Using AssessORF

Deepank Korandla

19 May 2021

Package

AssessORF 1.11.0

Contents

1 Introduction

In genomics pipelines, a newly sequenced genome must be annotated before it can be analyzed and compared to other genomes. One of the crucial first steps in genome annotation is identification of gene boundaries within the genome, also known as gene finding or gene prediction. However, gene prediction is not an error-free process and errors can propagate in downstream analyses. There are three common types of errors in gene finding: a real gene is missing entirely from the predicted set, a gene is included in the predicted set that is not real, and the wrong start is chosen for a real gene. Errors in choosing the correct start site are the most frequent because often times there are multiple possible starts for a gene and it can be difficult to determine which one is right. When there are multiple possible starts for a gene, *ab initio* gene prediction programs are limited in that they can only use information from the sequence of the genome to pick the best start of the gene. Such programs utilize heuristics and scoring systems to come to a solution, but the chosen start for a gene might not be the start used *in vivo*. `AssessORF` serves as a tool for assessing the quality of gene predictions and gene prediction programs, utilizing proteomics data and evolutionary conservation data as forms of evidence.

1.1 Why Proteomics?

One of the direct products of coding genes are proteins, so mapping real protein fragments back to the corresponding genome can provide a rough guide as to where the genes are within a genome. In other words, aligning peptide sequences to a genome can prove which genes from a set of predicted genes are definitively real as well as show which genes have starts that do not agree with the protein evidence. Protein evidence contradicts a gene start specifically when there are protein hits in the same open reading frame (ORF) that are directly upstream of or overlapping the start. Additionally, protein evidence in ORFs without a predicted gene can point towards the existence of potential genes missed by the gene finding program. The data from proteomics experiments focused on sequencing the complete set of proteins of an organism is thus highly useful in assessing the quality of a set of predicted genes for that organism.

1.2 Why Evolutionary Conservation?

Closely related strains usually have similar genomic sequences, and the relative positions of genes across closely related strains are conserved. This means that the start positions for genes in one strain's genome are expected to align with the start positions for genes in a closely related strain's genome. Aligning genomes from related strains to a central genome (the genome of the organism of interest) therefore provides a way to determine whether or not the right decision was made in choosing the start for each predicted gene. Starts for genes in highly conserved areas of the central genome should also be highly conserved in order to be considered a good start. Additionally, as described above, there can be multiple possible start sites for a gene. If the chosen start for a gene is significantly more conserved than neighboring starts, then that provides strong evidence towards the correctness of that gene.

The conservation of stop codons across genomes can provide information about the correctness of a gene as well, albeit in a way that is different from the conservation of starts. There is almost always only one stop possible for a gene because the stop codon serves as the termination signal when the gene is translated into protein. Therefore, the strength of conservation of the given stop for a gene is not informative since there are no other possible (in-frame) stops upstream of the given stop. It is possible, however, for the related genomes

to all have a position with a stop codon that aligns to a position in the central genome that does not have stop. For the purposes of this package, that position in the central genome is defined as being associated with a conserved stop. If a codon in the middle of a gene in the central genome has a high degree of stop codon conservation, then that is a sign that at least the region of the gene upstream of the conserved stop does not code for protein. If there are no strong conserved starts downstream of the conserved stop that could serve as the correct start site of the gene, then it is likely that the gene is an over-prediction and should not exist.

2 Package Structure

Assessing the quality of a set of computationally-derived, *ab initio* gene predictions for a genome requires external data to be aligned to that genome before judgments on quality can be made. `AssessORF` follows this methodology, and once external data has been acquired and set up (see the "Getting the Data" section below), usage of the package typically happens in two steps:

1. Map and align the provided evidence, which can either be proteomics data, evolutionary conservation data, or both, back to the central genome.

2. Add in gene predictions for that central genome and categorize how much evidence there is supporting or against each gene in that set.

Each of these steps is implemented by a key function of the package, and each of those two functions returns an object that contains the outcome of the corresponding step. The function `MapAssessmentData` performs the first step (mapping and alignment) and returns a "mapping" object. Within R, mapping objects are structured as lists and are of class `Assessment` and subclass `DataMap`. The function `AssessGenes` performs the second step (gene categorization) by taking in a mapping object and gene predictions and returning a "results" object. Results objects are also structured as lists and are of class `Assessment` and subclass `Results`. `AssessORF` provides functions for viewing and visualizing the data stored in both mapping and results objects, and the `Assessment` help documentation has detailed explanations of what is in each object and methods associated with each object.

