Introduction to DGE

View on GitHub

Approximate time: 30 minutes

Learning Objectives:

- Utilize genomic annotation databases to perform functional analysis on gene lists
- Understand the types of information stored in different databases
- Explore the pros and cons of several popular R packages for retrieval of genomic annotations

Genomic annotations

The analysis of next-generation sequencing results requires associating genes, transcripts, proteins, etc. with functional or regulatory information. To perform functional analysis on gene lists, we often need to obtain gene identifiers that are compatible with the tools we wish to use and this is not always trivial. Here, we discuss **ways in which you can obtain gene annotation information and some of the advantages and disadvantages of each method**.

Databases

We retrieve information on the processes, pathways, etc. (for which a gene is involved in) from the necessary database where the information is stored. The database you choose will be dependent on what type of information you are trying to obtain. Examples of databases that are often queried, include:

General databases

Offer comprehensive information on genome features, feature coordinates, homology, variant information, phenotypes, protein domain/family information, associated biological processes/pathways, associated microRNAs, etc.:

- **Ensembl** (use Ensembl gene IDs)
- **NCBI** (use Entrez gene IDs)
- **UCSC**
- **EMBL-EBI**

Annotation-specific databases

Provide annotations related to a specific topic:

- **Gene Ontology (GO):** database of gene ontology biological processes, cellular components and molecular functions - based on Ensembl or Entrez gene IDs or official gene symbols
- **KEGG:** database of biological pathways - based on Entrez gene IDs
- **MSigDB:** database of gene sets
- **Reactome:** database of biological pathways
- **Human Phenotype Ontology:** database of genes associated with human disease
- **CORUM:** database of protein complexes for human, mouse, rat
- **...**

This is by no means an exhaustive list, there are many other databases available that are not listed here.

Genome builds

Before you begin your search through any of these databases, you should know which **build of the genome** was used to generate your gene list and make sure you use the **same build for the annotations** during functional analysis. When a new genome build is acquired, the names and/or coordinate location of genomic features (gene, transcript, exon, etc.) may change. Therefore, the annotations regarding genome features (gene, transcript, exon, etc.) is genome-build specific and we need to make sure that our annotations are obtained from the appropriate resource.

For example, if we used the GRCh38 build of the human genome to quantify gene expression used for differential expression analysis, then we should use the **same GRCh38 build** of the genome to convert between gene IDs and to identify annotations for each of the genes.

Tools for accessing databases

Within R, there are many popular packages used for gene/transcript-level annotation. These packages provide tools that take the list of genes you provide and retrieve information for each gene using one or more of the databases listed above.

Annotation tools: for accessing/querying annotations from a specific databases

Tool	Description	Pros	Cons
org.Xx.eg.db	Query gene feature information for the organism of interest	gene ID conversion, biotype and coordinate information	only latest genome build available

Tool	Description	Pros	Cons
EnsDb.Xx.vxx	Transcript and gene-level information directly fetched from Ensembl API (similar to TxDb, but with filtering ability and versioned by Ensembl release)	easy functions to extract features, direct filtering	Not the most up-to-date annotations, more difficult to use than some packages
TxDb.Xx.UCSC.hgxx.knownGene	UCSC database for transcript and gene-level information or can create own *TxDb* from an SQLite database file using the *GenomicFeatures* package	feature information, easy functions to extract features	only available current and recent genome builds - can create your own, less up-to-date with annotations than Ensembl
annotables	Gene-level feature information immediately available for the human and model organisms	super quick and easy gene ID conversion, biotype and coordinate information	static resource, not updated regularly
biomaRt	An R package version of the Ensembl BioMart online tool	all Ensembl database information available, all organisms on Ensembl, wealth of information	

Interface tools: for accessing/querying annotations from multiple different annotation sources

- **AnnotationDbi:** queries the *OrgDb*, *TxDb*, *Go.db*, *EnsDb*, and *BioMart* annotations.
- **AnnotationHub:** queries large collection of whole genome resources, including ENSEMBL, UCSC, ENCODE, Broad Institute, KEGG, NIH Pathway Interaction Database, etc.

