Gene Set Enrichment Analysis with R and Bioconductor

Zuguang Gu

2022-11-26

About

This is a book on gene set enrichment analysis.

Introduction

...

Gene Set Databases

Placeholder

Overview

Gene Ontology

The GO database

The GO.db package

use `GO.db` as a database object

Objects in the GO.db package

The `GOTERM` object

Term relations in GO.db

Graph analysis of GO tree

Link GO terms to genes

The org.Hs.eg.db package

OrgDb objects for other organisms

GO gene sets from BioMart

Gene set format in R

KEGG pathways

KEGG API

The KEGGREST package

The MSigDB database

Reactome pathways

UniProt keywords

Gene ID mapping

Generate gene sets for other organisms by mapping to orthologues

Summary

Over-Representation Analysis

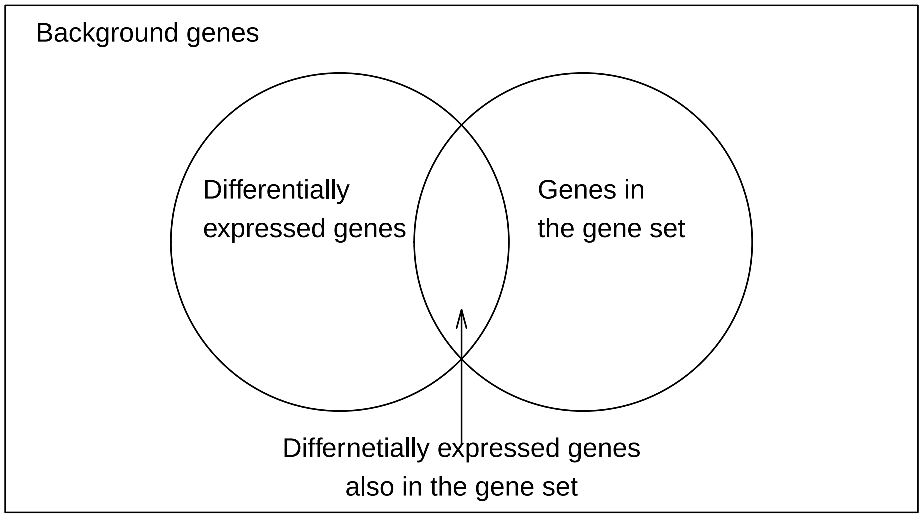
Overview

Over-representation analysis (ORA) uses a simplified model for gene set enrichment anlaysis. It directly works on the numbers of genes in different categories, i.e., whether genes are in the list of interest and whether genes are in the gene set. Because of its simplicity, ORA is the mostly used method for gene set enrichment analysis. In this chapter, I will introduce different statistical tests for ORA and the implementations in R. I will also point out the limitations of ORA which users need to pay attention to when they apply ORA on their datasets.

What is over-representation?

In many cases, ORA is applied to a list of differentially expressed (DE) genes, thus, to simplify the description of the text in this chapter, I use DE genes to represent the list of genes of interest. But keep in mind that the list of DE genes is just a special case of the gene list. In applications, it can be any type of gene lists.

For an experiment which meansures gene expression in several conditions, the total genes can be summarized with a Venn diagram in Figure x. In the Venn diagram, the background genes are the total genes in the genome or has measured in the experiment. The left circle corresponds to the total DE genes, and the right circle corresponds to the total genes in a gene set. A natural feeling is if the number of DE genes also in the gene set is large enough, we can say the existance of DE genes in the gene set are more than expected, then we can conclude DE genes are over-represented in the gene sets. We can also say it in another way around, the gene set is over-represented in the DE gene lists.


Background genes
Differentially expressed genes
Genes in the gene set
Differnetially expressed genes also in the gene set

There are several ways to quantitatively measure such over-representation. As we already have the observed number of DE genes in the gene set which corresponds to the intersection between two circles in the Venn diagram, the next step is to find the "expected" number of DE genes in the gene set. Let's denote the total number of background genes as n, the number of DE genes as n_{de}, the number of genes in the gene set as n_{set} and the number of DE genes in the gene set as k_{obs}. We first calculate the following two p-values

$$p_{\mathrm{de}} = n_{\mathrm{de}}/n$$

$$p_{\mathrm{set}} = n_{\mathrm{set}}/n$$

where p_{de} and p_{set} can be thought as the probabilities of a gene being DE and being in the gene set. For the "expected" scenario, we assume the two events: "the gene is a DE gene" and "the gene is in the gene set," are independent, or in other words, there is no over-representation or under-representation between the two types of gene lists, then the expected number of DE genes in the gene set denoted as k_{exp} can be calculated by directly multiplying p_{de} and p_{set}.

$$k_{\mathrm{exp}} = p_{\mathrm{de}} p_{\mathrm{set}} n$$

We can simply compare k_{obs} and k_{exp} by their ratio denoted as r:

$$r = \frac{k_{\mathrm{obs}}}{k_{\mathrm{exp}}} = \frac{k_{\mathrm{obs}}}{p_{\mathrm{de}} p_{\mathrm{set}} n} = \frac{k_{\mathrm{obs}} n}{n_{\mathrm{de}} n_{\mathrm{set}}}$$

If $r > 1$, there are more DE genes than expected in the gene set, then we can conclude the two attributes of being a DE gene and being in the gene set have a positive association, or we can say there is over-representation between the two types of gene lists.

Less oftenly used, when $r < 1$, we can say there is an under-representation between DE genes and the gene set.

In the previous text, we compared the observed and expected numbers of DE genes in the gene set to evaluate the over-representation. Next we look at the problem from a slightly different aspect. We treat p_{de} as the "background probability" of a gene being DE, because it is calculated against the total number of genes in the background. Next we calculate the probability of a genes being DE but only in the gene set, which we can treat as the "foreground probability." Let's denote it as $p_{\mathrm{de}}^{\mathrm{fore}}$ and it can be calculated as

$$p_{\mathrm{de}}^{\mathrm{fore}} = k_{\mathrm{obs}}/n_{\mathrm{set}}.$$

Then we can compare the background and foreground probability. If the foreground probability is higher than background probability, i.e., $p_{\mathrm{de}}^{\mathrm{fore}} > p_{\mathrm{de}}$ or $p_{\mathrm{de}}^{\mathrm{fore}}/p_{\mathrm{de}} > 1$, we can conclude DE genes are over-represented in the gene set. It is easy to see the following relation:

$$\frac{p_{\mathrm{de}}^{\mathrm{fore}}}{p_{\mathrm{de}}} = \frac{k_{\mathrm{obs}} n}{n_{\mathrm{de}} n_{\mathrm{set}}} = r.$$

Similarly, we can also treat p_{set} as the "background probability" of a gene being in the gene set, and we calculate the "foreground probability" denotated as $p_{\mathrm{set}}^{\mathrm{fore}}$ of a gene being in the gene set, but only restricted in the DE genes. It is easy to see the following relation.

$$\frac{p_{\mathrm{de}}^{\mathrm{fore}}}{p_{\mathrm{de}}} = \frac{p_{\mathrm{set}}^{\mathrm{fore}}}{p_{\mathrm{set}}}$$

The ratio r can be used to quantitatively measure whether there is an over-representation between DE genes and the gene set, where a higher value of r implies there is stronger over-representation. Under the statistical procedures, we need to calculate a p-value for the over-representation to assign a "signficance level" for the enrichment. Although r is able to measure the over-representation, its distribution in analytical form is hard to obtain. In the next section, we introduce specific distributions or statistical tests for calculating p-values under the ORA framework.