It is important to note that the mapping object built for a particular genome is not dependent upon the gene boundaries chosen for that genome, allowing the same mapping object to be used to assess the quality of gene predictions from multiple programs. This is especially useful because generating a mapping object, even when using proteomics data or evolutionary conservation data by themselves, is a long process.

Building off of these concepts, `AssessORF` has a corresponding data package, `AssessORFData`, that provides mapping objects that have already been built and saved for a set of 20 strains. `AssessORFData` also provides multiple results objects for each of those strains, and each results object for a strain uses a set of gene annotations from a different source.

The next couple of sections will describe how to use `AssessORF` from putting together the data to analyzing the results.

3 Installation

In order to install the package, follow these steps:

1. Install the latest version of R using CRAN.
2. Install `AssessORF` in R by running the following commands:

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

BiocManager::install("AssessORF")
```

3. Optionally, install the corresponding data package, `AssessORFData`, using the following commands:

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

BiocManager::install("AssessORFData")
```

4 Getting the Data

The central genome should be in FASTA format or GenBank format.

4.1 Proteomics

For this package, proteomics data should be given as a set of peptide sequences along with their associated confidence scores. This data is usually generated in the latter stages of the proteomics pipeline, following database searching of the mass spectra and statistical analysis. Additionally, the data should come from experiments focused on sequencing the complete proteome of the organism. Proteomics datasets are available online through websites such as ProteomeXchange but may require additional steps before use with the package. For example, the ProteomeXchange serves as a repository for mass spectra data from proteomics experiments, and this data must first be searched against a database and analyzed before use with the package.

4.2 Evolutionary Conservation

Evolutionary conservation is determined by aligning genomes from closely related organisms to the central genome. This allows determination of how often sections in the central genome are covered by syntenic matches to related genomes and how often positions within those matching blocks correspond to start codons (ATG, GTG, TTG) in both genomes (central and related). Start codons in the central genome may be highly covered by syntenic matches to related genomes, but the positions of the start codons in the central genome may not always line up with start codons in other genomes.

Related genomes should be genomes from strains that are closely related to the strain of the central genome. In most cases, using the set of non-partial genomes from the same genus will work best. However, if the number of available genomes for that genus is too small (less than a couple hundred), it may be necessary to use all genomes from a higher taxonomic rank or include closely related genera from the same family in the set.

To compile the set of related genomes, I recommend downloading links to their sequences from the Prokaryotes section of NCBI's Genome Browser using the following protocol:

1. Search for the taxon.
2. Exclude partial genomes (this is an option in the "Partial" section of the "Filters" menu).
3. Download the selected records to a CSV file.

There may also may be rare instances where there are too many genomes for a particular species (i.e. the 8352 genomes for *Streptococcus pneumoniae*). In those cases, remove all genomes for that species from the CSV file and replace them with only genomes for that species that are complete on the assembly level (this is an option in the "Assembly Level" section of the "Filters" menu).

5 Creating the Mapping Object

After gathering the necessary data, it is time to prepare them for use with the mapping function. For the proteomics data, put the peptide sequence into one vector and their confidence scores (if available) into a second vector. The two vectors should have the same length and have the proteomics hits in the same order (i.e. the score for the xth sequence in the sequence vector is at index x in the score vector).

