(subD, "SmCCA/PredictionError.csv"))

        # Remove the temporary file.
        system(paste0("rm ", subD, "temp.Rdata"))
        return(CVidx)
        })

    # Close cluster
    stopCluster(cl)
```

2.2.3 Extract penalty pair with the smallest total prediction error

Finally, we extract the total prediction errors (Figure 2, Step I.3) and conclude the best penalty pair as the pair with the smallest error (Figure 2, Step I.4).

```
    # Combine prediction errors from all K folds and compute the total prediction
    # error for each sparsity penalty pair.
    testCC <- predError <- NULL
    for(j in 1:K){
        resultT <- paste0(CVDir, "CV_", j, "/SmCCA/PredictionError.csv")
        dCorT <- read.csv(resultT)[ , -1]
        testCC <- cbind(testCC, abs(dCorT[ , 4]))
        predError <- cbind(predError, dCorT[ , 5])
    }

    S1 <- rowMeans(testCC)
    S2 <- rowMeans(predError)
    T12 <- dCorT[ , -3]; T12[ , 3] <- S1; T12[ , 4] <- S2
    write.csv(T12, file = paste0(CVDir, "TotalPredictionError.csv"))
```

Table 1 shows the total prediction error (CC.Pred.Error) for all penalty options. Note that in this example, we are only including 26 optional penalty pairs, and we require there are at least as many genes as miRNAs in each multi-omics module (i.e., $c_1 \geq c_2$). The fourth column (Test.CC) records the aggregated pseudo canonical correlations for the test data set.

We can visualize the total prediction errors with a contour plot (Figure 3).

Table 1: Total Prediction Error from a 3-fold CV for the synthetic dataset

X	l1	l2	Test.CC	CC.Pred..Error
1	0.10	0.05	1.653380	0.2164767
2	0.15	0.05	1.625990	0.2207548
3	0.20	0.05	1.587710	0.2230805
4	0.25	0.05	1.541390	0.2731577
5	0.30	0.05	1.479960	0.3446770
6	0.10	0.10	1.653170	0.2220495
7	0.15	0.10	1.625370	0.2194567
8	0.20	0.10	1.594123	0.2224069
9	0.25	0.10	1.545233	0.2726456
10	0.30	0.10	1.497150	0.3245182
11	0.10	0.15	1.609257	0.2476096
12	0.15	0.15	1.582933	0.2989744
13	0.20	0.15	1.529037	0.3657604
14	0.25	0.15	1.500757	0.4044987
15	0.30	0.15	1.453507	0.4586189
16	0.15	0.20	1.534180	0.4173106
17	0.20	0.20	1.488540	0.4832922
18	0.25	0.20	1.448607	0.5262836
19	0.30	0.20	1.400920	0.5834178
20	0.15	0.25	1.465967	0.5563104
21	0.20	0.25	1.435200	0.5928120
22	0.25	0.25	1.409617	0.6214640
23	0.30	0.25	1.349440	0.6840775
24	0.20	0.30	1.389823	0.6790695
25	0.25	0.30	1.355420	0.7179057
26	0.30	0.30	1.308063	0.7662127

```
library(plotly)
library(reshape2)

f1 <- list(
  family = "Arial, sans-serif",
  size = 20,
  color = "black"
)
f2 <- list(
  family = "Old Standard TT, serif",
  size = 20,
  color = "black"
)
a <- list(
  title = "l1",
  titlefont = f1,
  showticklabels = TRUE,
  # tickangle = 45,
  tickfont = f2
)
```