Statistical tests

Standard denotations in ORA

In literature, the problem of ORA is often formulated into the following 2x2 contigency table. In the table, rows are split into whether genes are DE or not, and columns are split into whether the genes are in the gene set or not. Values in the table are the number of genes in each category. n corresponds to the total number of background genes.

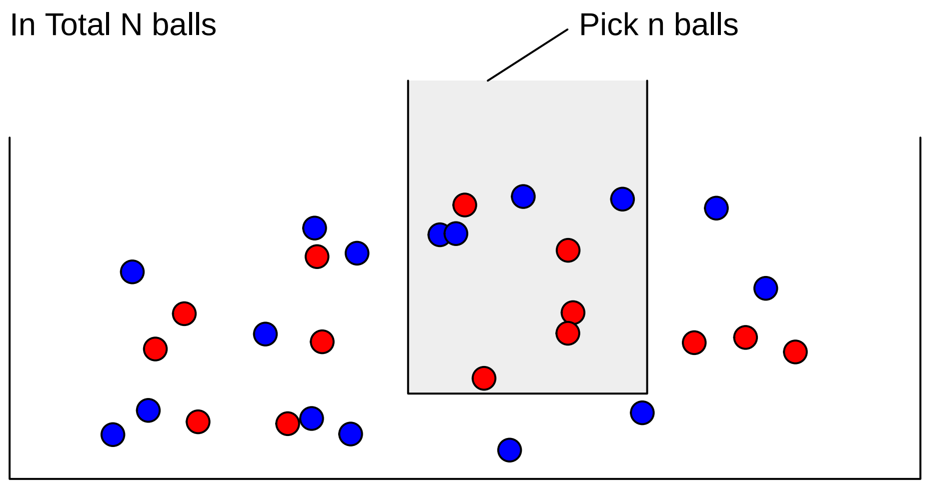
	In the gene set	Not in the gene set	Total
DE genes	n_{11}	n_{12}	n_{1+}
Non-DE genes	n_{21}	n_{22}	n_{2+}
Total	n_{+1}	n_{+2}	n

The distributions or statistical tests introduced in the section are all based on this table. If you are confused by the text or the mathematical denotations in later sections, you can always come back and refer to this table.

Hypergeometric distribution

Hypergeometric distribution is a probability distribution for discrete events. I will first briefly introduce the form of hypergenometric distribution as introduced in other textbooks. Later I will map to ORA.

The hypergeometric distribution comes from the following question. Assume there are N balls in a bag, where there are K red balls, and $N - K$ blue balls. If randomly picking n balls from there, what is the probability of having k red balls out of n balls?



We assume each ball can be picked independently and with equal probability, then, we can have the following numbers:

- Total number of ways of picking n balls from the bag: $\binom{N}{n}$.
- Number of ways of pick k red balls from K red balls: $\binom{K}{k}$.
- Number of ways of picking $n-k$ blue balls from $N-K$ blue balls: $\binom{N-K}{n-k}$.

Since picking red balls and picking blue balls are independent, the number of ways of picking n balls which contain k red and $n-k$ blue balls is $\binom{K}{k}\binom{N-K}{n-k}$. Then the probability denoted as P from the question is calculated as:

$$P = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}},$$

where in the denominator is the number of ways of picking n balls without distinguishing whether they are red or blue.

If N, K and n are all fixed values, the number of red balls that are picked can be denoted as a random variable X. Then X follows the hypergenometric distribution with parameters N, K and n, written as $X \sim \text{Hyper}(N, K, n)$.

ORA basically has the same form as the ball problem. We just need to change the description while the statistical model is unchanged. Let's change the original question to swtich to genes:

Assume there are N *balls* (genes) in *a bag* (an experiment), where there are K *red balls* (DE genes), and $N-K$ *blue balls* (non-DE genes). If randomly picking n *balls* (the amount of genes with the same size as the gene set) from there, what is the probability of having k *red balls out of* n *balls* (DE genes in the gene set)?

Let's also map the original denotations to those used in Table x.

Total balls	N	Total genes	n
Red balls	K	DE genes	n_{1+}
Blue balls	$N-K$	non-DE genes	n_{2+}
Balls to pick	n	genes in gene set	n_{+1}
Red balls that are picked	k	DE genes in the gene set	n_{11}
Blue balls that are picked	$n-k$	non-DE genes in the gene set	n_{21}

Specifically for ORA, we denote the number of DE genes in the gene set as a random variable X, then

$$X \sim \text{Hyper}(n, n_{1+}, n_{+1}).$$

We can also transite the question in another way around. We can map genes in the gene set to red balls and total DE genes as the balls to pick. In this case, $X \sim \text{Hyper}(n, n_{+1}, n_{1+})$. The two forms actually are identical.

If there is no relation between whether genes are DE and genes are in the gene set, it is expected that n_{11} is not too large, while if the observed value of n_{11} is large enough, we could conclude that it is not likely that the two attributes of genes are independent, where DE genes preferably exist in the gene set. Thus, we can use the probability of obtaining number of DE genes in the gene set equal to or larger than n_11 denoted as $Pr(X \geq n_{11})$ to measure the unlikeness. If obtaining a small p-value with the observed value n_{11}, we would say the null hypothesis.

$$\text{Pr}(X >= n_{11}) = \sum_{x \in n_{11}, \ldots, \min n_{+1}, n_{1+}} \text{Pr}(X = x)$$

Binomial distribution

The hypergenometric distribution can be approximated by the Binomial distribution. Now we need to change the problem a little bit and we only look at the genes in the gene set, In a gene set with n_{+1} genes, each gene can be a DE genes with probabilty p_{de}, which is estimated as

$$p_{\text{de}} = n_{1+}/n$$

The probelm can be thought as we tend to pick all n_{+1} genes from the gene set, but each gene only has a successfull rate to be picked. The probability of successfully picked is p_{de}, which means it has a unsuccessful rate of $1 - p_{\text{de}}$. Then the probability of successfully picking n_{11} genes can be calcualted as

$$P_{\text{Binom}} = \binom{n_{+1}}{n_{11}} p_{\text{de}}^{n_{11}} (1 - p_{\text{de}})^{n_{+1} - n_{11}}$$

In the formula, the binomial coefficient term correspond to the total number of ways to pick n_{11} genes from n_{+1} genes, among them, the n_{11} genes are all successful, which is the second term which contains assumultive production of n_{11} p-values. We also know the other $n_{+1} - n_{11}$ are not picked and they should also be multiplied.

If again, denote the number of DE genes in the gene set as a random variable X, then $X \sim \text{Binom}(n_{+1}, p_{\text{de}})$. And p-value is calculated as $Pr(X \geq n_{11})$.