For the central genome and the related genomes, use the DECIPHER package to put them into a SQL database. The example below describes the process for putting genomes into a database in situations where NCBI's Genome Browser is used as the source for related genomes. Users are encouraged to adapt this workflow to suit their own needs.

```
library(DECIPHER)

## Path to the SQL database file (will be created if necessary)
## In this example, a temporary file will be used, but it is recommended that
## the user specify a file path they prefer to use as the location of the
## database instead.
databaseFile <- tempfile()
## Here is what the alternate option would like:
# databaseFile <- "<insert file path to database here>"

## Path to the file containing the links to the related genomes
## In this example, the file comes from NCBI's genome browser site and is in
## CSV format. If the user is also using a CSV file downloaded from NCBI's
## genome browser site as the source for the related genomes links, the user
## can replace the 'system.file' function call below with  the path to their
## CSV file instead. The rest of this example assumes that the links are
## provided in this format. For help on adding genome sequences provided in
## other formats to a database, consult DECIPHER's manual and vignettes.
relGenomesFile <- system.file("extdata",
                              "AdenoviridaeGenomes.csv",
                              package = "AssessORF")
## Here is what the alternate option would like:
# relGenomesFile <- "<insert path to CSV file here>"

## Extract the FTP links from the CSV file and make sure they are in the right
## format so they can be downloaded from the NCBI server. In order to minimize
## the time spent running the example, only the first 13 rows of the table are
## used. The user should drop the '[1:13, ]' part if they plan on using this
## example as a starting point for putting sequences into a database.
genomesTable <- read.csv(relGenomesFile, stringsAsFactors = FALSE)[1:13, ]
ftps <- genomesTable$GenBank.FTP
ftps <- paste(ftps, paste0(basename(ftps), "_genomic.fna.gz"), sep="/")
ftps <- ftps[(substring(ftps, 1, 6) == "ftp://")]

## In this example, the first genome in the table is the central genome.
## In most user scenarios however, the path to the central genome will not be in
## the related genomes file. The user should instead specify the path to the
## file containing the sequence of the central genome (in FASTA format) and
## append that file path to the start of the vector containing the FTP links.
```

```
## Here is what that would look like:
# genomeFile <- "<insert file path to genome here>"
# ftps <- c(genomeFile, ftps)

## This vector will hold which genomes were succesfully added to the database.
pass <- logical(length(ftps))

## Add the sequences to the database.
## The loop can cause timeout errors if there is no internet connection
## when the package is built so it has been commented out so the example
## runs smoothly. Users should uncomment this loop when adapting the
## example for their own use.
# for (pIdx in seq_along(pass)) {
#   t <- try(Seqs2DB(ftps[pIdx], "FASTA", databaseFile, as.character(pIdx),
#                    compressRepeats=TRUE, verbose=FALSE),
#            silent=TRUE)
#
#   if (!(is(t, "try-error"))) {
#     pass[pIdx] <- TRUE
#   }
# }

## This vector contains the list of identifers for the genomes in the database,
## based on which ones were successfully added. Identifers are character
## strings. The first one, identifier "1", corresponds to the central genome.
## The remaining identifers, in this case "2":"13", correspond to the related
## genomes.
identifiers <- as.character(which(pass))
```

In the example above, a set of Adenoviridae genomes from NCBI was used as the source for both the central genome and the related genomes. Human adenovirus 1 is the strain of interest here, and its genome serves the central genome (identifier 1). The remaining genomes in the set serve as the related genomes (identifiers 2 - 13). Please note that the package is only intended to be used with prokaryotes. Viruses are used in some examples in the package because manipulations with viral genomes run much faster than those with prokaryotic genomes (due to viral genomes being much smaller in size).

The next example shows how to use the `MapAssessmentData` function with the SQL database (and database identifiers) generated in the previous example. Since there is no proteomics data in this instance, `useProt` will be set to `FALSE`.

```
library(AssessORF)

## Reminder: the first identifier in the database, in this case identifier "1",
## corresponds to the central genome. The remaining identifers, in this case
## "2":"13", correspond to the related genomes.
myMapObj <- MapAssessmentData(databaseFile,
                              central_ID = identifiers[1],
                              related_IDs = identifiers[-1],
                              speciesName = "Human adenovirus 1",
                              useProt = FALSE)
## Distance from related genomes to central genome (i.e. strain's genome) measured.
```

```
## Most distant related genomes selected. Beginning evolutionary conservation mapping.
## Evolutionary conservation mapped.
```

```
## Remember to use 'unlink' to remove a database once it is no longer needed.
unlink(databaseFile)
```

The following code chunk shows a generalized call to the `MapAssessmentData` function, outlining which parameters users should specify when working with both evolutionary conservation data and proteomics data.

```
myMapObj <- MapAssessmentData(databaseFile, ## File path to the SQL database containing the genomes
                              central_ID = identifiers[1], ## Identifier for the central genome
                              related_IDs = identifiers[-1], ## Identifers for the related genomes
                              protHits_Seqs = protSeqs, ## Sequences for the proteomics hits
                              protHits_Scores = protScores, ## Confidence scores for the proteomics hits
                              strainID = strain, ## The identifer for the strain
                              speciesName = species) ## The name of the species
```

