integIRTy: a method to identify altered genes in cancer accounting for multiple mechanisms of regulation using Item Response Theory

Pan Tong and Kevin R Coombes

May 6, 2019

Contents

List of Figures

List of Tables

1 Introduction

`integIRTy` is an R package that integrates multiple modalities of high throughput assays using Item Response Theory (IRT) [Tong and Coombes, 2012]. The goal is to identify genes that are altered in cancer either marginally in individual assay or jointly across different assays. `integIRTy` is able to systematically integrate across assay types while automatically adjusting for the heterogeneity among different platforms and different samples. When applied to a single assay, `integIRTy` is more robust and reliable than conventional methods such as student's t-test or Wilcoxon rank-sum test. When applied to integrate multiple assays, `integIRTy` can identify novel altered genes that cannot be found by looking at individual assays separately.

2 A Quick Example

`integIRTy` provides two top-level interfaces to the user. One is `intIRTeasyRun`, which accepts dichotomized matrices prepared by the user; the other is `intIRTeasyRunFromRaw`, which accepts original data and uses the dichotomization methods proposed in [Tong and Coombes, 2012] to transform the data. Both interfaces perform a series of computations: fitting IRT models in individual assays, estimating latent traits from individual assay, inferring latent traits from integrated data, and performing permutations to assess statistical significance.

We will present a simple example to illustrate how to use `integIRTy`. We begin with the `intIRTeasyRunFromRaw` function. The usage of `intIRTeasyRun` is similar. Later, we will break down the procedures step by step.

As usual, the first step is to load the package from the library.

```
> library(integIRTy)
```

We use a subset of TCGA ovarian cancer data [Network et al., 2011] (shipped along with the `integIRTy` package). We use trhe `data` function to load expression, methylation, and copy number data for both tumor and normal samples.

```
> data(OV)
> ls()
```

```
[1] "CN_N"    "CN_T"    "Expr_N"  "Expr_T"  "Methy_N" "Methy_T"
```

The `intIRTeasyRunFromRaw` function requires two lists of matrices corresponding to tumor and normal samples. The lists must contain data from matching assays in the same order. The rows of each matrix should also be in the same order, corresponding to the same set of genes. The two lists are prepared below.

```
> controlList <- list(Expr_N, Methy_N, CN_N)
> tumorList <- list(Expr_T, Methy_T, CN_T)
```

To avoid repeated computation, we define a function that detects if an object is already present:

```
> testObject <- function(object) {
   exists(as.character(substitute(object)))
 }
```

Now we can call `intIRTeasyRunFromRaw` to integrate the data. Note that the assay type is specified in the order of the two lists so that the program can choose the right dichotomization methods. We also add gene sampling for significance assessment.

```
> if(!testObject(runFromRaw)) {
   runFromRaw <- intIRTeasyRunFromRaw(platforms=tumorList,
                                      platformsCtr=controlList,
                                      assayType=c("Expr", "Methy", "CN"),
                                      permutationMethod="gene sampling")
 }
```

```
Performing data dichtomization for each platform
Performing easyRun on dichtomized data
Performing ltm fit for each platform
Performing latent trait estimation for each platform
Calculating permuted latent trait for each platform
```

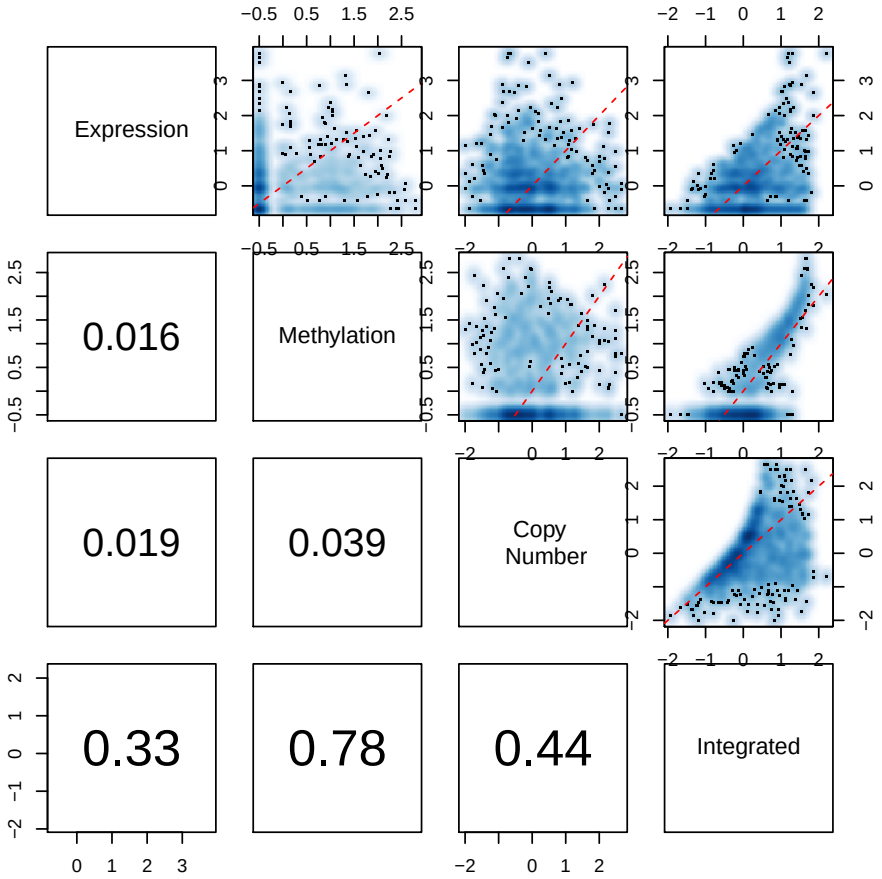
```
> class(runFromRaw)

[1] "list"

> attributes(runFromRaw)

$names
[1] "fits"              "estimatedScoreMat" "permutedScoreMat"
[4] "dscrmnList"        "dffcltList"        "gussngList"
```