We can also do in other other direction, we look at DE genes and each gene has a probability being in the gene set.

$$p_{\text{set}} = n_{+1}/n$$

And approximatedly, $X \sim \text{Binom}(n_{1+}, p_{\text{set}})$. But note, the two distributions are not identical.

z-test

Let's look back Table xx, if the two events "genes are DE" and "gene is in the gene set" are independent, then the probability of a gene being DE in the gene set should be identical to the probability of a gene being DE not in the gene set, which are the following two probabilities. They correspond to the first two columns in Table x.

$$p_1 = n_{11}/n_{+1}$$

$$P_2 = n_{12}/n_{+2}$$

For genes in the gene set, number of DE genes actually follow a Binomilal distribution $\text{Binom}(n_{+1}, p_1)$, and for genes not in the gene set, number of DE genes also follow a Binomial distribution: $\text{Binom}(n_{+2}, p_2)$. Now the problem is to test whether the two Binomial distribution is idential. The null hypothesis is $p_1 = p_2$, then when n_{+1} and n_{+2} are large, the following z-score:

$$z = \frac{p_1 - p_2}{\sqrt{p(1-p)}\sqrt{\frac{1}{n_{+1}} + \frac{1}{n_{+2}}}}$$

where $p = \frac{n_{11}+n_{12}}{n_{+1}+n_{+2}} = \frac{n_{1+}}{n}$. z follows a standard normal distribution $N(0,1)$. And the p-value is calculated as

$$P(X > z) + P(x < -z)$$

If we only want to test over-representation, p-value can be calculated as

$$P(x > z)$$

It easy to see the test is idential is p_1 and p_2 are calculated as the probabilities of a genes being in the gene set, for DE genes and non-DE genes.

Fisher's exact test

Fisher's exact test is used to test 2x2 contigency table. If the marginal values on the contigency table are fixed.

Chi-square test

Pearson's Chi-square test can also be applied to test whether there are dependencies on categorized data. The Chi-square statistic measures the relative sum of squares of the difference between observed values and expected in categories:

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed value in category i and E_i is the expected value in category i.

If we apply it to the 2x2 contigency table, actually the data is splt into four non-intersected categories, DE/set, DE/not in set, non-DE/in set and non-DE/not in set. Let's take the first category, e.g. genes are DE and also in the set, the observed value is simply n_{11}. The expected value is $np_{1+}p_{+1}$ which assumes whether a gene is DE and whether a gene is in the set is independent, where $p_{1+} = n_{1+}/n$ and $p_{+1} = n_{+1}/n$. And if we write all four categories, we have

$$\chi^2 = \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{(n_{ij} - np_{i+}p_{+j})^2}{np_{i+}p_{+j}}$$

where $p_{i+} = n_{i+}/n$ and $p_{+j} = n_{+j}/n$.

If n is large, the χ^2 statistic can be approximated as followinng χ^2 distribution with degree of freedom of 1.

Simply calcus reveals that the value of the χ^2 is the square of the z-statistic.

Calculate in R

The four distributions of statistical tests are very basic in statistics and there are already functions for calculating p-values.

To demonstrate the different distributions and tests, we use a dataset from EBI Altas database with accession ID E-GEOD-101794. I only take a subset of results from the dataset which compares gene expression between Crohn's disease and non inflammatory bowel disease. The data is from an RNASeq experiment and **DESeq2** was applied for differnetial expression analysis. In this secton, I take out significant DE genes by setting FDR < 0.05. The original gene ID type is Ensembl ID, since Entrez ID is the primary ID type in most of the standard Bioconductor annotation packages, I will first convert genes from Ensembl IDs to Entrez IDs. Note this conversion is not one to one, genes may lost if there is no mapping. After the conversion, there are 948 genes remains for the analysis.

```
lt = readRDS("data/EBI_E-GEOD-101794_expression.rds")
diff_table = lt$diff_table
diff_genes = rownames(diff_table)[diff_table$p_adjust < 0.05]
library(org.Hs.eg.db)
diff_genes = mapIds(org.Hs.eg.db, keys = diff_genes, keytype = "ENSEMBL", column = "ENTREZID")
diff_genes = diff_genes[!is.na(diff_genes)]
length(diff_genes)
# [1] 948
```

We test whether DE genes are enriched in the GO gene set "GO:0000165" (MAPK cascade). The genes in the gene set can be obtained by directly subsetting the object `org.Hs.egGO2ALLEGS`. It is very important the gene set is also in the same gene ID type as DE genes.

```
gene_set = unique(org.Hs.egGO2ALLEGS[["GO:0000165"]])
```

ORA analysis also depends on the background genes. There are different selections of background genes, and different selection will affect the numbers in the 2x2 contigency table and eventually affect the final p-value. In later sections of this chapter, we will discuss the effect of choose different background. Here for the following example, I will use all protein coding genes as background genes. The object `org.Hs.egGENETYPE` contains gene types for all genes. I first use the general function `toTable()` to convert the internal data object to a data frame.

```
gene_type_table = toTable(org.Hs.egGENETYPE)
head(gene_type_table)
#   gene_id      gene_type
# 1       1 protein-coding
# 2       2 protein-coding
# 3       3         pseudo
# 4       9 protein-coding
# 5      10 protein-coding
# 6      11         pseudo
```

Then select genes of which the correpsonding type is "protein-coding."

```
universe = gene_type_table$gene_id[gene_type_table$gene_type == "protein-coding"]
length(universe)
# [1] 20598
```

`GENETYPE` is also a valid keytype for directly querying `org.Hs.eg.db` database,

```
select(org.Hs.eg.db, key = "protein-coding", keytype = "GENETYPE", column = "ENTREZID")
```

The next step is optional if you can make sure universe genes covers all DE genes and all genes in the gene set.

```
diff_genes = intersect(diff_genes, universe)
gene_set = intersect(gene_set, universe)
```

Now we have vectors of DE genes, gene set, and universe genes. We can calculate values in the 2x2 contigency table. We first calculate values for n_{1+}, n_{+1}, n and n_{11}. All other values can be easily calculated based on these four values.

```
length(diff_genes)                       # n_1+
# [1] 842
length(gene_set)                        # n_+1
# [1] 764
length(universe)                        # n
# [1] 20598
length(intersect(diff_genes, gene_set))  # n_11
# [1] 58
```

The 2x2 contigency table for the GO gene sets is as follows. Next I will demonstrate how to calcualte p-values from various statistical test in prevous sections.

	In the gene set	Not in the gene set	Total
DE	58	779	837
Noe DE	706	19055	19761
Total	764	19834	20598

The function `phyper()` calculates the p-value from hypergeometric distribution. The usage of `phyer()` is:

```
phyper(q, m, n, k, lower.tail = FALSE)
```

In ORA, `q` is the number of differential genes in the gene set, `m` is the size of the gene set, `n` is the number of genes not in the gene set, `k` is the number of differnetial genes. One thing that should be careful is the p-value calculated by `phyper()` corresponds to the probabilty of $Pr(X > q)$ which does not inlcude the p-value when $X = q$, thus, to calculate the p-value which also includes the scenario of $X = q$, we need to slightly modify the previous use of `phyper()` by substract 1 so that $Pr(X > q - 1) = Pr(X \geq q)$.

By default in `hyper()`, the argumnet `lower.tail` is set to `TRUE` which calculates $Pr(X \leq q)$, we need to explicitely ..

```
phyper(q - 1, m, n, k, lower.tail = FALSE)
```

Now we fill the values in the contigency table to the function call.

```
phyper(58 - 1, 764, 19834, 837, lower.tail = FALSE)
# [1] 3.75562e-06
```

Readers may ask is `phyper(q, m, n, k, lower.tail = TRUE)` always identical to `1 - phyper(q, m, n, k, lower.tail = FALSE)`. For very small p-values, the second one will return zero because the . . .

```
phyper(100, 764, 19834, 837, lower.tail = FALSE)
# [1] 3.194366e-26
1 - phyper(100, 764, 19834, 837, lower.tail = TRUE)
# [1] 0
```

So to get a more meaningful p-value, it is suggested to use `lower.table = FALSE`.

since the hypergeometric can also be calcualted from the other dimension, `m` can be the number of differneital genes, `n` is the number of non-diff genes and `k` is the number of genes in the gene set.Note it is the same as

```
phyper(58 - 1, 837, 19761, 764, lower.tail = FALSE)
# [1] 3.75562e-06
```

P-value from Binominal distribution can be calcualted with the function `pbinom()`. The usage is

```
pbinom(q - 1, size, prob, lower.tail = FALSE)
```

`q` is the number of DE genes in the gene set, `size` is the total number of genes in the gene set, `prob` is the successful rate for the binomial distribution, here it is the background probabiliyt of DE genes. Similarly, we substract 1 form the original `q` and taking teh upper tail.

