Introduction to utr.annotation package

Yating Liu

August/22/2021

Contents

1. Install utr.annotation package

1.1 Install dependencies

```
cran_pkgs <- c("parallel", "doParallel", "data.table", "readr",
    "stringr", "vcfR", "dplyr", "tidyr", "keras", "devtools",
    "reticulate")
bioc_pkgs <- c("biomaRt", "Biostrings", "AnnotationHub", "ensembldb")

for (pkg in cran_pkgs) {
    if (!(pkg %in% installed.packages())) {
        install.packages(pkg)
    }
```

```
}

if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
}

for (pkg in bioc_pkgs) {
    if (!(pkg %in% installed.packages())) {
        BiocManager::install(pkg)
    }
}
```

Install a couple R packages

```
library(keras)
# Install tensorflow backend
reticulate::install_miniconda()
keras::install_keras(version = "2.2.4", tensorflow = "1.14.0",
    method = "conda")
# install deep learning model data package
devtools::install_bitbucket("jdlabteam/mrl.dl.model")
```

Install keras package and deep learning model for MRL prediction.

1.2 Install utr.annotation package

```
# Install the release version from CRAN
install.packages("utr.annotation")
# Or install the latest version from Bitbucket
devtools::install_bitbucket("jdlabteam/utr.annotation")
```

1.3 Change the environment variable R_MAX_VSIZE

We will extract the database information and store it as an R object, which is larger than the default maximal vector heap size (2 MB). Therefore, we need to set a larger maximal vector heap size. The environment variable R_MAX_VSIZE can be used to specify the maximal vector heap size. You can do this by opening terminal and run the following commands

```
cd ~
touch .Renviron
open .Renviron
Save R_MAX_VSIZE=50Gb as the first line of .Renviron
```

2. Load utr.annotation package

```
library(utr.annotation)
```

3. Run UTR annotation

3.1 Input variant file

utr.annotation accepts variant files in either CSV or VCF format, which must have 4 required columns: Chr, Pos, Ref, Alt. Chr represents the chromosome number with or without “chr” prefix. Pos represents

the variant position in the chromosome. Ref represents the reference sequence. Alt represents the altered sequence. If there are multiple Alt sequence in one row and concatenated with "," (ex: A,C), this row will be separated into multiple rows which have one Alt sequence

3.2 Parameters for runUTRAnnotation function

- variantFile: the full path of the input variant file, which should be in CSV or VCF format and must have 4 required columns: Chr, Pos, Ref, Alt. Chr represents the chromosome number with or without "chr" prefix. Pos represents the variant position in the chromosome. Ref represents the reference sequence. Alt represents the altered sequence.
- annotationResult: the full path of the output annotation file
- species: human or mouse
- ensemblVersion (Optional): by default, we will use the latest Ensembl Version annotation for the specified species. You can specify the version number you'd like to use, for example ensemblVersion = 93
- dataDir (Optional): the path to store the Ensembl database information, if not specified database information will be stored in a new folder named as input variant file name with a "db_" prefix, for example db_variants_sample.csv/ when input variant file name is variants_sample.csv
- conservationBwFiles (Optional): the full path of the folder which contains conservation files in bigWig format. You can download the phastCons and phyloP conservation track files from UCSC Genome Browser and put them in a folder, then specify the conservationBwFiles as the directory of this folder. If you do not specify the conservationBwFiles, we will skip outputting the conservation scores for the variant positions.
- cores (Optional): number of cores to use for parallel computing. If not specified, then use single core by default.
- format (optional) csv or vcf, the default is csv
- mrl_prediction (optional) Whether predict mean ribosome load and check if it increases or decreases. The default is set to TRUE.

3.3 Run runUTRAnnotation on a sample variant file

Here is a sample variant file containing a small subset of variants from An JY et al, 2018. It has 4 columns and 13 variants.

```
# load sample variants
require(readr)
variants <- read_csv(system.file("extdata", "variants_sample.csv",
    package = "utr.annotation"))
# There're 13 variants in total
print(nrow(variants))
#> [1] 13
# The file has four required columns
print(variants)
#> # A tibble: 13 x 4
#>    Chr         Pos Ref   Alt
#>    <chr>     <dbl> <chr> <chr>
#>  1 chr1    1308643 CAT   C
#>  2 chr12  69693767 G     T
#>  3 chr12   4685084 A     G
#>  4 chr15  45691267 A     G
#>  5 chr16  57167173 C     A
#>  6 chr16  71644959 TTTTA T
#>  7 chr17  39065232 G     A
#>  8 chr1    3747728 T     C
#>  9 chr1   38338861 C     A
```

```
#> 10 chr11  75169798 A     C
#> 11 chr11 121055309 G     T
#> 12 chr12  47798924 A     G
#> 13 chr15  38253188 G     A
```

Parameter settings:

- variantFile: path to the input variant file. For example, run UTR annotation on variants in the "variants_sample.csv" file
- annotationResult = path to the output annotation file. For example, the annotated variants will be saved to "annotated_variants_sample.csv"
- species = "human": variants are from human species
- ensemblVersion = 93: will use Ensembl database version 93
- dataDir = "test_db": test_db folder will be created and will be used to store the database information
- conservationBwFiles = Conservation_scores: the conservation scores files in bigWig format were downloaded from UCSC Genome browser and saved in the Conservation_scores folder. There are two conservation files in this folder:
  - Conservation scores for alignments of 99 vertebrate genomes with human, downloaded from: https://hgdownload.soe.ucsc.edu/goldenPath/hg38/phastCons100way/hg38.phastCons100way.bw
  - Basewise conservation scores (phyloP) of 99 vertebrate genomes with human, downloaded from: https://hgdownload.soe.ucsc.edu/goldenPath/hg38/phyloP100way/hg38.phyloP100way.bw
- mrl_prediction = TRUE