```
b <- list(
  title = "l2",
  titlefont = f1,
  showticklabels = TRUE,
  # tickangle = 45,
  tickfont = f2
)
hmelt <- melt(T12[ , -3], id.vars = c("l1", "l2"))
contourPlot <- plot_ly(hmelt, x = ~l1, y = ~l2, z = ~value, type = "contour") %>%
  # add_markers() %>%
  # add_markers(y = ~rev(s)) %>%
  layout(xaxis = a, yaxis = b, showlegend = TRUE, legend = f1)
export(contourPlot, file = paste0(CVDir, "TotalPredictionError.pdf"))
```

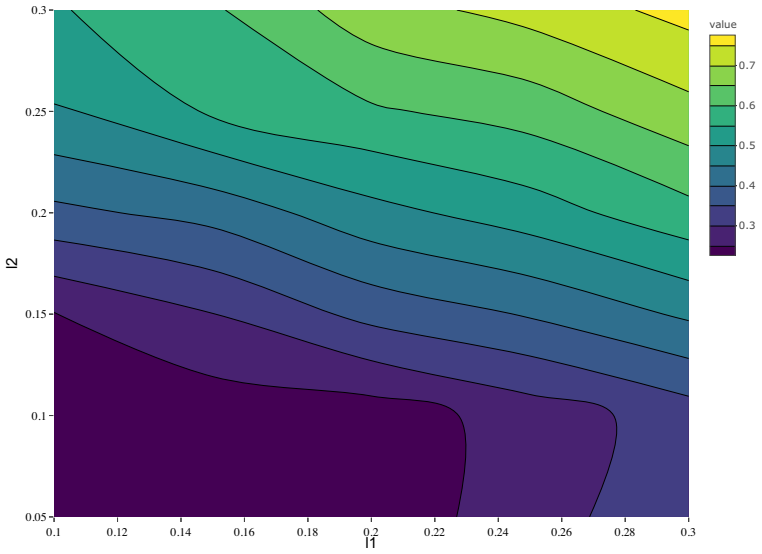

Figure 3: **Total prediction error contour plot**
The x- and y-axes indicate LASSO penalties considered for mRNA and miRNA, respectively. Blue to yellow scale indicates increasing of total prediction error.

For the synthetic data set, the optimal penalty pair that gives the smallest prediction error is $(l_1, l_2) = (0.1, 0.05)$.

```
pen <- which(S2 == min(S2))
l1 <- T12$l1[pen]; l2 <- T12$l2[pen]
print(paste0("Optimal penalty pair (l1, l2): (", l1, ",", l2, ")"))
# [1] "Optimal penalty pair (l1, l2): (0.1,0.05)"
```

2.3 Step II: Integrate two omics data types and a quantitative phenotype

With a pre-selected penalty pair, we apply SmCCA to subsampled features of X_1, X_2 and Y, and repeat the process to generate a robust similarity matrix (Figure 1, Step II). If the penalties were selected through a K-fold CV, the subsampling proportions s_1, s_2 need to be consistent with what was used in the CV. As for the number of subsamples, a larger number of subsamples leads to more accurate results, while a smaller number of subsamples is faster computationally. We use 500 in this example. In general, we recommend to subsample 1000 times or more.

```
Ws <- getRobustPseudoWeights(X1, X2, Y.scaled, l1, l2, s1, s2, NoTrait = FALSE,
                             FilterByTrait = FALSE, Bipartite = TRUE,
                             SubsamplingNum = numSubsamp, CCcoef = CCcoef)
Abar <- getAbar(Ws, AbarLabel)
```

2.4 Step III: Obtain multi-omics modules and plot subnetworks

From the similarity matrix obtained in the last step, we can get multi-omics modules by applying a hierarchical tree cutting and plotting the reconstructed networks (Figure 1). The edge signs are recovered from pairwise feature correlations.

```
Modules <- getMultiOmicsModules(Abar, p1)
save(Ws, Abar, Modules, file = paste0(CVDir, "SmCCNetWeights.Rdata"))
```

The trimmed module (edge cut = 0.1) is shown below. If a full module does not contain any edge that passes the cut threshold, a message "No edge passes threshold" will be produced. To see all complete module, set **edgeCut = 0**.