Let's fill the values into the function call.

```
pbinom(58 - 1, 764, 837/20598, lower.tail = FALSE)
# [1] 6.032068e-06
```

We can also calculate from the other dimension. In this case, `size` is the total number of DE genes, and prob is the probability of a genes being in teh gene set.

```
pbinom(58 - 1, 837, 764/20598, lower.tail = FALSE)
# [1] 6.315109e-06
```

Since the binomial distribution is an approximation of the hypergenometric distribution, p-values form two different dimensions are slighly different.

To calculate the z-test, we first calculate p_1 and p_2 which are the probability of genes being DE in teh gene set and not in the gene set. ALso the pool probability.

```
p1 = 58/764      # prob of genes being DE in the gene set
p2 = 779/19834   # prob of genes being DE not in the gene set
p = 837/20598    # prob of genes being DE in total
```

Following the formula of z-score, we have:

```
z = abs(p1 - p2)/sqrt(p * (1-p))/sqrt(1/764 + 1/19834)
```

Since z follows a standard normal distribution, we can use `pnorm()` to calcualte p-value. Here we assume the z-test is a two-sided test, thus the p-value is $Pr(X > z) + Pr(X < -z)$.

```
2*pnorm(z, lower.tail = FALSE)
# [1] 4.820302e-07
```

We can also try to calculate the p-valeu from other other direction. similarly,

```
p1 = 58/837     # prob of being a gene set gene in the DE list
p2 = 706/19761 # prob of being a gene set gene in the non-DE list
p = 764/20598  # prob of being a gene set gene in total

z = abs(p1 - p2)/sqrt(p * (1-p))/sqrt(1/837 + 1/19761)
2*pnorm(z, lower.tail = FALSE)
# [1] 4.820302e-07
```

The two p-values are identical.

Fisher's exact test can be directly performed by the function `fisher.test()`. The input is the 2x2 contigency table without the margins. In the following code, transforming the matrix gives identical result.

```
cm = matrix(c(58, 779, 706, 19055), nrow = 2)
cm
#      [,1]  [,2]
# [1,]   58   706
# [2,]  779 19055
fisher.test(cm)
#
#   Fisher's Exact Test for Count Data
#
# data:  cm
# p-value = 5.513e-06
# alternative hypothesis: true odds ratio is not equal to 1
# 95 percent confidence interval:
#  1.495845 2.656573
# sample estimates:
# odds ratio
#    2.00944
```

`fisher.test()` generates many other results, here the odd ratio is the statistic of fisher exact test. The odd ratio (OD) is defined as

$$\mathrm{OD} = \frac{n_{11}}{n_{21}} / \frac{n_{11}}{n_{22}} = \frac{n_{11}n_{22}}{n_{12}n_{21}}$$

which is the ratio of fraction of DE in the gene set and not in the gene set. If odd ratio is larger than 1, over-representation.

Last, the Chi-square test can be applied with the function `chisq.test()`. Similarly the input is the 2x2 contigency table without the margins. Note here we also set `correct = FALSE` to perform the original Chi-square test.

```
chisq.test(cm, correct = FALSE)
#
#   Pearson's Chi-squared test
#
# data:  cm
# X-squared = 25.334, df = 1, p-value = 4.82e-07
```

If we compare the Chi-square statistic to the previous z, we can see there is a relation of $chisq = z^2$.

```
z^2
# [1] 25.33442
```

Since there are several ways to perform the ORA test, it would be interesting to test which method runs faster. In the following code, I used the **microbenchmark** package which benchmark pieces of codes with higher precision.

```
library(microbenchmark)

microbenchmark(
    hyper = phyper(58 - 1, 764, 19834, 837, lower.tail = FALSE),
    fisher = fisher.test(cm),
    binom = pbinom(58 - 1, 764, 837/20598, lower.tail = FALSE),
    chisq = chisq.test(cm, correct = FALSE),
    ztest = {
        p1 = 58/764
        p2 = 779/19834
        p = 837/20598

        z = abs(p1 - p2)/sqrt(p * (1-p))/sqrt(1/764 + 1/19834)

        2*pnorm(z, lower.tail = FALSE)
    },
    times = 1000
)
# Unit: nanoseconds
#    expr    min        lq        mean    median        uq       max neval
#   hyper   1129    1698.0    2493.252    2533.5    3032.0     18874  1000
#  fisher 996610 1136764.5 1645767.922 1206255.0 1398183.0 191658211  1000
#   binom    999    1529.5    2461.760    2361.0    3107.0     17015  1000
#   chisq  41947   53769.0   63229.380   62469.0   70764.5    142784  1000
#   ztest   2079    2923.0    4551.284    4562.0    5550.0     23784  1000
```

We can see from the benchmark result, hypergeometric and bionimal distribution-based method are the fastest. As a comparison, Chi-square test and fisher's method run slow, especially for fisher's exact method. The reason is the latter two also include many other calculations besides p-values. This benchmark results actually tells us, if in the future readers want to implement ORA analysis by their own, hypergeometric and bimonial methods should be considered firstly.

Assume genes are independent and can be picked with equal probabiliyt, hypergeometric or fisher's exact test gives the exact distribution without approximation.

Implement ORA in R

As has been demonstarted, it is more recommand to use hypergeometric distribution to calculate p-values.

To implement an function that performs ORA anlaysis, a natrual thought is first implement a `ora_single()` which performs ORA for a single gene set, and a wrapper function `ora()` which applies `ora_single()` to every gene set.

In the following, I assume the gene sets is represented as a list of vectors where each vector corresponds to a gene set. Also I assume genes in the DE gene list, in teh gene sets and in teh backgorund are already in teh same gene ID type.

Next code demonstrates how to implement `ora_single()`. Given the three argument `genes`, `gene_set`, `universe` which are three character vectors, we just need to calculate the numbers for `phyer()`.

```r
ora_single = function(genes, gene_set, universe) {
    n_universe = length(universe)

    x = length(intersect(genes, gene_set))
    m = length(gene_set)
    n = n_universe - m
    k = length(genes)

    phyper(x - 1, m, n, k, lower.tail = FALSE)
}
ora_single(diff_genes, gene_set, universe)
# [1] 4.504578e-06
```

Next we implement `ora()`. `ora()` is a rather simple function which basically apply `ora_single()` to every gene sets. In the following code, use of `sapply()` can also be replaced by a `for` loop.