  WARNING: **If you are using Mac M1**, mrl_prediction will be set to FALSE internally and MRL prediction will be skipped because tensorflow cannot run on Mac M1 currently.

```
variants_sample <- system.file("extdata", "variants_sample.csv",
    package = "utr.annotation")
# sample conservation bigWig files only include
# conservation scores at the regions of variants in
# variants_sample.csv
Conservation_scores <- system.file("extdata", "Conservation_scores",
    package = "utr.annotation")

runUTRAnnotation(variantFile = variants_sample, annotationResult = "annotated_variants_sample.csv",
    species = "human", ensemblVersion = 93, dataDir = "test_db",
    conservationBwFiles = Conservation_scores, mrl_prediction = TRUE)
```

4. UTR annotation output

Next, we will look into the annotation output from runUTRAnnotation and go over important columns. The detailed explanation on each output column can be found at utr.annotation repository

4.1 Read annotated variants

Annotations on whether and how each variant impact UTR elements are added to the variants table as new columns

```
results <- read_csv(system.file("extdata", "annotated_variants_sample.csv",
    package = "utr.annotation"))
print(colnames(results))
#>  [1] "Chr"                          "Pos"
#>  [3] "Ref"                          "Alt"
#>  [5] "Transcript"                   "transcript_id"
#>  [7] "utr3_transcript_id"           "utr5_transcript_id"
#>  [9] "cds_transcript_id"            "startCodon_transcript_id"
```

```
#> [11] "startCodon_positions"          "stopCodon_transcript_id"
#> [13] "stopCodon_positions"           "kozak_transcript_id"
#> [15] "kozak_positions"               "num_uAUG"
#> [17] "num_uAUG_altered"              "num_kozak"
#> [19] "num_kozak_altered"             "utr_num_uAUG_gainedOrLost"
#> [21] "utr_num_kozak_gainedOrLost"    "num_polyA_signal"
#> [23] "num_polyA_signal_altered"      "num_polyA_signal_gainedOrLost"
#> [25] "stop_codon"                    "stop_codon_altered"
#> [27] "start_codon"                   "start_codon_altered"
#> [29] "kozak"                         "kozak_altered"
#> [31] "lost_start_codon"              "lost_stop_codon"
#> [33] "kozak_score"                   "kozak_altered_score"
#> [35] "tss_kozak_score_gainedOrLost"  "hg38.phastCons100way.bw"
#> [37] "hg38.phyloP100way.bw"          "mrl"
#> [39] "mrl_altered"                   "mrl_gainedOrLost"
```

4.2 Annotations on whether and how the variants impact the important features elements in 5' UTR: start codon, upstream ORFs, upstream Kozak, and mean ribosome load

utr5_transcript_id column shows the Ensembl ids of the transcripts whose 5' regions overlap with the corresponding variant. For example, chr1_1308643_CAT_C overlaps with the 5' UTR region of the transcript ENST00000379031. If a variant overlaps with the UTR regions of multiple transcripts, then these transcript ids will be concatenated. Variants that don't overlap with the 5' UTR regions of one or more transcripts (utr5_transcript_id = NA) will be annotated as NA for all the 5' UTR features related columns that we'll explain below.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "utr5_transcript_id")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   utr5_transcript_id
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     ENST00000379031
#>  2 chr12  69693767 G     T     ENST00000476098
#>  3 chr12   4685084 A     G     ENST00000540688
#>  4 chr15  45691267 A     G     <NA>
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>
#>  7 chr17  39065232 G     A     <NA>
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     <NA>
#> 13 chr15  38253188 G     A     <NA>
```

Whether and how these variants affect start codons *lost_start_codon* column tells whether a variant disrupts start codons of any transcript. For example, the variant chr1_1308643_CAT_C disrupts start codon of one or more transcripts.

- lost_start_codon = NA: the variant doesn't overlap with any transcript's start codon
- lost_start_codon = TRUE: the variant overlaps with some transcripts' start codons and disrupts them
- lost_start_codon = FALSE: the variant overlaps with some transcripts' start codons and doesn't disrupts them

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "lost_start_codon")])
#> # A tibble: 13 x 5
```

```
#>    Chr         Pos Ref   Alt   lost_start_codon
#>    <chr>     <dbl> <chr> <chr> <lgl>
#>  1 chr1    1308643 CAT   C     TRUE
#>  2 chr12  69693767 G     T     NA
#>  3 chr12   4685084 A     G     NA
#>  4 chr15  45691267 A     G     NA
#>  5 chr16  57167173 C     A     NA
#>  6 chr16  71644959 TTTTA T     NA
#>  7 chr17  39065232 G     A     NA
#>  8 chr1    3747728 T     C     NA
#>  9 chr1   38338861 C     A     NA
#> 10 chr11  75169798 A     C     NA
#> 11 chr11 121055309 G     T     NA
#> 12 chr12  47798924 A     G     TRUE
#> 13 chr15  38253188 G     A     TRUE
```