6 Generating the Results Object

1. Acquire a set of genes, either from a gene prediction program or from an online database with annotations, for the strain of interest.
   - Make sure that the same genome sequence is used to generate both the mapping object and set of genes to assess.
2. Parse genes into the three vectors: left boundaries, right boundaries, and strand information.
   - This requires an understanding of how the output from the gene source is structured and will vary from program to program.
   - Strand information must be given as a single character and must either be "+" or "−". "+" refers to the forward strand, the given sequence for the central genome. "−" refers to the reverse strand, the reverse complement of the given sequence for the central genome.
   - The left boundary positions and the right boundary positions of the genes must be in forward strand terms, i.e. where they would be the 5' to 3' reading direction of the forward strand. For genes on the forward strand, this means that the left boundaries correspond to the start of the gene and the right boundaries correspond to the stop of the gene. For genes on the reverse strand, this means that the left boundaries correspond to the stop of the gene and the right boundaries correspond to the start of the gene.
   - Position one of a gene (according to the left and right boundaries) must correspond to the first nucleotide of the gene and not the first nucleotide before the gene (i.e the zero-th nucleotide).
3. Use the 'AssessGenes' function with the mapping object and the set of predicted genes to generate the results object.

Below is an example of how to use the `AssessGenes` function, using the pre-saved mapping object for *Streptococcus pyogenes* MGAS5005 and the corresponding set of Prodigal-predicted genes.

```
currMapObj <- readRDS(system.file("extdata",
                                  "MGAS5005_PreSaved_DataMapObj.rds",
                                  package = "AssessORF"))

currProdigal <- readLines(system.file("extdata",
                                      "MGAS5005_Prodigal.sco",
                                      package = "AssessORF"))[-1:-2]

prodigalLeft <- as.numeric(sapply(strsplit(currProdigal, "_", fixed=TRUE), `[`, 2L))
prodigalRight <- as.numeric(sapply(strsplit(currProdigal, "_", fixed=TRUE), `[`, 3L))
prodigalStrand <- sapply(strsplit(currProdigal, "_", fixed=TRUE), `[`, 4L)

currResObj <- AssessGenes(geneLeftPos = prodigalLeft,
                          geneRightPos = prodigalRight,
                          geneStrand = prodigalStrand,
                          inputMapObj = currMapObj,
                          geneSource = "Prodigal")
```

7 Viewing the Results Object

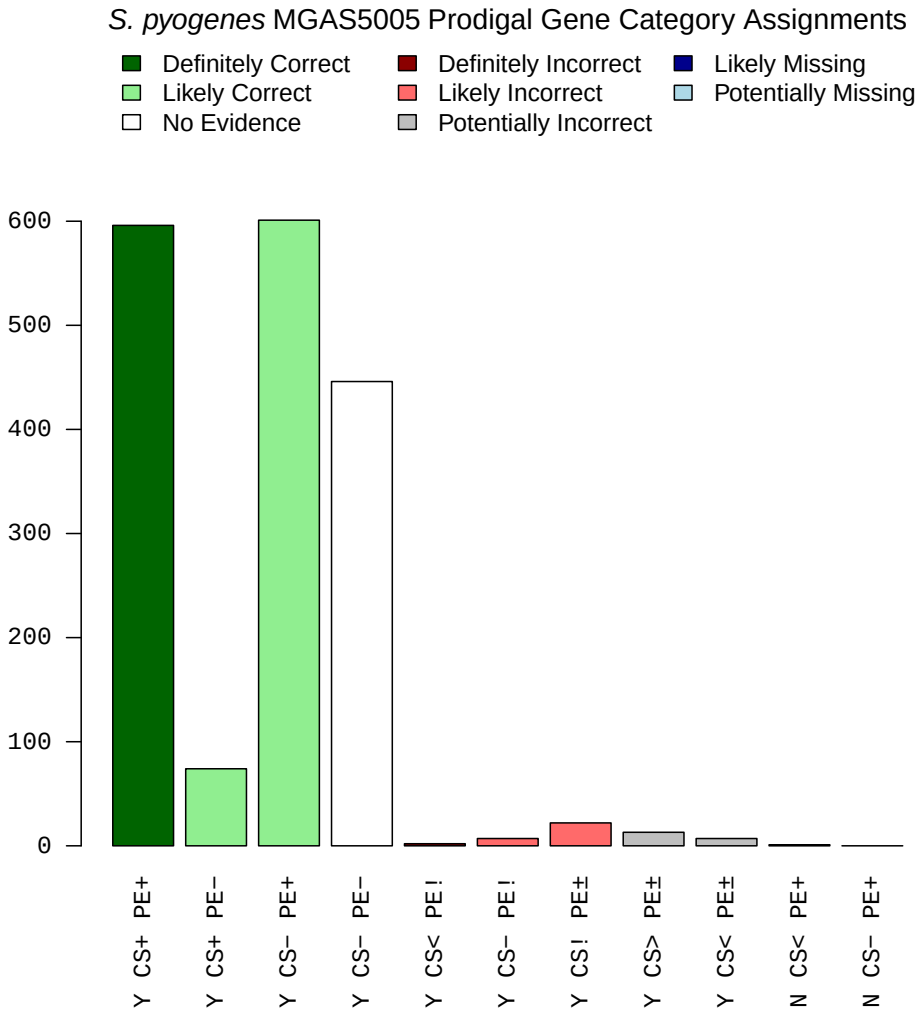
The results object contains the assigned category for each predicted gene, which describes how much evidence there is for or against that gene. There are a number of ways to look at the distribution of categories and the overall correctness of the set of predicted genes.