P-values are the most important output from the analysis. In `ora()`, to make the output more informative, we can reformat the output as a data frame and add more columns there.

```
ora = function(genes, gene_sets, universe) {
    sapply(gene_sets, ora_single)
}
```

`ora()` can already do a great job of full functional for ORA analysis. The next version is an improved one

1. As I have introduced previously, gene sets can be represented in both lists of data frames. Here the improved vesion supports both list and data frames as input. If the input is the data frame, it will be converetd to a list internally. Note, if the gene sets are in a data frame, the first column should be gene set names and teh second column should be genes.

2. Sometimes users do not know what "background" to provide.

3. DE genes, gene sets and background genes sometimes come from differnet sources. It is not already ensured the background genes include all DE genes and gene sets. these two lines of code actually do intersection of DE genes and gene sets to. Note these two lines are the most time comsuing part in this function.

4. being different from `ora_single()`, we run `phyer()` in an vectorized way where the first four argument can all be vectors. This means, we can first calculate xxx as vectors then calculate … simultanuously, without using a `sapply()` or `for` loop. because xxx is normally slower than directly vectorizing the calculation.

5. we add more columns to the result table,

```
ora = function(genes, gene_sets, universe) {
    # 1
    if(is.data.frame(gene_sets)) {
        gene_sets = gs_dataframe2list(gene_sets)
    }
    # 2
    if(missing(universe)) {
        universe = unique(unlist(gene_sets))
    } else {
        universe = unique(universe)
    }
    # 3
    gs_names = names(gene_sets)
    genes = intersect(genes, universe)
    gene_sets = lapply(gene_sets, function(x) intersect(x, universe))

    n_universe = length(universe)
    n_genes = length(genes)
    #4
    x = sapply(gene_sets, function(x) length(intersect(x, genes)))
    m = sapply(gene_sets, length)
    n = n_universe - m
    k = n_genes
    p = phyper(x - 1, m, n, k, lower.tail = FALSE)
    # 5
    data.frame(gene_set = gs_names,
               hits = x,
               gene_set_size = m,
               ratio_in_gene_set = x/m,
               ratio_in_genes = x/n_genes,
               enrichment_score = x*n_universe/m/n_genes,
               p_value  = p,
               p_adjust = p.adjust(p, "BH"))
}
```

The new version `ora()` is a general-purpose function. It has no assumption of specific organism and gene sets. ... In the following part of this book, we will use `ora()` xxx

Current tools

There are many tools that implement ORA analysis compared to other gene set enrichment analysis tools which will introduced in later chapters, mainly because it runs fast, the method is simply to understand. All most all the ORA web-based tools are in a two-step analysis. 1. upload the gene lists and setting parameters and 2. see the results. In this section, we will go through three web-based ORA tools, as well as one Bioconductor package.

Online tools

There are quite a lot of web-based tools for ORA analysis.

Perform ORA with clusterProfiler

The **clusterProfiler** is the most widely used R packages for gene set enrichment anlaysis. Since it is implemnented as an R package, it can easily integerate into the bioconductor annotation ecosystem for the most up-to-date and rich data.

First let's load the package.

```
library(clusterProfiler)
```

Gene ontology enrichment

The function `enrichGO()` applies ORA on GO gene sets. The two mandatory arguments are the gene vector and the name of the corresponding organism package. On Bioconductor, there are a variaty organism packages.

```
tb = enrichGO(gene = diff_genes, OrgDb = "org.Hs.eg.db", ont = "BP")
```

As has been introduced in Chapter x, in organism package, there is a database object with the same name as the package. The database object can be directly set to `OrgDb` argument. But note, of course you need to load the corresponding organism package first.

```
library(org.Hs.eg.db)
enrichGO(gene = diff_genes, OrgDb = org.Hs.eg.db, ont = "BP")
```

Entrez ID is the primary gene ID type in the organism packages. If the ID type of the input gene list is already in EntreZ ID, that is all you need to set with `enrichGO()`. But if the ID type is something else, but supported in the organism package, such as `SYMBOL` or `ENSEMBL`, they can be set with teh `keytype` argument and internally they will be converted. If the ID tyep is not supported in teh organism package, you have to look for other resources to convert to Entrez IDs manually.

Background genes are important for ORA analysis. You can also set the background genes with teh `universe` argument in `enrichGO()`. If it is not specified, background genes will be all genes in the gene set collections. In Section x, I will talk about the effect of selecting different background genes in ORA.

Argument `ont` controls which GO ontology to use. The default value is `MF`, here I set it to `BOP`.

The value returned by `enrichGO()` is a table-like object. Let's print the first several rows of the result table. The output may look slightly different if you try it on your xxx.

```
head(tb)
#           ID             Description GeneRatio   BgRatio       pvalue
# 1 GO:0001819 positive regulation ..    79/787 486/18903 7.437659e-26
# 2 GO:0019221 cytokine-mediated si..    74/787 496/18903 5.900976e-22
# 3 GO:0032103 positive regulation ..    71/787 464/18903 9.629161e-22
# 4 GO:0031349 positive regulation ..    56/787 307/18903 5.042886e-21
#       p.adjust       qvalue                     geneID Count
# 1 3.824444e-22 2.972715e-22 2268/4843/7305/6556/..    79
# 2 1.517141e-18 1.179263e-18 53832/608/51208/2920..    74
# 3 1.650438e-18 1.282875e-18 7305/8692/3430/5743/..    71
# 4 6.482630e-18 5.038904e-18 7305/8692/3430/5743/..    56
```

`tb` is actually is a `enrichResult` object which is defined in **clusterProfiler**[1], but it works prefectly with functions which expects a data frame as input, e.g. subsetting or `write.table()`. But because if you use `tb` as a data frame, it only contains significant terms. If want the complete table no matter the term is significant or not, you can directly extract the `result` slot.

```
full_table = tb@result
```

KEGG pathway enrichment

In Chapter, I introduce how to obtain KEGG pathway gene sets directly from KEGG database. In **clusterProfiler**, there is also a `enrichKEGG()` function which automatically download pathway gene sets and perform ORA.

To be consistent to KEGG IDs, genes are suggested already in Entrez ID types because it is used on KEGG. If not, please consider to convert to Entrez ID explicitely. The organism is set as a three letter code.

```
tb = enrichKEGG(diff_genes, organism = "hsa")
```

Reactome pathway enrichment

The package **ReactomePA** which utiliazed the same ORA implementaion as in **clusterProfiler** performs ORA on Reactome pathways. Again, as introduced in xx, the gene ID should already be in EntreZ IDs. Organism can be specified with the `organism` argument, but there is only a few organism suppoted on Reactome database.

```
library(ReactomePA)
tb = enrichPathway(diff_genes, organism = "human")
```

MSigDB gene set enrichment

There is no built-in function specific for MSigDB gene sets in **clusterProfiler**, but **clusterProfiler** provides a general-purpose function `enrichr()` which accepts manually-specified gene sets. The gene sets object is simply a two-column data frame:

- the first column is the gene set ID
- the second column is the gene ID

```
msigdb_h  = get_msigdb(version = "2023.1.Hs", collection = "h.all", as_table = TRUE)
head(msigdb_h)
#                          gene_set gene
# 1 HALLMARK_TNFA_SIGNALING_VIA_NFKB 3726
# 2 HALLMARK_TNFA_SIGNALING_VIA_NFKB 2920
```

[1]Actually it is defined in the **DOSE** packages. All these package share the same implementation, just focusing on different types of gene sets.

```
# 3 HALLMARK_TNFA_SIGNALING_VIA_NFKB  467
# 4 HALLMARK_TNFA_SIGNALING_VIA_NFKB 4792
# 5 HALLMARK_TNFA_SIGNALING_VIA_NFKB 7128
# 6 HALLMARK_TNFA_SIGNALING_VIA_NFKB 5743
tb = enricher(gene = diff_genes, TERM2GENE = msigdb_h)
```