`intIRTeasyRunFromRaw` returns a list of the following elements:

- `fits`: a list of model fits for each platform as returned by `fitOnSinglePlat` function; specifically, these include
  - `fit`: An object returned by calling ltm package. Item parameters and other auxillary inforamtion (i.e. loglikelihood, convergence, Hessian) can be accessed from this object. For more details, please refer to ltm package
  - `model`: The model type
  - `guessing`: The guessing parameter
  - `sampleIndices`: The sample indices used in the model
  - `geneIndices`: The gene indices used in the model
- `estimatedScoreMat`: A matrix of estimated latent traits. The first several columns correspond to the individual assays; the last column represents the integrated latent trait with all data.
- `permutedScoreMat`: A matrix of latent trait estimates after permuting the binary matrix within columns. This is only available if addPermutedScore is set to TRUE. The first several columns correspond to the individual assays; the last column represents the integrated data.
- `dscrmnList`: A list of discrimination parameters. Each element contains all of the discrimination parameters as a vector for each assay. The last element contains the discrimination parameters for the integrated data which is formed by combining discrimination parameters from each assay sequentially.
- `dffcltList`: Same format as dscrmnList except it contains difficulty parameter.
- `gussngList`: Same format as dscrmnList except it contains guessing parameter. Be default, this is just all 0's.
- `permutedScoreMatWithLabelPerm`: A matrix of latent trait estimates using sample label permutation. This is only available if permutationMethod='sample label permutation' is used. The first several columns correspond to the individual assays; the last column represents the integrated data.

Below, we just extract the estimated latent traits for each assay and visualize them by matrix plot. The upper diagonal panels show the pairwise smoothed scatter where dark clouds indicate the density. The lower panels show the correlation of the latent traits.

```
> runFromRaw_ScoreMat <- runFromRaw$estimatedScoreMat
> pairs(runFromRaw_ScoreMat, lower.panel=panel.cor, upper.panel=panel.smooth,
        labels=c('Expression', 'Methylation', 'Copy \n Number', 'Integrated'),
        cex.labels=1.2, gap=1.7)
```



3 Building The Pipeline Step By Step

Usually the `intIRTeasyRunFromRaw` function is enough for most applications. In case of assays rather than expression, methylation and copy number is present, the user might need to build the pipeline themselves. Also, we allow the user to define their own dichotomization methods. Below we show step by step how to build the pipeline for data integration.

3.1 Data Dichotomization

We implement 3 dichotomization methods proposed in [Tong and Coombes, 2012]. The dichotomize() function is a wrapper for each method. Many options can be specified. For details, please refer to the package documentation.

```
> if(!testObject(binDat_CN)){
    binDat_expr <- dichotomize(Expr_T, Expr_N, assayType='Expr')
```

```
   binDat_methy <- dichotomize(Methy_T, Methy_N, assayType='Methy')
   binDat_CN <- dichotomize(CN_T, CN_N, assayType='CN')
 }
```

3.2 Fit IRT Model On Each Assay

With dichotomized data, we can fit an IRT model on each assay. Three IRT models are implemented: the Rasch model where all item discrination are set to 1; the constrained 2PL model where all item discrimation are set to be equal but not necessarily 1; the 2PL model where no constraint is put on the item difficulty and discrimination parameter. By default, the 2PL model will be used.

```
> if(!testObject(fit2PL_CN)){
   fit2PL_Expr <- fitOnSinglePlat(binDat_expr, model=3)
   fit2PL_Methy <- fitOnSinglePlat(binDat_methy, model=3)
   fit2PL_CN <- fitOnSinglePlat(binDat_CN, model=3)
 }
```

3.3 Estimating Latent Traits

Result returned by fitOnSinglePlat() contains item parameters, which can be used to estimate the latent traits through the computeAbility() function.