Printing the results object prints out the number of genes in each category as well as the accuracy scores for the set of predicted genes:

```
print(currResObj)
## An Assessment object that categorizes predicted genes
## Strain: S. pyogenes MGAS5005
## Number of Genes Provided from Prodigal: 1768
##
## Score Using All Evidence: 0.9607
## Score Using Only Proteomic Evidence: 0.9919
## Score Using Only Conserved Start Evidence: 0.9371
## Y CS+ PE+ (correct with strong evidence): 596
## Y CS+ PE- (correct with some evidence): 74
## Y CS- PE+ (correct with some evidence): 601
## Y CS- PE- (no evidence): 446
## Y CS< PE! (definitely incorrect): 2
## Y CS- PE! (likely incorrect): 7
## Y CS! PE+ (likely incorrect): 12
## Y CS! PE- (likely incorrect): 10
## Y CS> PE+ (potentially incorrect): 10
## Y CS> PE- (potentially incorrect): 3
## Y CS< PE+ (potentially incorrect): 5
## Y CS< PE- (potentially incorrect): 2
## N CS< PE+ (likely missing genes): 1
## N CS- PE+ (potentially missing genes): 0
##
## Number of Genes with Supporting Evidence: 1271
## Number of Genes with Contradictory Evidence: 51
## Number of ORFs with Protein Evidence and No Given Start: 1
```

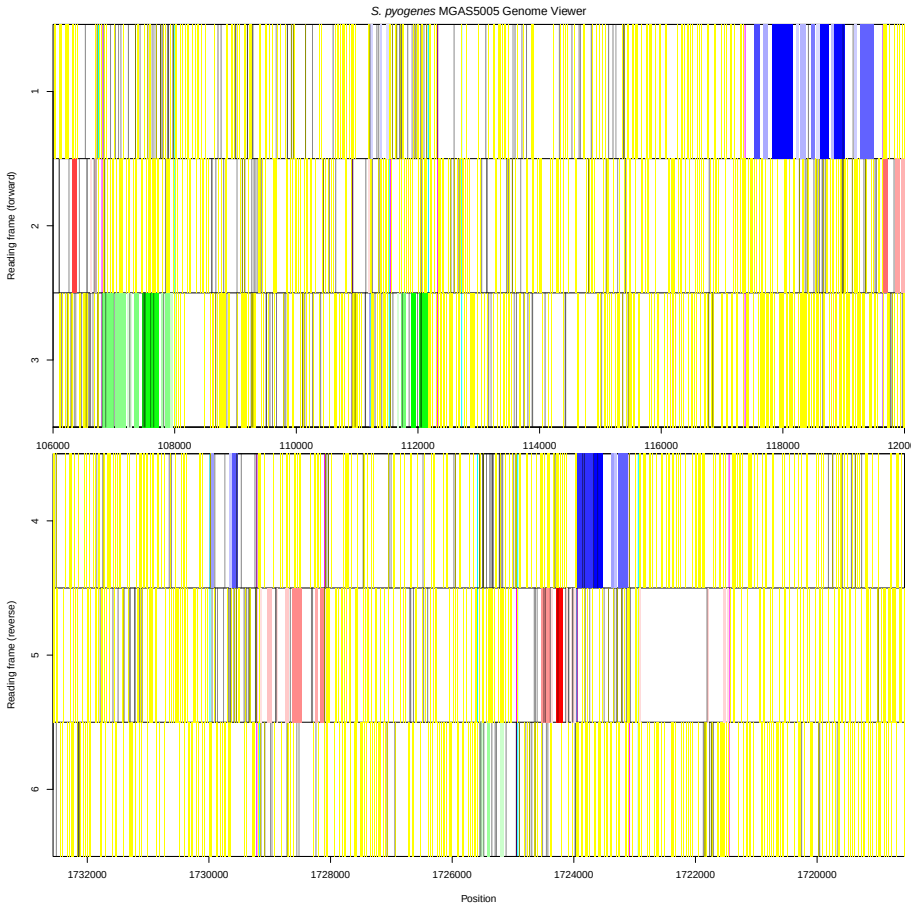
Plotting the results object shows the number of genes in each category in bar chart format:

```
plot(currResObj)
```