ORA on non-model organisms

Organism with a OrgDb object

```
enrichGO(gene_list, ont = "BP", OrgDb = "org.Rn.eg.db")
enrichGO(gene_list, ont = "BP", OrgDb = org.Rn.eg.db)
```

```
org = ah[["AH108106"]]
enrichGO(gene_list, ont = "BP", OrgDb = org)
```

```
ora(gene_list,
    get_go_gene_sets_from_orgdb(org, ontology = "BP"),
    get_all_pc_genes_from_orgdb(org))
```

GO gene sets from BioMart

```
...
at = c("entrezgene_id", "go_id", "namespace_1003")
tb = getBM(attributes = at, mart = ensembl)
tb = tb[tb$namespace_1003 == "biological_process", ]
enricher(gene_list, TERM2GENE = tb)
```

```
ora(gene_list, tb)
```

KEGG gene sets for other organisms

```
enrichKEGG(gene_list, organism = "aml")
```

gene sets inferered from orthologues

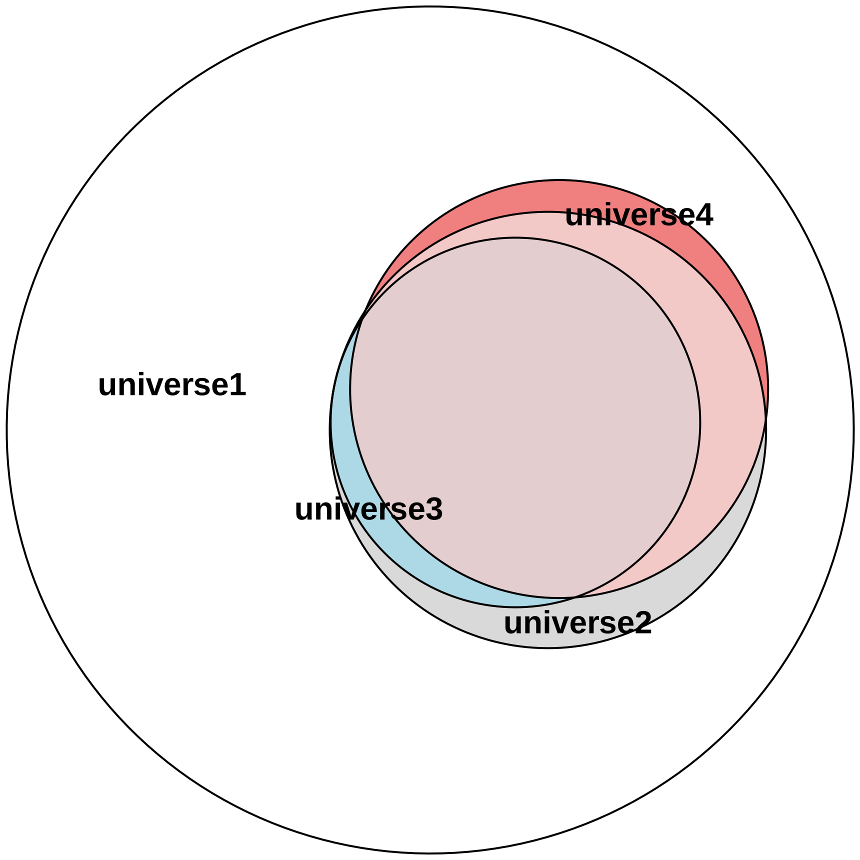
```
enricher(gene_list, TERM2GENE = gs_list2dataframe(gs_panda))
ora(gene_list, gs_panda)
```

Choose a proper background

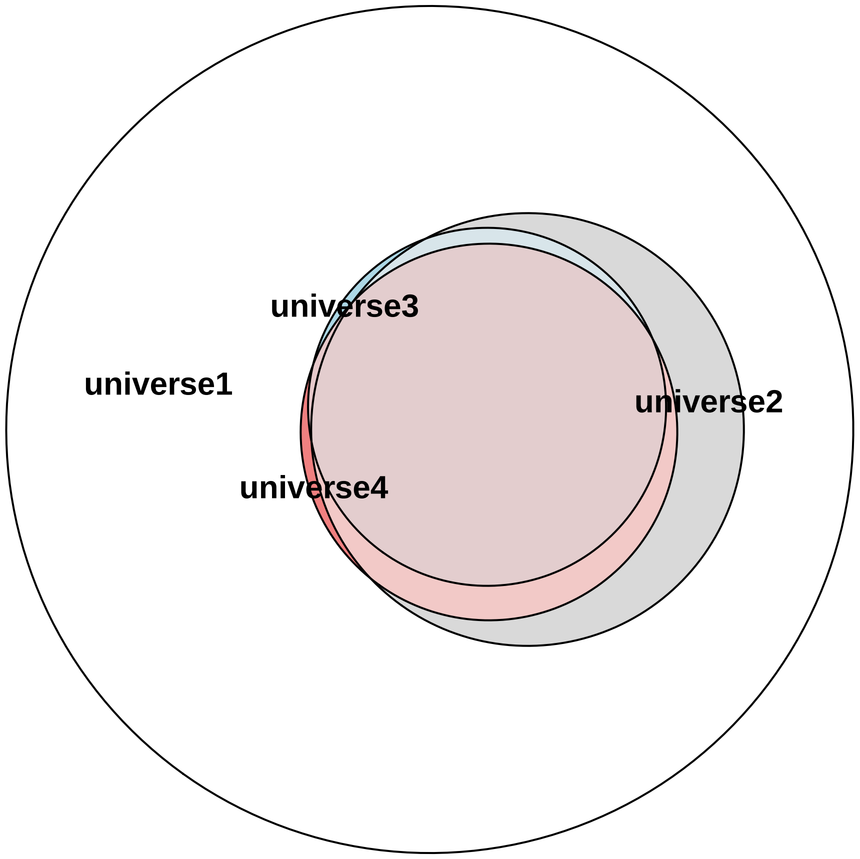
Background is important

1. all genes in the genome
2. all protein-coding genes

3. all expressed protein-coding genes
4. all genes that has a annotation in the gene set collection



```
tb1 = ora(gene_list, gs, universe1)
tb2 = ora(gene_list, gs, universe2)
tb3 = ora(gene_list, gs, universe3)
tb4 = ora(gene_list, gs, universe4)

plot(euler(list(
    universe1 = tb1$gene_set[tb1$p_adjust < 0.01],
    universe2 = tb2$gene_set[tb2$p_adjust < 0.01],
    universe3 = tb3$gene_set[tb3$p_adjust < 0.01],
    universe4 = tb4$gene_set[tb4$p_adjust < 0.01]
)))
```



Limitations of ORA

ORA analysis is simply and it runs fast. Thus, currently, there are many online tools support it. However, there are many limiations.