First we extract the item parameters:

```
> dffclt_expr <- coef(fit2PL_Expr$fit)[, 'Dffclt']
> dscrmn_expr <- coef(fit2PL_Expr$fit)[, 'Dscrmn']
> dffclt_methy <- coef(fit2PL_Methy$fit)[, 'Dffclt']
> dscrmn_methy <- coef(fit2PL_Methy$fit)[, 'Dscrmn']
> dffclt_CN <- coef(fit2PL_CN$fit)[, 'Dffclt']
> dscrmn_CN <- coef(fit2PL_CN$fit)[, 'Dscrmn']
```

Then, latent traits from each assay can be estimated separately.

```
> if(!testObject(score_expr)){
   score_expr <- computeAbility(binDat_expr, dscrmn=dscrmn_expr,
                                dffclt=dffclt_expr)
   score_methy <- computeAbility(binDat_methy, dscrmn=dscrmn_methy,
                                 dffclt=dffclt_methy)
   score_CN <- computeAbility(binDat_CN, dscrmn=dscrmn_CN,
                              dffclt=dffclt_CN)
 }
```

3.4 The Integrated Latent Trait

The integrated latent trait can be estimated similarby by combining the items into a larger test:

```
> if(!testObject(score_integrated)){
   score_integrated <- computeAbility(respMat=cbind(binDat_expr,
                                        binDat_methy, binDat_CN),
                                      dscrmn=c(dscrmn_expr, dscrmn_methy, dscrmn_CN),
                                      dffclt=c(dffclt_expr, dffclt_methy, dffclt_CN))
 }
```

By building the pipeline step by step, we get the same result as using the intIRTeasyRunFromRaw() function:

```
> all(score_integrated==runFromRaw_ScoreMat[, 4])
```

```
[1] TRUE
```

3.5 The intIRTeasyRun() Function

The intIRTeasyRun() function is a pipeline for binary input. It can be used as follows:

```
> # not run to allow vignette to complete more quickly
> runFromBinary <- intIRTeasyRun(platforms=list(binDat_expr,
                                  binDat_methy, binDat_CN))
```

4 Parallelizing `integIRTy`

We add an option for parallel computing to speed up the computation. Parallel computing uses `foreach` as the backend while workers are requested by `doParallel`. The parallelism can happen when fitting serveral IRT models (due to multiple assay types to integrate) or estimating the latent traits or performing permutation to assess statistical significance.

The following functions can be paralleled which can be controlled by the *parallel* option: intIRTeasyRun(), intIRTeasyRunFromRaw(), calculatePermutedScoreByGeneSampling(), computeAbility() and dichotomizeExpr().

Enabling parallelism is quite simple for the user. The first step involves requesting workers. The second step is to set the option *parallel* to TRUE in a specified function. By default, all parallel options are set to be FALSE.

Several cluster types can be requested including MPI, SOCK, PVM and NWS. For example, on a windows machine with multiple cores, the following code requests 3 workers through SOCK:

```
library(doParallel)
cl <- makeCluster(3, type = "SOCK")
registerDoParallel(cl)
```

Note that the above command also works on Linux servers. However, it requests master nodes when run within R.

For Linux servers, especially linux clusters with MPI installed, we can request more workers as:

```
registerDoParallel(makeCluster(30))
```

Usually it's not a good practice to request the master nodes on the cluster. To avoid this, one can use qsub command to submit a PBS job using the R script written without further modification.

We proceed with the 3 nodes requested and call the intIRTeasyRun() to illustrate how to use the parallel computing:

```
runFromBinary <- intIRTeasyRun(platforms=list(binDat_expr,
                                    binDat_methy, binDat_CN),
                                addPermutedScore=TRUE, parallel=TRUE)
```

Now we close the connection to the workers.

```
stopCluster(cl)
```

5 File Location and Session Info

```
> getwd()

[1] "/tmp/Rtmp63OhDk/Rbuild1d3a75e5b704/integIRTy/vignettes"

> sessionInfo()

R version 3.5.3 (2019-03-11)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Debian GNU/Linux buster/sid

Matrix products: default
BLAS: /srv/R/R-patched/build.19-04-05/lib/libRblas.so
LAPACK: /srv/R/R-patched/build.19-04-05/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] integIRTy_1.0.6   doParallel_1.0.14 iterators_1.0.11  foreach_1.5.1
[5] ltm_1.1-1         polycor_0.7-10    msm_1.6.7         MASS_7.3-51.4

loaded via a namespace (and not attached):
 [1] mclust_5.4.3       lattice_0.20-38    codetools_0.2-16   mvtnorm_1.0-10
 [5] grid_3.5.3         KernSmooth_2.23-15 Matrix_1.2-17      splines_3.5.3
 [9] tools_3.5.3        abind_1.4-7        survival_2.44-1.1  compiler_3.5.3
[13] expm_0.999-4
```

References

Cancer Genome Atlas Research Network et al. Integrated genomic analyses of ovarian carcinoma. *Nature*, 474(7353):609–615, 2011.

Pan Tong and Kevin R Coombes. integirty: a method to identify genes altered in cancer by accounting for multiple mechanisms of regulation using item response theory. *Bioinformatics*, 28(22):2861–2869, 2012.