startCodon_transcript_id column show the Ensembl ids of the transcripts whose start codon regions overlap with the corresponding variant. For example, we found the variant chr1_1308643_CAT_C's lost_start_codon is TRUE as showned above, and its startCodon_transcript_id column is ENST00000379031, therefore we can tell the variant chr1_1308643_CAT_C disrupts the start codon of transcript ENST00000379031.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "startCodon_transcript_id")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   startCodon_transcript_id
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     ENST00000379031
#>  2 chr12  69693767 G     T     <NA>
#>  3 chr12   4685084 A     G     <NA>
#>  4 chr15  45691267 A     G     <NA>
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>
#>  7 chr17  39065232 G     A     <NA>
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     ENST00000427332
#> 13 chr15  38253188 G     A     ENST00000299084
```

If you want to look into how a start codon disrupting variant changes the start codon, you can check on *start_codon* and *start_codon_alterd* columns. For example, for the variant chr1_1308643_CAT_C, the start codon sequence on reference allele is "ATG" and on alterative allele is "G".

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "start_codon",
    "start_codon_altered")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   start_codon start_codon_altered
#>    <chr>     <dbl> <chr> <chr> <chr>       <chr>
#>  1 chr1    1308643 CAT   C     ATG         G
#>  2 chr12  69693767 G     T     <NA>        <NA>
#>  3 chr12   4685084 A     G     <NA>        <NA>
#>  4 chr15  45691267 A     G     <NA>        <NA>
#>  5 chr16  57167173 C     A     <NA>        <NA>
#>  6 chr16  71644959 TTTTA T     <NA>        <NA>
#>  7 chr17  39065232 G     A     <NA>        <NA>
```

```
#>  8 chr1    3747728 T     C     <NA>         <NA>
#>  9 chr1   38338861 C     A     <NA>         <NA>
#> 10 chr11  75169798 A     C     <NA>         <NA>
#> 11 chr11 121055309 G     T     <NA>         <NA>
#> 12 chr12  47798924 A     G     ATG          ACG
#> 13 chr15  38253188 G     A     ATG          ATA
```

Whether these variants create new ORFs or disrupt existing ORFs in 5' UTR *utr_num_uAUG_gainedOrLost* shows whether a variant creates new ORFs or disrupt existing ORFs in 5' UTR.

- utr_num_uAUG_gainedOrLost = NA: the variant doesn't overlap with 5' UTR regions of any transcript;
- utr_num_uAUG_gainedOrLost = equal: the variant overlaps with 5' UTR regions of some transcripts, but doesn't impact upstream ORFs
- utr_num_uAUG_gainedOrLost = gained: the variant overlaps with 5' UTR regions of some transcripts, and it creates new upstream ORFs
- utr_num_uAUG_gainedOrLost = lost: the variant overlaps with 5' UTR regions of some transcripts, and it disrupts some upstream ORFs

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "utr_num_uAUG_gainedOrLost")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   utr_num_uAUG_gainedOrLost
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     equal
#>  2 chr12  69693767 G     T     equal
#>  3 chr12   4685084 A     G     equal
#>  4 chr15  45691267 A     G     <NA>
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>
#>  7 chr17  39065232 G     A     <NA>
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     <NA>
#> 13 chr15  38253188 G     A     <NA>
```

If you'd like to know how many upstream ORFs in reference and alterative allels, check out *num_uAUG* and *num_uAUG_altered* columns. If a variant doesn't overlap with the 5' UTR regions of any transcript, these two columns will be NA.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "num_uAUG", "num_uAUG_altered")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   num_uAUG num_uAUG_altered
#>    <chr>     <dbl> <chr> <chr>    <dbl>            <dbl>
#>  1 chr1    1308643 CAT   C            0                0
#>  2 chr12  69693767 G     T            4                4
#>  3 chr12   4685084 A     G            0                0
#>  4 chr15  45691267 A     G           NA               NA
#>  5 chr16  57167173 C     A           NA               NA
#>  6 chr16  71644959 TTTTA T           NA               NA
#>  7 chr17  39065232 G     A           NA               NA
#>  8 chr1    3747728 T     C           NA               NA
#>  9 chr1   38338861 C     A           NA               NA
#> 10 chr11  75169798 A     C           NA               NA
```

```
#> 11 chr11 121055309 G     T           NA                NA
#> 12 chr12  47798924 A     G           NA                NA
#> 13 chr15  38253188 G     A           NA                NA
```

Whether and how the variants impact upstream Kozak sequences The runUTRAnnotation function searches and counts Kozak sequences in 5' UTR. A Kozak is defined as a 7nt sequence [GA]..ATGG. The first position must be G or A, and the last four positions must be ATGG.

utr_num_kozak_gainedOrLost column shows whether a variant creates new Kozak or disrupt existing Kozak in 5' UTR.

- utr_num_kozak_gainedOrLost = NA: the variant doesn't overlap with 5' UTR regions of any transcript;
- utr_num_kozak_gainedOrLost = equal: the variant overlaps with 5' UTR regions of some transcripts, but doesn't impact their upstream Kozak
- utr_num_kozak_gainedOrLost = gained: the variant overlaps with 5' UTR regions of some transcripts, and it creates new upstream Kozak
- utr_num_kozak_gainedOrLost = lost: the variant overlaps with 5' UTR regions of some transcripts, and it disrupts some upstream Kozak

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "utr_num_kozak_gainedOrLost")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   utr_num_kozak_gainedOrLost
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     equal
#>  2 chr12  69693767 G     T     equal
#>  3 chr12   4685084 A     G     equal
#>  4 chr15  45691267 A     G     <NA>
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>
#>  7 chr17  39065232 G     A     <NA>
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     <NA>
#> 13 chr15  38253188 G     A     <NA>
```

If you'd like to know how many upstream Kozaks in reference and alterative allels, check out *num_kozak* and *num_kozak_altered* columns. If a variant doesn't overlap with the 5' UTR regions of any transcript, these two columns will be NA.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "num_kozak", "num_kozak_altered")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   num_kozak num_kozak_altered
#>    <chr>     <dbl> <chr> <chr>     <dbl>             <dbl>
#>  1 chr1    1308643 CAT   C             0                 0
#>  2 chr12  69693767 G     T             0                 0
#>  3 chr12   4685084 A     G             0                 0
#>  4 chr15  45691267 A     G            NA                NA
#>  5 chr16  57167173 C     A            NA                NA
#>  6 chr16  71644959 TTTTA T            NA                NA
#>  7 chr17  39065232 G     A            NA                NA
#>  8 chr1    3747728 T     C            NA                NA
#>  9 chr1   38338861 C     A            NA                NA
#> 10 chr11  75169798 A     C            NA                NA
#> 11 chr11 121055309 G     T            NA                NA
```

```
#> 12 chr12  47798924 A     G           NA                NA
#> 13 chr15  38253188 G     A           NA                NA
```