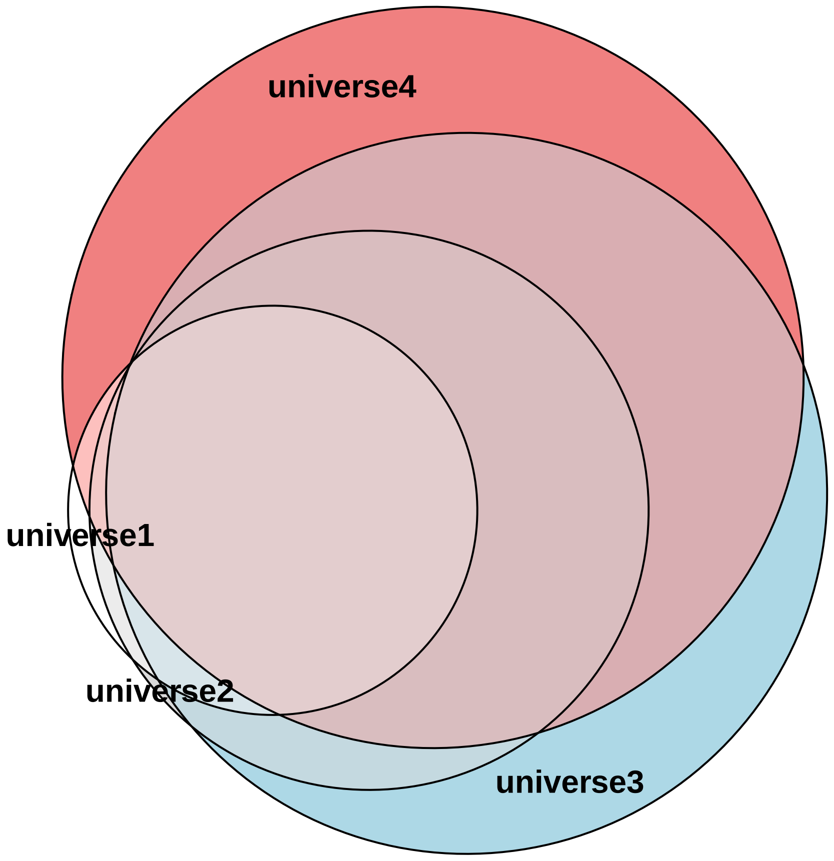
different tools generate inconsistent results

There are a lot of tools, but using the same gene set database, results from tools normally do not agree very well. The reasons are:

1. Different versions of annotation databases.
2. How they process redundant terms
3. Different default cutoffs
4. Different methodd for control p-values
5. differnet background genes.

```
gene_list = mapIds(org.Hs.eg.db, keys = rownames(diff_tb)[order(diff_tb$p_adjust)], keytype = "EN
gene_list = gene_list[!is.na(gene_list)]
```

```
tb1 = ora(gene_list[1:128], gs, universe2)
tb2 = ora(gene_list[1:256], gs, universe2)
tb3 = ora(gene_list[1:512], gs, universe2)
tb4 = ora(gene_list[1:1024], gs, universe2)

plot(euler(list(
    universe1 = tb1$gene_set[tb1$p_adjust < 0.01],
    universe2 = tb2$gene_set[tb2$p_adjust < 0.01],
    universe3 = tb3$gene_set[tb3$p_adjust < 0.01],
    universe4 = tb4$gene_set[tb4$p_adjust < 0.01]
)))
```



sensitive to the selection of background genes.

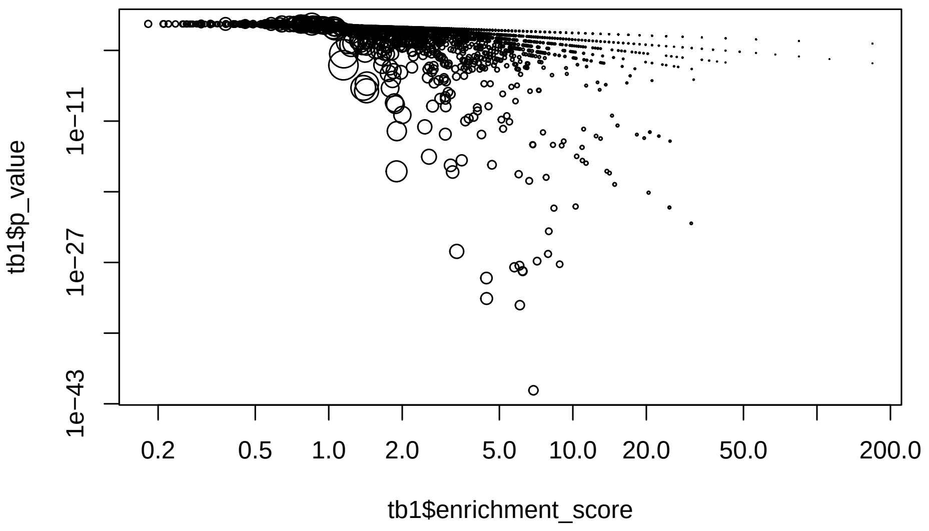
Genome / all protein-coding gene / all genes measured? The selection mainly affects the value in the blue cells. Note the ORA actually compares diff gene in the set to the 'other gene,' and we actually assume 'other genes' are not diff nor in the set, thus, "other genes" are not useless. Select a "large" background may increase false positives. Select a "small" background may miss some positives, but it has low false positives. Also some arguments: If genes are not measured, they should not be put into the analysis.

In general, with larger background set, the p-value becomes significant. ORA

Preference of larger gene sets

There is a trend that large gene sets may have more significant p-values. This is actually expected because as the sample size increases, the test power also increases. Under the context of hypergeometric test or Binomial test, number of genes is the "sample size."

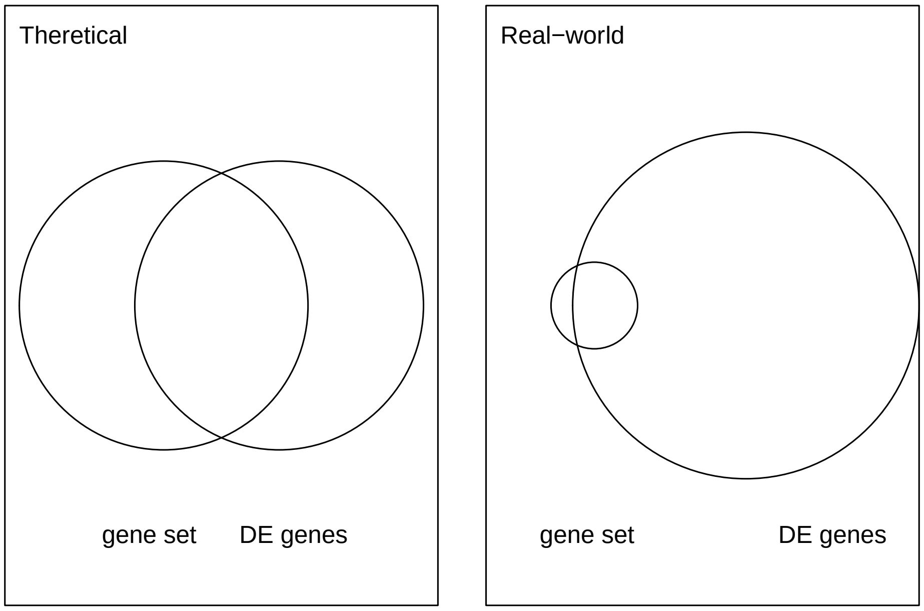
```
plot(tb1$enrichment_score, tb1$p_value, log = "xy", cex = sqrt(tb1$gene_set_size)/50)
```



Imbalanced contigency table

In real-world scenario, as show in xx. There are many gene sets with small number of genes, which makes the 2x2 contigency imbalanced.

