Jose Carrillo

Professor Ash Stuart

Advanced Bioinformatics

10 May 2019

Leader Genes Approach: Reference Manual

“app.R” sources both “mining.py” and “processing.R”

Global variables and their initialization:

*from “app.R”* – The following four variables are used extensively during the lifetime of the application, and must be initialized outside the *ui* and *server* functions, although their actual values are set within those functions

* term <- “”; Initialized with either the user’s input, and py$term is set to its value, or it is not used at all in case the user chooses to import data
* genes\_interactions <- “”; Data frame containing all known gene-gene/protein-protein interactions within a given STRING network
* genes\_scores <- “”; Data frame containing a sum of the combined\_score attribute for each gene within a given STRING network
* leader\_genes <- “”; Data frame containing the first ten rows from *genes\_scores* sorted by the combined\_score attribute

*from “mining.py”*

* attempt = 0; Tracks query attempts in case of an HTTP error
* processed\_docs = 0; Tracks how many documents have been parsed
* organism = “9606[Taxonomy ID]”; The application considers only human genes, so this is included in gene database queries
* mesh\_terms = []; A list for building the final search term given the user’s input
* parsed\_uids = []; A list of NCBI identifiers that have been parsed to prevent redundancy of parsing the same file when not needed
* preliminary\_gene\_uids = {}; A dictionary mapping NCBI IDs to genes, used when cross-checking genes after preliminary mining
* cross\_checked\_genes = []; A list of genes that have been cross-checked with PubMed for functional association
* term = “”; Set from the user’s input in “app.R”, otherwise is not used or returns an error when trying to mine data with a blank term

**Script: “mining.py”**

*re\_initialize():* Used for debugging and resetting global variables

* Arguments: None
* Usage: Sets all global variables to the original initialization value
* Return Value: None

*get\_mesh\_terms(*search\_term*):* Searches for relevant MeSH terms to build the search query

* Arguments: *search­\_term* is a string value inputted by the user
* Usage: Sets global variable *term* to the search query used for data mining
* Return Value: None

*parsed(*uid*)*: Checks if a document has been parsed or not

* Arguments: *uid* is a string value corresponding to a specific NCBI identifier
* Usage: Appends *parsed\_uids* list if *uid* is not already in the list
* Return Value: None

*check\_attempts()*: Ensures that no more than 10 queries are processed every second

* Arguments: None
* Usage: Prints the number of documents each time one is processed to keep track as well as having the application sleep for one second every 10 queries
* Return Value: None

*search(*db, term, retmax=”100000”): Utilizes Entrez.esearch() function

* Arguments: *db* is the NCBI database to access, *term* is the query*, retmax* is the number of results to return. The default value is 100000 corresponding to the maximum number of results Entrez can return at a time
* Usage: For data mining, returning the identifiers of relevant articles to the search query
* Return Value: A list of NCBI identifiers resulting from the search query

*fetch(*db, uid*)*: Utilizes Entrez.efetch() function

* Arguments: *db* is the NCBI database to acesss, *uid* is the NCBI identifier of the document to fetch
* Usage: For data mining, returns the actual XML document to parse, is very prone to producing an HTTP error because it has the slowest runtime
* Return Value: XML document of the corresponding identifier or the value 0 if an HTTP 400-599 error occurs more than three times

*fetch\_summary(*db, uid*)*: Utilizes Entrez.summary() function

* Arguments: *db* is the NCBI database to acesss, *uid* is the NCBI identifier of the document summary to fetch
* Usage: For data mining, returns an XML document summary of the corresponding identifier, is used when the data needed is in the summary as opposed to requiring the entire document (in that case, *fetch(*db, uid*)* is used instead)
* Return Value: XML document summary of the corresponding identifier

*insert\_gene(*uid, xml\_gene*)*: Inserts a gene into the *preliminary\_gene\_uids* dictionary

* Arguments: *uid* is the NCBI identifier of the gene document, *xml\_gene* is the gene document
* Usage: Uses the XML document to find the gene symbol and inserts it into *preliminary\_gene\_uids* with the identifier as the key and the symbol as the value
* Return Value: None

*verify\_doc(*db, uid, xml\_doc*)*: Comprehensive function to verify functional association between a document and the phenotype of interest from Gene, MedGen, or GTR databases

* Arguments: *db* is the NCBI databse accessed, but can only be Gene, MedGen, or GTR, *uid* is the corresponding NCBI identifier of the XML document, *xml\_doc*
* Usage: Verifies if the fetched document contains the search query as evidence of a functional association
* Return Value: None