Whether and how the variants impact ribosome load in the 5' UTR For a variant that overlaps with the 5' UTR of some transcript, we retrieve 100nt 5' UTR sequences upstream of its start codon and use it to predict the mean ribosome load (MRL) with a convolutional neural network (CNN) model trained on human 5' UTR sequences with vary length from 25 to 100nt Sample et al., 2019. If its 5' UTR is shorter than 100nt, we will pad it with "N"s

mrl_gainedOrLost column shows whether the variant changes predicted MRL

- mrl_gainedOrLost = NA: the variant doesn't overlap with 5' UTR regions of any transcript;
- mrl_gainedOrLost = equal: the variant overlaps with 5' UTR regions of some transcripts, but doesn't impact MRL
- mrl_gainedOrLost = gained: the variant overlaps with 5' UTR regions of some transcripts, and it loads more ribosome on alterative alleles
- mrl_gainedOrLost = lost: the variant overlaps with 5' UTR regions of some transcripts, and it loads less ribosome on alterative alleles

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "mrl_gainedOrLost")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   mrl_gainedOrLost
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     equal
#>  2 chr12  69693767 G     T     gained
#>  3 chr12   4685084 A     G     lost
#>  4 chr15  45691267 A     G     <NA>
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>
#>  7 chr17  39065232 G     A     <NA>
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     <NA>
#> 13 chr15  38253188 G     A     <NA>
```

mrl and *mrl_altered* columns show the predicted mean ribosome load (MRL) on referece and alterative alleles

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "mrl", "mrl_altered")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt     mrl mrl_altered
#>    <chr>     <dbl> <chr> <chr> <dbl>       <dbl>
#>  1 chr1    1308643 CAT   C      4.91        4.91
#>  2 chr12  69693767 G     T      4.99        5.20
#>  3 chr12   4685084 A     G      6.71        6.58
#>  4 chr15  45691267 A     G     NA          NA
#>  5 chr16  57167173 C     A     NA          NA
#>  6 chr16  71644959 TTTTA T     NA          NA
#>  7 chr17  39065232 G     A     NA          NA
#>  8 chr1    3747728 T     C     NA          NA
#>  9 chr1   38338861 C     A     NA          NA
#> 10 chr11  75169798 A     C     NA          NA
#> 11 chr11 121055309 G     T     NA          NA
#> 12 chr12  47798924 A     G     NA          NA
#> 13 chr15  38253188 G     A     NA          NA
```

4.3 Annotation on whether and how the variants impact the important features elements in 3' UTR: stop codon and polyA signals

utr3_transcript_id column shows the Ensembl ids of the transcripts whose 3' regions overlap with the corresponding variant. For example, chr17_39065232_G_A overlaps with the 3' UTR region of the transcript ENST00000315392. If a variant overlaps with the UTR regions of multiple transcripts, these transcript ids were concatenated, such as chr15_45691267_A_G. Variants that don't overlap with the 3' UTR regions of one or more transcripts (utr3_transcript_id = NA) will be annotated as NA for all the 3' UTR features related columns that we'll explain below.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "utr3_transcript_id")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   utr3_transcript_id
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     <NA>
#>  2 chr12  69693767 G     T     <NA>
#>  3 chr12   4685084 A     G     <NA>
#>  4 chr15  45691267 A     G     ENST00000260324;ENST00000568606
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     ENST00000393524;ENST00000568954
#>  7 chr17  39065232 G     A     ENST00000315392
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     <NA>
#> 13 chr15  38253188 G     A     <NA>
```

Whether and how these variants affect stop codons *stopCodon_transcript_id* column shows the Ensembl ids of the transcripts whose stop codon regions overlap with the corresponding variant. If a variant doesn't overlap the stop codon regions of any transcripts, its stopCodon_transcript_id is NA. None of the variants in this sample file overlap with stop codon. Their stopCodon_transcript_id column are all NAs.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "stopCodon_transcript_id")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   stopCodon_transcript_id
#>    <chr>     <dbl> <chr> <chr> <lgl>
#>  1 chr1    1308643 CAT   C     NA
#>  2 chr12  69693767 G     T     NA
#>  3 chr12   4685084 A     G     NA
#>  4 chr15  45691267 A     G     NA
#>  5 chr16  57167173 C     A     NA
#>  6 chr16  71644959 TTTTA T     NA
#>  7 chr17  39065232 G     A     NA
#>  8 chr1    3747728 T     C     NA
#>  9 chr1   38338861 C     A     NA
#> 10 chr11  75169798 A     C     NA
#> 11 chr11 121055309 G     T     NA
#> 12 chr12  47798924 A     G     NA
#> 13 chr15  38253188 G     A     NA
```

lost_stop_codon column tells whether a variant disrupts stop codon of any transcript. Since none of the variants overlap with stop codons, their lost_stop_codon will be all NAs.