```
bigCor <- cor(cbind(X1, X2))
edgeCut <- 0.1
for(idx in 1:length(Modules)){
    filename <- paste0(CVDir, "Net_", idx, ".pdf")
    plotMultiOmicsNetwork(Abar = Abar, CorrMatrix = bigCor,
                          multiOmicsModule = Modules, ModuleIdx = idx, P1 = p1,
                          EdgeCut = edgeCut, FeatureLabel = AbarLabel,
                          SaveFile = filename)
}
```

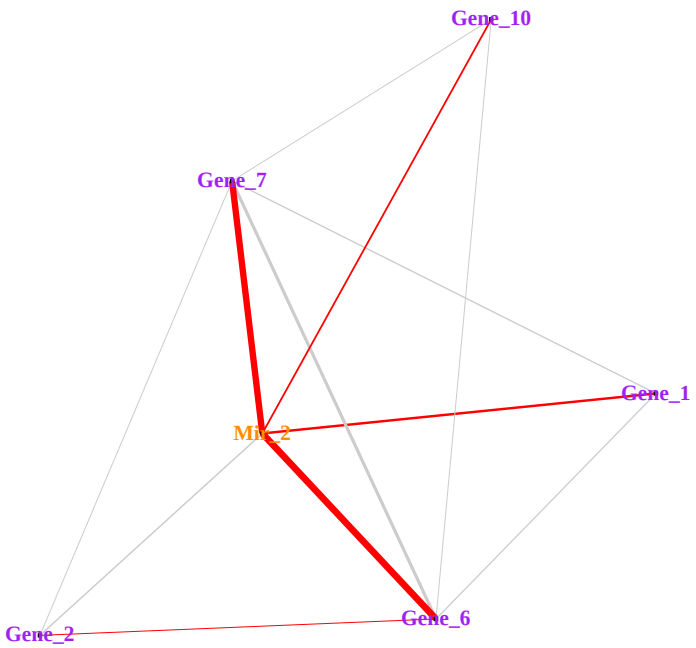

Figure 4: Trimmed module 1
The strength of the node connections is indicated by the thickness of edges. Red edges and gray edges are for negative and positive connections, respectively.

3 Alternative canonical correlation analysis (CCA) methods

The function **getRobustPseudoWeights()** includes two other CCA methods (sparse supervised CCA (SsCCA) and sparse CCA (SCCA)), both of which are also coupled with subsampling scheme for more robust results. Users should pick the appropriate CCA method according to their studies.

3.1 SsCCA

SsCCA prioritizes omics features according to the correlation to the phenotype. This approach can be useful when the phenotype is not quantitative. To use SsCCA, set **NoTrait = FALSE** and **FilterByTrait = TRUE**.

3.2 SCCA

If the study purpose is to integrate two omics data type without any phenotype information, one can choose SCCA. To use SCCA, set **NoTrait = TRUE**.

4 Session info

```
sessionInfo()
## R version 3.4.3 (2017-11-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.4
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] igraph_1.2.1    Matrix_1.2-14   pbapply_1.3-4   PMA_1.0.11
## [5] impute_1.52.0   BiocStyle_2.6.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.17    bookdown_0.7    lattice_0.20-35 digest_0.6.15
##  [5] rprojroot_1.3-2 grid_3.4.3      backports_1.1.2 magrittr_1.5
##  [9] evaluate_0.10.1 stringi_1.2.2   rmarkdown_1.9   tools_3.4.3
## [13] stringr_1.3.1   parallel_3.4.3  xfun_0.1        yaml_2.1.19
## [17] compiler_3.4.3  pkgconfig_2.0.1 htmltools_0.3.6 knitr_1.20
warnings()
## NULL
```

5 References

Shi, W.J., Y. Zhuang, P.H. Russell, B.D. Hobbs, M.M. Parker,P.J. Castaldi, P. Rudra, B. Vestal, C.P. Hersh, L.M. Saba, and K. Kechris, "Unsupervised Discovery of PhenotypeSpecific Multi-Omics Networks." (*Submitted*)