Plotting the results object in combination with its corresponding mapping object generates a genome viewer plot that shows how evolutionary conservation evidence & proteomics evidence align to the genome of interest, particularly in relation to the set of predicted genes:

```
plot(currMapObj, currResObj, interactive_GV = FALSE,
     rangeStart_GV = 106000, rangeEnd_GV = 120000)
```



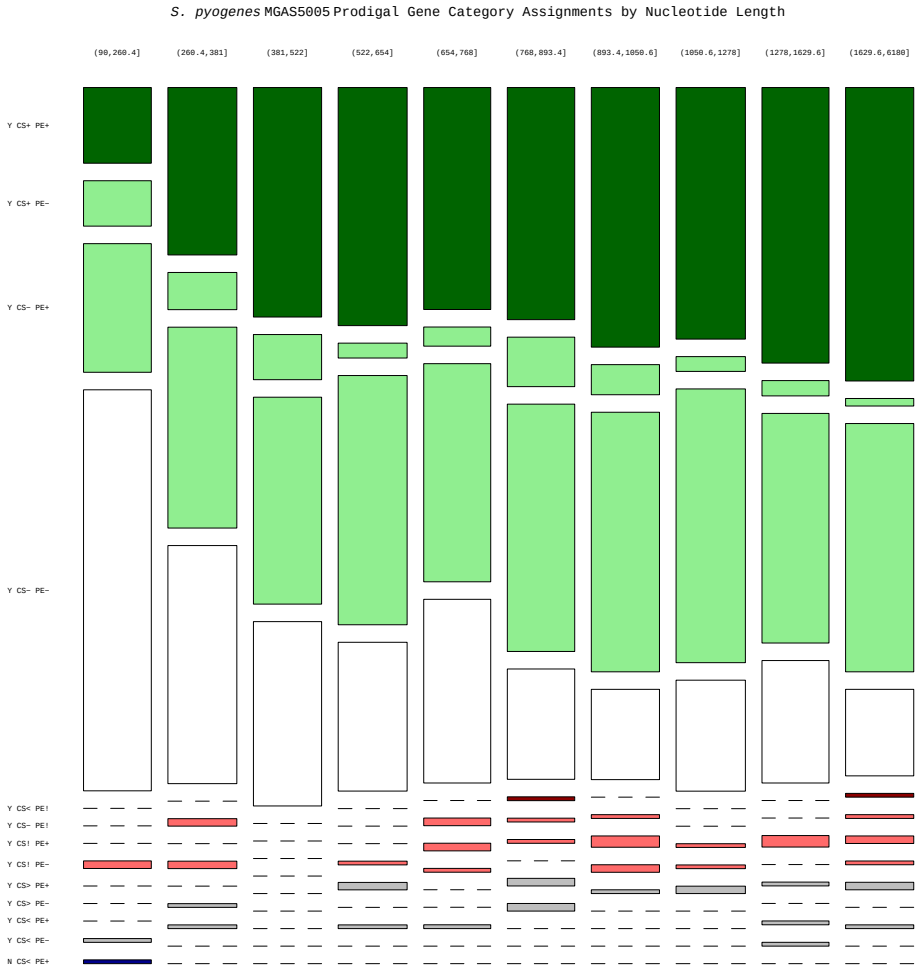
In the genome viewer plot, predicted starts are magenta lines, predicted stops are cyan lines, genome stops are yellow lines, conserved starts are gray lines, and proteomic hits are blue / red / green blocks.