- lost_stop_codon = NA: the variant doesn't overlap with any transcript's stop codon
- lost_stop_codon = TRUE: the variant overlaps with some transcripts' stop codons and disrupts them

- lost_stop_codon = FALSE: the variant overlaps with some transcripts' stop codons and doesn't disrupts them

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "lost_stop_codon")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   lost_stop_codon
#>    <chr>     <dbl> <chr> <chr> <lgl>
#>  1 chr1    1308643 CAT   C     NA
#>  2 chr12  69693767 G     T     NA
#>  3 chr12   4685084 A     G     NA
#>  4 chr15  45691267 A     G     NA
#>  5 chr16  57167173 C     A     NA
#>  6 chr16  71644959 TTTTA T     NA
#>  7 chr17  39065232 G     A     NA
#>  8 chr1    3747728 T     C     NA
#>  9 chr1   38338861 C     A     NA
#> 10 chr11  75169798 A     C     NA
#> 11 chr11 121055309 G     T     NA
#> 12 chr12  47798924 A     G     NA
#> 13 chr15  38253188 G     A     NA
```

If you want to look into how a stop codon disrupting variant changes the stop codon, you can check on *stop_codon* and *stop_codon_altered* columns.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "stop_codon", "stop_codon_altered")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   stop_codon stop_codon_altered
#>    <chr>     <dbl> <chr> <chr> <lgl>      <lgl>
#>  1 chr1    1308643 CAT   C     NA         NA
#>  2 chr12  69693767 G     T     NA         NA
#>  3 chr12   4685084 A     G     NA         NA
#>  4 chr15  45691267 A     G     NA         NA
#>  5 chr16  57167173 C     A     NA         NA
#>  6 chr16  71644959 TTTTA T     NA         NA
#>  7 chr17  39065232 G     A     NA         NA
#>  8 chr1    3747728 T     C     NA         NA
#>  9 chr1   38338861 C     A     NA         NA
#> 10 chr11  75169798 A     C     NA         NA
#> 11 chr11 121055309 G     T     NA         NA
#> 12 chr12  47798924 A     G     NA         NA
#> 13 chr15  38253188 G     A     NA         NA
```

Whether and how the variants impact polyA signals in 3' UTR The function runUTRAnnotation searches and counts polyA signals in the 3' UTRs. A polyA signal is defined as a 6nt sequence AATAAA or ATTAAA.

num_polyA_signal_gainedOrLost shows whether a variant creates new PolyA signals or disrupt existing PolyA signals in 3' UTR. * num_polyA_signal_gainedOrLost = NA: the variant doesn't overlap with the 3' UTR regions of any transcript; * num_polyA_signal_gainedOrLost = equal: the variant overlaps with the 3' UTR regions of some transcripts, but doesn't impact their polyA signals * num_polyA_signal_gainedOrLost = gained: the variant overlaps with the 3' UTR regions of some transcripts, and it creates new polyA signals. For example, chr17_39065232_G_A. * num_polyA_signal_gainedOrLost = lost: the variant overlaps with 3' UTR regions of some transcripts, and it disrupts some polyA signal. For example, chr15_45691267_A_G and chr16_71644959_TTTTA_T.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "num_polyA_signal_gainedOrLost")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   num_polyA_signal_gainedOrLost
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     <NA>
#>  2 chr12  69693767 G     T     <NA>
#>  3 chr12   4685084 A     G     <NA>
#>  4 chr15  45691267 A     G     lost
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     lost
#>  7 chr17  39065232 G     A     gained
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     <NA>
#> 13 chr15  38253188 G     A     <NA>
```

If you'd like to know how many polyA signals in reference and alterative allels, check out *num_polyA_signal* and *num_polyA_signal_altered* columns. If a variant doesn't overlap with the 3' UTR regions of any transcript, these two columns will be NA.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "num_polyA_signal",
    "num_polyA_signal_altered")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   num_polyA_signal num_polyA_signal_altered
#>    <chr>     <dbl> <chr> <chr> <chr>            <chr>
#>  1 chr1    1308643 CAT   C     <NA>             <NA>
#>  2 chr12  69693767 G     T     <NA>             <NA>
#>  3 chr12   4685084 A     G     <NA>             <NA>
#>  4 chr15  45691267 A     G     2;2              1;1
#>  5 chr16  57167173 C     A     <NA>             <NA>
#>  6 chr16  71644959 TTTTA T     7;7              6;6
#>  7 chr17  39065232 G     A     8                9
#>  8 chr1    3747728 T     C     <NA>             <NA>
#>  9 chr1   38338861 C     A     <NA>             <NA>
#> 10 chr11  75169798 A     C     <NA>             <NA>
#> 11 chr11 121055309 G     T     <NA>             <NA>
#> 12 chr12  47798924 A     G     <NA>             <NA>
#> 13 chr15  38253188 G     A     <NA>             <NA>
```

4.4 Annotations on whether and how the variants impact the important features elements in Kozak sequence at translation start site (TSS)

A TSS Kozak of a transcript is defined as a 8nt sequence, the first three nuleotides are the last three nuleotides of its 5' UTR sequence and the next five nuleotides are the first five nuleotides of its coding sequence.

kozak_transcript_id column shows the Ensembl ids of the transcripts whose Kozak regions overlap with the corresponding variant. If a variant doesn't overlap the TSS Kozak regions of any transcripts, its kozak_transcript_id is NA. Here we can see chr1_1308643_CAT_C and chr12_4685084_A_G overlap with the TSS Kozak region of ENST00000379031 and ENST00000540688, respectively.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "kozak_transcript_id")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   kozak_transcript_id
```