> **NOTE:** These are both packages that can be used to create the `tx2gene` files we had you download at the beginning of this workshop.

AnnotationDbi

AnnotationDbi is an R package that provides an interface for connecting and querying various annotation databases using SQLite data storage. The AnnotationDbi packages can query the *OrgDb*, *TxDb*, *EnsDb*, *Go.db*, and *BioMart* annotations. There is helpful documentation available to reference when extracting data from any of these databases.

While AnnotationDbi is a popular tool, **we will not be walking through code to use this package**. However, if you are interested in more detail, we have materials linked here with examples using our current dataset.

AnnotationHub

AnnotationHub is a wonderful resource for accessing genomic data or querying large collection of whole genome resources, including ENSEMBL, UCSC, ENCODE, Broad Institute, KEGG, NIH Pathway Interaction Database, etc. All of this information is stored and easily accessible by directly connecting to the database.

To get started with AnnotationHub, we first load the library and connect to the database:

```
# Load libraries
library(AnnotationHub)
library(ensembldb)

# Connect to AnnotationHub
ah <- AnnotationHub()
```

> What is a cache?
>
> A cache is used in R to store data or a copy of the data so that future requests can be served faster without having to re-run a lengthy computation.
>
> The `AnnotationHub()` command creates a client that manages a local cache of the database, helping with quick and reproducible access. When encountering question

> `AnnotationHub does not exist, create directory?`, you can anwser either `yes` (create a permanant location to store cache) or `no` (create a temporary location to store cache). `hubCache(ah)` gets the file system location of the local AnnotationHub cache. `hubUrl(ah)` gets the URL for the online hub.

To see the types of information stored inside our database, we can just type the name of the object:

```
# Explore the AnnotationHub object
ah
```

Using the output, you can get an idea of the information that you can query within the AnnotationHub object:

```
AnnotationHub with 47240 records
# snapshotDate(): 2019-10-29
# $dataprovider: BroadInstitute, Ensembl, UCSC, ftp://ftp.ncbi.nlm.nih.gov/gene/DA
# $species: Homo sapiens, Mus musculus, Drosophila melanogaster, Bos taurus, Pan t
# $rdataclass: GRanges, BigWigFile, TwoBitFile, Rle, OrgDb, EnsDb, ChainFile, TxDb
# additional mcols(): taxonomyid, genome, description, coordinate_1_based,
#   maintainer, rdatadateadded, preparerclass, tags, rdatapath, sourceurl,
#   sourcetype
# retrieve records with, e.g., 'object[["AH5012"]]'

            title
  AH5012  | Chromosome Band
  AH5013  | STS Markers
  AH5014  | FISH Clones
  AH5015  | Recomb Rate
  AH5016  | ENCODE Pilot
  ...       ...
  AH78364 | Xiphophorus_maculatus.X_maculatus-5.0-male.ncrna.2bit
  AH78365 | Zonotrichia_albicollis.Zonotrichia_albicollis-1.0.1.cdna.all.2bit
  AH78366 | Zonotrichia_albicollis.Zonotrichia_albicollis-1.0.1.dna_rm.toplevel.2b
  AH78367 | Zonotrichia_albicollis.Zonotrichia_albicollis-1.0.1.dna_sm.toplevel.2b
  AH78368 | Zonotrichia_albicollis.Zonotrichia_albicollis-1.0.1.ncrna.2bit
```

Notice the note on retrieving records with `object[[AH2]]` - this will be how we can **extract a single record** from the AnnotationHub object.