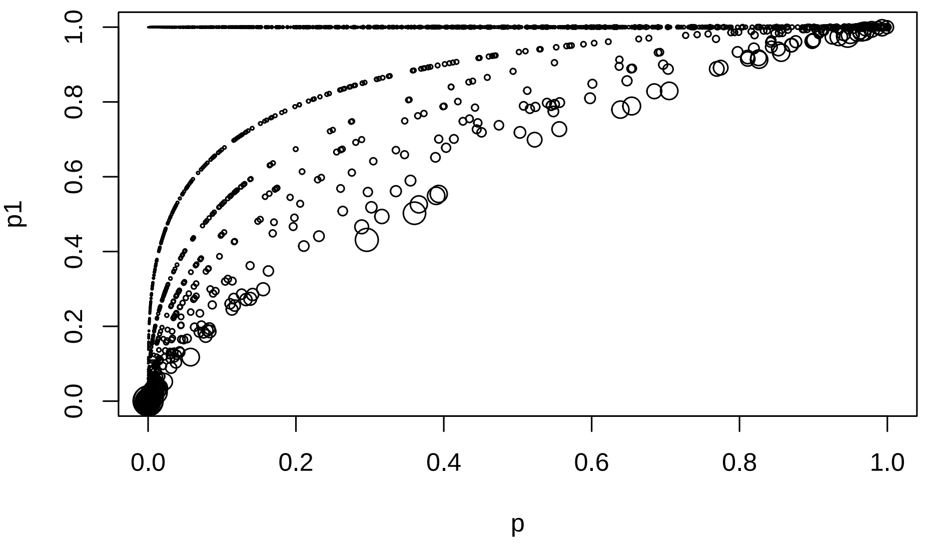
```
library(grid)
grid.newpage()
pushViewport(viewport(x = 0, width = 0.5, just = "left"))
pushViewport(viewport(xscale = c(-1.5, 1.5), yscale = c(-1.5, 1.5), width = 0.9, height = 0.9))
grid.rect()
grid.circle(x = -0.4, y = 0, r = 1, default.units = "native")
grid.circle(x = 0.4, y = 0, r = 1, default.units = "native")
grid.text("gene set", x = -0.5, y = -1.1, just = "top", default.units = "native")
grid.text("DE genes", x = 0.5, y = -1.1, just = "top", default.units = "native")
grid.text("Theretical", x = -1.4, y = 1.4, default.units = "native", just = c("left", "
popViewport()
popViewport()

pushViewport(viewport(x = 0.5, width = 0.5, just = "left"))
pushViewport(viewport(xscale = c(-1.5, 1.5), yscale = c(-1.5, 1.5), width = 0.9, heigh
grid.rect()
grid.circle(x = -0.75, y = 0, r = 0.3, default.units = "native")
grid.circle(x = 0.3, y = 0, r = 1.2, default.units = "native")
grid.text("gene set", x = -0.8, y = -1.1, just = "top", default.units = "native")
grid.text("DE genes", x = 0.9, y = -1.1, just = "top", default.units = "native")
grid.text("Real-world", x = -1.4, y = 1.4, default.units = "native", just = c("left", "
popViewport()
popViewport()
```



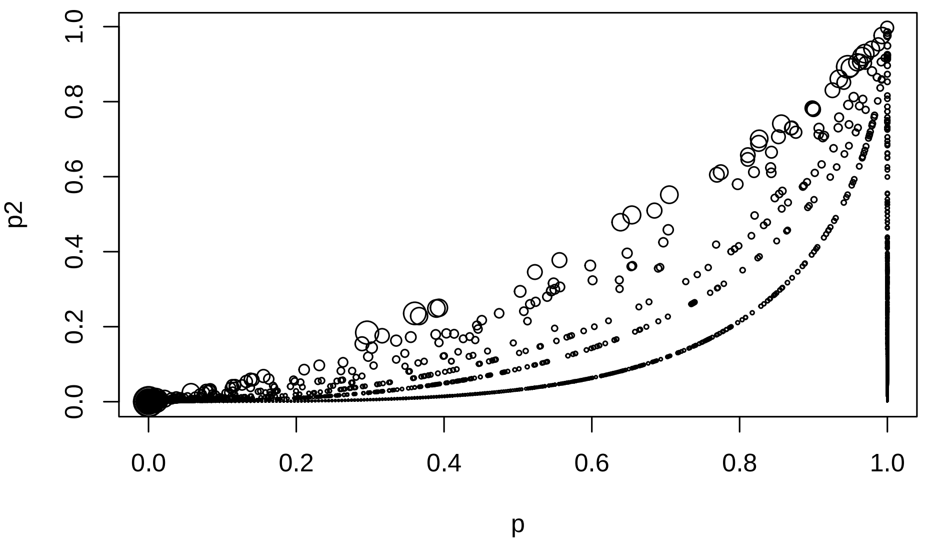
In many cases when the gene set is small, the enrichment anlaysis will be sensitive to the value of number of DE genes in the gene set, i.e., the valeu of n_{11}.

```
x = tb1$hits
m = tb1$gene_set_size
k = x/tb1$ratio_in_genes; k = k[!is.na(k)][1]
n = tb1$enrichment_score*m*k/x; n = n[!is.na(n)][1] - m

p = phyper(x - 1, m, n, k, lower.tail = FALSE)
p1 = phyper(x - 1 - 2, m, n, k, lower.tail = FALSE)
```

```
p2 = phyper(x - 1 + 2, m, n, k, lower.tail = FALSE)

plot(p, p1, cex = sqrt(tb1$gene_set_size)/50)
```



```
plot(p, p2, cex = sqrt(tb1$gene_set_size)/50)
```



In DE analysis, normally we set a cutoff of adjusted p-values for filter the significant DE genes, since the p-values are sensitive to n_{11}, actually xxx

Theoretical reasons

1. the assumption

The GSEA method

Placeholder

Overview

The GSEA method, version one

GSEA v1, step 1

GSAE v1, step 2

Permutation-based test

The GSEA method, version 2

Compare the two GSEA

Permutations

Other aspects of GSEA

The direction of GSEA

Leading edge genes

Normalized enrichment score

Compare ORA and GSEA

GSEA framework

Placeholder

Overview

The univariate methods

Gene-level methods

Transformation of gene-level statistics

Set-level methods

Current tools

Implement ORA under univariate framework

Null distribution of set-level statistics

Permutation-based distribution

The multivariate methods

Implementation of GSEA framework

geneset to be a list of gene sets

current tools for GSEA framework

Important aspects of GSEA methogology

recommandations of methods

Gene Set Enrichment Analysis in Genomics

Placeholder

Overview

The GREAT method

Construct region sets

The binomial model

The hypergeometric model when dealing with background

implementation

The web-based tool

The rGREAT package

Local GREAT analysis

RNASeq and methylation-adjusted ORA

SNP-based GSEA

general comments

Topology-based pathway enrichmetn

Overview

Gene sets are represented as a vector of

Use topology informatino

Pathway common structure

General process of utilizing topology information

Centrality measures

centrality-based pathway enrichment

SPIA: pathway impact analysis

R packages for topology-based GSEA

Extensions of GSEA

Single sample-based GSEA

Ensembles of multiple GSEA methods

TBA

Visualization

general xxx

Visualize by statistical plots

network visualization

Enrichment map

Clustering and simplifying GSEA results

Placeholder

Overview

measures of similarities

overlap-based

semantic measures

IC

Enrichment map

David

simplifyEnrichment