In this example, `interactive_GV` was set to `FALSE` which results in an static genome viewer plot being generated. Setting `interactive_GV` to be `TRUE` results in a genome viewer that can interacted with via the `locator`, which facilitates further exploration of how the available evidence supports some genes and disproves others. With the interactive genome viewer, users may scroll to the left or right, zoom in to particular region, or zoom out.

Specifying an initial range of genomic positions to plot using `rangeStart_GV` and `rangeEnd_GV` is optional but useful for zooming into a particular region of the genome to understand why a particular gene (or group of genes) were categorized as correct / incorrect. Omitting those arguments will result genome viewer plot that spans the whole genome, which may overwhelm some graphical devices.

It is also possible to do a mosaic plot of the results object. The mosaic plot shows how the distribution of categories varies by gene length:

```
mosaicplot(currResObj)
```



8 Comparing Results Objects

Since the same mapping object can be reused to assess any set of genes for that particular genome, it is useful to compare the results from assessing the set of predictions from one program to the results from assessing the set of predictions from another program. The `CompareAssessmentResults` function compares the genes and their corresponding category assignments inside two results objects generated from the same mapping object. The example below shows a comparison of the results object generated above using predicted genes from Prodigal to a results object generated using genes from another program, GeneMarkS-2:

```
resObj2 <- readRDS(system.file("extdata",
                               "MGAS5005_PreSaved_ResultsObj_GeneMarkS2.rds",
                               package = "AssessORF"))

CompareAssessmentResults(currResObj, resObj2)
## Comparison of Two Assessment Results Objects from S. pyogenes MGAS5005
## Object 1 - Prodigal vs Object 2 - GeneMarkS2
##
## Number of Shared Coding Regions (number of stops found in both sets): 1729
##
## Number of Shared Genes (number of start-stop pairs found in both sets): 1632
##
## Number of Shared Stop - Different Start Coding Regions: 97
## (This is where the stop for a coding region is found in both sets,
## but the corresponding starts in each set differ.)
##
## For the shared stop - different start set, there were 20 instances
## where the start from object 1 has conservation evidence ('CS+')
## and the start from object 2 does not ('CS-', 'CS>', 'CS<').
## For the shared stop - different start set, there were 12 instances
## where the start from object 2 has conservation evidence ('CS+')
## and the start from object 1 does not ('CS-', 'CS>', 'CS<').
```

9 Session Info

All of the output in this vignette was produced under the following conditions:

```
## R version 4.1.0 beta (2021-05-03 r80259)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## Matrix products: default
## BLAS:   /home/biocbuild/bbs-3.14-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.14-bioc/R/lib/libRlapack.so
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] AssessORF_1.11.0    DECIPHER_2.21.0     RSQLite_2.2.7
##  [4] Biostrings_2.61.0   GenomeInfoDb_1.29.0 XVector_0.33.0
##  [7] IRanges_2.27.0      S4Vectors_0.31.0    BiocGenerics_0.39.0
## [10] BiocStyle_2.21.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.6             compiler_4.1.0         BiocManager_1.30.15
##  [4] bitops_1.0-7           tools_4.1.0            zlibbioc_1.39.0
##  [7] digest_0.6.27          bit_4.0.4              evaluate_0.14
## [10] memoise_2.0.0          pkgconfig_2.0.3        rlang_0.4.11
## [13] DBI_1.1.1              yaml_2.2.1             xfun_0.23
## [16] fastmap_1.1.0          GenomeInfoDbData_1.2.6 stringr_1.4.0
## [19] knitr_1.33             vctrs_0.3.8            bit64_4.0.5
## [22] rmarkdown_2.8          bookdown_0.22          blob_1.2.1
## [25] magrittr_2.0.1         htmltools_0.5.1.1      GenomicRanges_1.45.0
## [28] stringi_1.6.2          RCurl_1.98-1.3         cachem_1.0.5
## [31] crayon_1.4.1
```