```
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     ENST00000379031
#>  2 chr12  69693767 G     T     <NA>
#>  3 chr12   4685084 A     G     ENST00000540688
#>  4 chr15  45691267 A     G     <NA>
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>
#>  7 chr17  39065232 G     A     <NA>
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     ENST00000427332
#> 13 chr15  38253188 G     A     ENST00000299084
```

kozak and *kozak_altered* columns show the TSS Kozak sequence on reference and alterative alleles

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "kozak", "kozak_altered")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   kozak    kozak_altered
#>    <chr>     <dbl> <chr> <chr> <chr>    <chr>
#>  1 chr1    1308643 CAT   C     GCCATGAG GCCGAG
#>  2 chr12  69693767 G     T     <NA>     <NA>
#>  3 chr12   4685084 A     G     GAGATGGA GGGATGGA
#>  4 chr15  45691267 A     G     <NA>     <NA>
#>  5 chr16  57167173 C     A     <NA>     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>     <NA>
#>  7 chr17  39065232 G     A     <NA>     <NA>
#>  8 chr1    3747728 T     C     <NA>     <NA>
#>  9 chr1   38338861 C     A     <NA>     <NA>
#> 10 chr11  75169798 A     C     <NA>     <NA>
#> 11 chr11 121055309 G     T     <NA>     <NA>
#> 12 chr12  47798924 A     G     CCCATGGA CCCACGGA
#> 13 chr15  38253188 G     A     AAGATGAG AAGATAAG
```

tss_kozak_score_gainedOrLost column shows whether a variant alters Kozak score in translation start site (TSS). Kozak score is the PWM score on TSS Kozak PWM which is generated with top 20 Kozak sequences with the least ribosome load in the upstream, from Sample et al., 2019

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "tss_kozak_score_gainedOrLost")])
#> # A tibble: 13 x 5
#>    Chr         Pos Ref   Alt   tss_kozak_score_gainedOrLost
#>    <chr>     <dbl> <chr> <chr> <chr>
#>  1 chr1    1308643 CAT   C     lost
#>  2 chr12  69693767 G     T     <NA>
#>  3 chr12   4685084 A     G     lost
#>  4 chr15  45691267 A     G     <NA>
#>  5 chr16  57167173 C     A     <NA>
#>  6 chr16  71644959 TTTTA T     <NA>
#>  7 chr17  39065232 G     A     <NA>
#>  8 chr1    3747728 T     C     <NA>
#>  9 chr1   38338861 C     A     <NA>
#> 10 chr11  75169798 A     C     <NA>
#> 11 chr11 121055309 G     T     <NA>
#> 12 chr12  47798924 A     G     lost
```

```
#> 13 chr15  38253188 G     A     lost
```

kozak_score and *kozak_altered_score* column show the Kozak score on reference and alternative alleles

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "kozak_score",
    "kozak_altered_score")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   kozak_score kozak_altered_score
#>    <chr>     <dbl> <chr> <chr>       <dbl>               <dbl>
#>  1 chr1    1308643 CAT   C           0.909               0
#>  2 chr12  69693767 G     T          NA                  NA
#>  3 chr12   4685084 A     G           0.904               0.855
#>  4 chr15  45691267 A     G          NA                  NA
#>  5 chr16  57167173 C     A          NA                  NA
#>  6 chr16  71644959 TTTTA T          NA                  NA
#>  7 chr17  39065232 G     A          NA                  NA
#>  8 chr1    3747728 T     C          NA                  NA
#>  9 chr1   38338861 C     A          NA                  NA
#> 10 chr11  75169798 A     C          NA                  NA
#> 11 chr11 121055309 G     T          NA                  NA
#> 12 chr12  47798924 A     G           0.832               0.683
#> 13 chr15  38253188 G     A           0.855               0.706
```

4.5 Whether the variants overlap highly conserved regions

If you provide conservation files when execute runUTRAnnotation function, you will get the conservation scores of the variant regions. The conservation column names are the same as the names of the cooresponding conservation files in the specified conservation files path. For example, here we have two files in the Conservation_scores folder: hg38.phastCons100way.bw and hg38.phyloP100way.bw. hg38.phastCons100way.bw column shows the conservation scores at variant positions from hg38.phastCons100way.bw file. hg38.phyloP100way.bw column shows the conservation scores at variant positions from hg38.phyloP100way.bw. Variants overlapped with highly conserved regions could potentially have more biological impacts. For example, chr12_69693767_G_T is overlapped with a highly conserved position. At chr12 position 69,693,767, the phastCons is 1 and phyloP is 6.806. The conservation scores for a deletion will concatenate scores at each position with ";". For example, chr1_1308643_CAT_C will concatenate conservation scores on chr1 at position 1308643, 1308644, and 1308645.

```
print(results[, c("Chr", "Pos", "Ref", "Alt", "hg38.phastCons100way.bw",
    "hg38.phyloP100way.bw")])
#> # A tibble: 13 x 6
#>    Chr         Pos Ref   Alt   hg38.phastCons100way.~ hg38.phyloP100way.bw
#>    <chr>     <dbl> <chr> <chr> <chr>                  <chr>
#>  1 chr1    1308643 CAT   C     0.002;0.58;0.894       -0.894;0.787;1.239
#>  2 chr12  69693767 G     T     1                      6.806
#>  3 chr12   4685084 A     G     0.008                  0.641
#>  4 chr15  45691267 A     G     1                      2.649
#>  5 chr16  57167173 C     A     1                      1.849
#>  6 chr16  71644959 TTTTA T     1;1;1;0.999;0.999      2.48;2.607;4.156;0.496;0.~
#>  7 chr17  39065232 G     A     0.042                  0.421
#>  8 chr1    3747728 T     C     0.003                  0.016
#>  9 chr1   38338861 C     A     0                      -0.412
#> 10 chr11  75169798 A     C     0.002                  1.055
#> 11 chr11 121055309 G     T     1                      9.999
#> 12 chr12  47798924 A     G     1                      3.369
#> 13 chr15  38253188 G     A     1                      6.916
```

5. Run annotation on a large number of variants

5.1 Query ensembl database information and cache it

The time spending on query information from Ensembl database will depend on internet speed and Ensembl server, so we ran the query step with initUTRAnnotation function and cached the database information. runUTRAnnotation will use the cached database information to do annotation and the runtimes don't include database query time.