If you would like to see more information about any of the classes of data you can extract that information as well. For example, if you wanted to **determine all species information available**, you could explore that within the AnnotationHub object:

```
# Explore all species information available
unique(ah$species) %>% View()
```

In addition to species information, there is also additional information about the type of Data Objects and the Data Providers:

```
# Explore the types of Data Objects available
unique(ah$rdataclass) %>% View()

# Explore the Data Providers
unique(ah$dataprovider) %>% View()
```

Now that we know the types of information available from AnnotationHub we can query it for the information we want using the `query()` function. Let's say we would like to **return the Ensembl `EnsDb` information for Human**. To return the records available, we need to use the terms as they are output from the `ah` object to extract the desired data.

```
# Query AnnotationHub
human_ens <- query(ah, c("Homo sapiens", "EnsDb"))
```

The query retrieves all **hits for the `EnsDb` objects**, and you will see that they are listed by the release number. The most current release for GRCh38 is Ensembl98 and AnnotationHub offers that as an option to use. However, if you look at options for older releases, for Homo sapiens it only go back as far as Ensembl 87. This is fine if you are using GRCh38, however if you were using an older genome build like hg19/GRCh37, you would need to load the `EnsDb` package if available for that release or you might need to build your own with `ensembldb`.

```
human_ens

AnnotationHub with 13 records
# snapshotDate(): 2019-10-29
# $dataprovider: Ensembl
# $species: Homo sapiens
# $rdataclass: EnsDb
# additional mcols(): taxonomyid, genome, description, coordinate_1_based,
#   maintainer, rdatadateadded, preparerclass, tags, rdatapath, sourceurl,
#   sourcetype
# retrieve records with, e.g., 'object[["AH53211"]]'

            title
  AH53211 | Ensembl 87 EnsDb for Homo Sapiens
  AH53715 | Ensembl 88 EnsDb for Homo Sapiens
  AH56681 | Ensembl 89 EnsDb for Homo Sapiens
  AH57757 | Ensembl 90 EnsDb for Homo Sapiens
  AH60773 | Ensembl 91 EnsDb for Homo Sapiens
  ...       ...
  AH67950 | Ensembl 95 EnsDb for Homo sapiens
  AH69187 | Ensembl 96 EnsDb for Homo sapiens
  AH73881 | Ensembl 97 EnsDb for Homo sapiens
```

```
  AH73986 | Ensembl 79 EnsDb for Homo sapiens
  AH75011 | Ensembl 98 EnsDb for Homo sapiens
```

In our case, we are looking for the latest Ensembl release so that the annotations are the most up-to-date. To extract this information from AnnotationHub, we can use the AnnotationHub ID to **subset the object**:

```
# Extract annotations of interest
human_ens <- human_ens[["AH75011"]]
```

Now we can use `ensembldb` functions to extract the information at the gene, transcript, or exon levels. We are interested in the gene-level annotations, so we can extract that information as follows:

```
# Extract gene-level information
genes(human_ens, return.type = "data.frame") %>% View()
```

But note that it is just as easy to get the transcript- or exon-level information:

```
# Extract transcript-level information
transcripts(human_ens, return.type = "data.frame") %>% View()

# Extract exon-level information
exons(human_ens, return.type = "data.frame") %>% View()
```

To **obtain an annotation data frame** using AnnotationHub, we'll use the `genes()` function, but only keep selected columns and filter out rows to keep those corresponding to our gene identifiers in our results file:

```
# Create a gene-level dataframe
annotations_ahb <- genes(human_ens, return.type = "data.frame")  %>%
  dplyr::select(gene_id, gene_name, entrezid, gene_biotype) %>%
  dplyr::filter(gene_id %in% res_tableOE_tb$gene)
```

This dataframe looks like it should be fine as it is, but we look a little closer we will notice that the column containing Entrez identifiers is a list, and in fact there are many Ensembl identifiers that map to more than one Entrez identifier!

```
# Wait a second, we don't have one-to-one mappings!
class(annotations_ahb$entrezid)
which(map(annotations_ahb$entrezid, length) > 1)
```