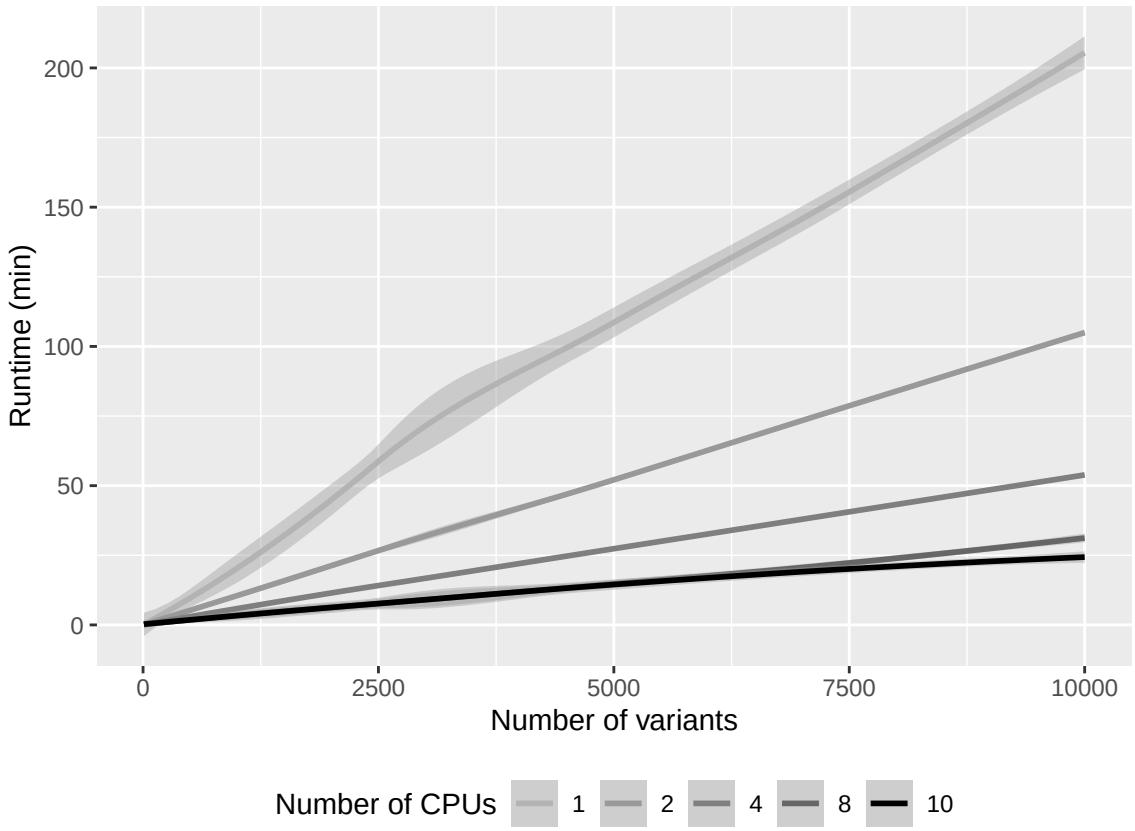
```
initUTRAnnotation(variantFile = "../benckmark/data/asd_10000.csv",
    species = "human", ensemblVersion = 93, dataDir = "../benckmark/db_1000_vars")

var_num <- c(10, 100, 250, 500, 1000, 2500, 5000, 7500, 10000)
cores <- c(1, 2, 4, 8, 10)
replicates <- c(1, 2, 3)
runtimes <- data.frame()
for (i in replicates) {
    for (core in cores) {
        for (num in var_num) {
            message("varfile = ", num, " core = ", core, " rep = ",
                i)
            res <- tempfile()
            varfile <- file.path("benchmark_htcf/data", paste0("asd_",
                num, ".csv"))
            t <- system.time(runUTRAnnotation(variantFile = varfile,
                annotationResult = res, species = "human", ensemblVersion = 100,
                dataDir = "data/db_GTEX_eOutlier_rare_variants",
                conservationBwFiles = "/tmp/conservation_tracks",
                cores = core))
            runtimes <- rbind(runtimes, data.frame(num_vars = num,
                replicate = i, core = core, runtime = t[["elapsed"]]))
        }
    }
}
saveRDS(runtimes, "benchmark_htcf/results/runtimes_upto_10000_vars.rds")
```

5.2 Runtime on up to 10,000 variants with different number of CPUs

The runtime of the runUTRAnnotation function on up to 10,000 variants with different number of CPUs are shown as below. We can see the runtime of the UTR annotation is linearly correlated with the number of variants. We ran the query step with initUTRAnnotation function and cached the database information on 10,000 variants like above. runUTRAnnotation will use the cached database information to do annotation and the runtimes don't include database query time.

```
require(ggplot2)
diff_num_vars <- readRDS(system.file("extdata", "runtimes_upto_10000_vars.rds",
    package = "utr.annotation"))
ggplot(diff_num_vars, aes(x = num_vars, y = runtime/60, group = core,
    color = factor(core))) + scale_color_manual(breaks = c("1",
    "2", "4", "8", "10"), values = c("grey70", "grey60", "grey50",
    "grey40", "black")) + geom_smooth() + labs(y = "Runtime (min)",
    color = "Number of CPUs", x = "Number of variants") + theme(legend.position = "bottom")
```



5.3 Partition the variants into smaller files using partitionVariantFile function

If we want to annotate a large number of variants (100,000 or more), we can leverage on high performance computers. First, partition the variant file into smaller partitions, run annotation on each partitions with multiple CPUs in parallel, in the end concatenate all annotations together into one annotation file.

There're two ways of partitioning:

1. specify the number of partitions with "chunkNum" to split the large file into a certain number of partitions For example, here we specify chunkNum = 3, chunkPath = "chunks_3" in partitionVariantFile function to split the variant file "variants_sample.csv" into 3 files with equal number of variants. As 13 is not dividable by 3, the first and second partitions will have 3 variants, and the third one will have 4 variants. Those partition files will be output to "chunks_3" folder. There're three partition files (CSV) in chunks_3 folder. "lookup.tab" is a text file listing all partition file names, which would be helpful for running array jobs on the cluster.

```
partitionVariantFile(variantFile = system.file("extdata", "variants_sample.csv",
    package = "utr.annotation"), chunkNum = 3, chunkPath = "chunks_3",
    species = "human", ensemblVersion = 93, dataDir = "db_all_variants")
# print(list.files('chunks_3', pattern = '*.csv'))
```

2. specify the number of variants in each partition with chunkSize to split the large file into partition files, each of which has a certain number of variants in it. For example, here we specify chunkSize = 7, chunkPath = "chunks_7_vars" in partitionVariantFile function to split the variant file "variants_sample.csv" into files with at most 7 variants. As 13 is not dividable by 7, the first partition will have 7 variants, and the second one will have 6 variants. Those partition files will be output to "chunks_7_vars" folder

```
partitionVariantFile(variantFile = system.file("extdata", "variants_sample.csv",
    package = "utr.annotation"), chunkSize = 7, chunkPath = "chunks_7_vars",
    species = "human", ensemblVersion = 93, dataDir = "db_all_variants")
```

Why is it important to set dataDir in partition step? partitionVariantFile function will query the Ensembl database to get the gene annotation, 5' UTR sequences, 3' UTR sequences and coding sequences of all transcripts which overlap with any variant in the original variant file before splitting the variant file. Then we will run annotation on partition files with runUTRAnnotation function. It will be time consuming and not necessary to query the Ensembl database for each partition file, especially when we have thousands of those partition files. By specifying the dataDir in the partitionVariantFile step, and use the same dataDir in the runUTRAnnotation, code will look into your dataDir and use the database information directly for all partition. Based on above code, we stored database information for all 13 variants in "db_all_variants" folder. Then we can specify dataDir = "db_all_variants" in runUTRAnnotation function when run annotation on each partition file

After running the above example script, we retrieved and stored database information inside db_all_variants. In db_all_variants/human/93, there're bioMart databse object, 5' UTR sequence, 3' UTR sequences, transcript regions that runUTRAnnotation will use when annotate individual partitions instead of querying them again.

```
partitionVariantFile(variantFile = system.file("extdata", "variants_sample.csv",
    package = "utr.annotation"), chunkSize = 7, chunkPath = "chunks_7_vars_v2",
    species = "human", ensemblVersion = 93, dataDir = "db_all_variants",
    getTranscript = FALSE)
```

getTranscript parameter is used to specify whether to get ids of the transcripts that overlap with all the variants during the partition. The default value is TRUE. If the number of variants is too large (for example $>$ 100,000), set it to FALSE because it is time consuming.

5.4 Run annotation on all partition files

Run annotation on each partition files in "chunks_3", and output annotation for each of these variant files to "partition_results" folder.

```
partitions <- list.files("chunks_3", pattern = "*.csv")
print(partitions)
for (f in partitions) {
    runUTRAnnotation(variantFile = file.path("chunks_3", f),
        annotationResult = file.path("partition_results", f),
        species = "human", ensemblVersion = 93, dataDir = "db_all_variants",
        mrl_prediction = F)
}
```

On high performance cluster, we can submit an array jobs, each job will run annotation on one partition file. We can allocate 4 CPUs for each job, and the array of jobs will be running in parallel.

Here's an example UTRAnnotation.R

```
library(utr.annotation)
args = commandArgs(trailingOnly = TRUE)

if (length(args) >= 5) {
    input <- args[1]
    output <- args[2]
    species <- args[3]
```

```
    version <- args[4]
    dbDir <- args[5]
    if (length(args) >= 6) {
        conservationBwTracks <- args[6]
        numCores <- args[7]
        message("Run UTR annotation on ", input, "\nspecies = ",
            species, "\nensembl version = ", version, " with conservation files ",
            paste(conservationBwTracks, collapse = ","))
        runUTRAnnotation(input, output, species = species, ensemblVersion = version,
            dataDir = dbDir, conservationBwFiles = conservationBwTracks,
            cores = numCores)

    } else {
        message("Run UTR annotation on ", input, "\nspecies = ",
            species, "\nensembl version = ", version, " without conservation files")
        runUTRAnnotation(input, output, species = species, ensemblVersion = version,
            dataDir = dbDir)
    }
} else {
    stop("Please pass at least 5 arguments, CSV input file, output CSV file, species, ensembl version, a
}
```

Here is an example Slurm script executing UTRAnnotation.R on all partition variant files in parallel. Each job will be running on 4 CPUs.

```
#!/bin/bash
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=5G

ml R
read idx filename< <( sed -n ${SLURM_ARRAY_TASK_ID}p "chunks_3/lookup.tab" )
Rscript UTRAnnotation.R chunks_3/$filename partition_results/annotated_$filename human 93 db_all_va
```

5.5 Concatenate annotated results into one annotation file using concatenateAnnotationResult function

Use varResultPath = "partition_results" to specify the folder of individual annotations and annotationFinalResult = "results/concatenated_annotation.csv" to specify the full path of the final output. The final concatenated annotation file containing annotation for all variants will be output to "results/concatenated_annotation.csv"

```
concatenateAnnotationResult(varResultPath = "partition_results",
    annotationFinalResult = "results/concatenated_annotation.csv")
# results <-
# read_csv('results/concatenated_annotation.csv')
# print(nrow(results)) print(colnames(results))
```