So what do we do here? And why do we have this problem? An answer from the Ensembl Help Desk is that this occurs when we cannot choose a perfect match; ie when we have two good matches, but one does not appear to match with a better percentage than the other. In that case, we assign both matches. What we will do is choose to **keep the first identifier for these multiple mapping cases**.

```
annotations_ahb$entrezid <- map(annotations_ahb$entrezid,1) %>%  unlist()
```

> **NOTE:** Not all databases handle multiple mappings in the same way. For example, if we used the OrgDb instead of the EnsDb:
>
> ```
> human_orgdb <- query(ah, c("Homo sapiens", "OrgDb"))
> human_orgdb <- human_ens[["AH75742"]]
> annotations_orgdb <- select(human_orgdb, res_tableOE_tb$gene, c("SYMBOL",
> ```
>
> We would find that multiple mapping entries would be automatically reduced to one-to-one. We would also find that more than half of the input genes do not return any annotations. This is because the OrgDb family of database are primarily based on mapping using Entrez Gene identifiers. Since our data is based on Ensembl mappings, using the OrgDb would result in a loss of information.

Let's take a look and see how many of our Ensembl identifiers have an associated gene symbol, and how many of them are unique:

```
which(is.na(annotations_ahb$gene_name)) %>% length()

which(duplicated(annotations_ahb$gene_name)) %>% length()
```

Let's identify the non-duplicated genes and only keep the ones that are not duplicated:

```
# Determine the indices for the non-duplicated genes
non_duplicates_idx <- which(duplicated(annotations_ahb$gene_name) == FALSE)

# How many rows does annotations_ahb have?
annotations_ahb %>% nrow()

# Return only the non-duplicated genes using indices
annotations_ahb <- annotations_ahb[non_duplicates_idx, ]

# How many rows are we left with after removing?
annotations_ahb %>% nrow()
```

Finally, it would be good to know **what proportion of the Ensembl identifiers map to an Entrez identifier**:

```
# Determine how many of the Entrez column entries are NA
which(is.na(annotations_ahb$entrezid)) %>%  length()
```

That's more than half of our genes! If we plan on using Entrez ID results for downstream analysis, we should definitely keep this in mind. If you look at some of the Ensembl IDs from our query that returned NA, these map to pseudogenes (i.e ENSG00000265439) or non-coding RNAs (i.e. ENSG00000265425). The discrepancy (which we can expect to observe) between databases is due to the fact that each implements its own different computational approaches for generating the gene builds.

Using AnnotationHub to create our tx2gene file

To create our `tx2gene` file, we would need to use a combination of the methods above and merge two dataframes together. For example:

```
## DO NOT RUN THIS CODE

# Create a transcript dataframe
 txdb <- transcripts(human_ens, return.type = "data.frame") %>%
   dplyr::select(tx_id, gene_id)
 txdb <- txdb[grep("ENST", txdb$tx_id),]

 # Create a gene-level dataframe
 genedb <- genes(human_ens, return.type = "data.frame")  %>%
   dplyr::select(gene_id, gene_name)

 # Merge the two dataframes together
 annotations <- inner_join(txdb, genedb)
```

In this lesson our focus has been using annotation packages to extract information mainly just for gene ID conversion for the different tools that we use downstream. Many of the annotation packages we have presented have much more information than what we need for functional analysis and we have only just scratched the surface here. It's good to know the capabilities of the tools we use, so we encourage you to spend some time exploring these packages to become more familiar with them.

> **NOTE:** The *annotables* package is a super easy annotation package to use. It is not updated frequently, so it's not great for getting the most up-to-date information for the current builds and does not have information for other organisms than human and mouse, but is a quick way to get annotation information.
>
> ```
> # Install package
> BiocManager::install("annotables")
> ```

```
# Load library
library(annotables)

# Access previous build of annotations
grch38
```