*seed\_preliminary\_gene\_uids()*: Populates dataset of preliminary genes by mining Gene, MedGen, and GTR

* Arguments: None, although global variable *term* must not be empty or null
* Usage: Initializes the application by mining for preliminary genes that may have a functional association with the phenotype of interest, populate *preliminary\_gene\_uids* for cross-checking later
* Return Value: None

*cross\_check\_gene(*gene*)*: Cross-checks gene against NCBI PubMed to verify functional association in literature

* Arguments: *gene* is the gene symbol to cross-check against PubMed
* Usage: *gene* and *term* are checked for co-occurrence in PubMed literature and
* Return Value: True or False, corresponding to if there was a co-occurrence found or not, respectively

*export\_mesh\_terms()*: Exports *mesh\_terms* to the “data” subdirectory

* Arguments: None
* Usage: Exports “mesh\_terms.txt”, a text file containing all terms used in the search query
* Return Value: None

*export\_preliminary\_gene\_uids():* Exports *preliminary\_gene\_uids* to the “data” subdirectory

* Arguments: None
* Usage: Exports “preliminary\_gene\_uids.json” containing the key (NCBI identifier) and value (gene symbol) pairs of the preliminary genes mined from Gene, GTR, and MedGen
* Return Value: None

*export\_cross\_checked\_genes():* Exports *cross\_checked\_genes* to the “data” subdirectory

* Arguments: None
* Usage: Exports “cross\_checked\_genes.csv” containing all gene symbols of preliminary genes that have been cross-checked with PubMed
* Return Value: None

*export\_parsed\_uids():* Exports *parsed\_uids* to the “data” subdirectory

* Arguments: None
* Usage: Exports “export\_parsed\_uids.txt” containing all NCBI identifiers across all databases that have been parsed by the application
* Return Value: None

**Script: “processing.R”**

*initialize(*term*)*: Calls Python functions with R reticulate to initialize preliminary data mining

* Arguments: *term* is the search term designated by the user for use in the query
* Usage: Calls *get\_mesh\_terms()*, *seed\_preliminary\_gene\_uids()*, and *cross\_check\_gene(*gene*)*, in succession while exporting MeSH terms as well as the cross-checked gene lists with *export\_mesh\_terms()* and *export\_cross\_checked\_genes()*. This function alone completely creates the initial cross-checked gene list
* Return Value: None

*process\_data(*genes\_interactions, hits*)*: Uses a data frame containing all gene interactions to return a sorted data frame with calculated global scores, the first ten rows are the leader genes

* Arguments: *genes\_interactions* is a data frame created with the STRINGdb method string\_db$*get\_interactions()* that contains all interactions of the network calculated by STRING. *hits* is a vector of STRING identifiers that were mapped from gene symbols
* Usage: The combined\_score attribute of *genes\_interactions* are summed to create a new data frame *genes\_scores* that contain all of the genes in the network with their global score. Only the top 400 genes are considered, because STRINGdb cannot accept a larger query.
* Return Value: Data frame *genes\_scores­* containing each gene symbol, its corresponding STRING identifier, and global score sorted in descending order

*calculate\_scores(*genes\_scores, genes\_interactions*)*: Utility function of *process\_data*() to sum combined\_scores

* Arguments: *genes\_interactions* data frame to process all interaction scores and *genes\_scores* to modify the global score attribute once it has finished calculating
* Usage: Sums all combined\_score attribute of all interactions and adds this to the global score attribute of *genes\_scores*
* Return Value: *genes\_scores* data frame with calculated global scores

*get\_clusters(*hits*)*:

* Arguments: *hits* is a vector of STRING identifiers
* Usage: Calculates the clusters of the STRING network using “fastgreedy” algorithm from R iGraph
* Return Value: Returns *clusters*, which is a list of STRING identifiers of clusters that can be plotted as a new STRING network and also processed with *process\_data()*

*map\_names(*STRING\_id*)*: Maps STRING identifiers to their corresponding gene symbol

* Arguments: *STRING\_id* is a vector of STRING identifiers
* Usage: Maps gene symbols using an *aliases* table from string\_db$*get\_aliases()*
* Return Value: *gene* is a vector of gene symbols

*expand\_network(*hits*)*: Finds neighboring genes of a STRING network and cross-checks them against PubMed

* Arguments: *hits* is a vector of STRING identifiers
* Usage: Utilizes string\_db$*get\_neighbors()* to find neighboring genes of the network, and cross-checks each of these genes with PubMed. If any new genes are found, it appends those STRING identifiers to *hits* and recursively calls itself until no new genes are found
* Return Value: Returns *hits* with new STRING identifiers, genes that have been found to have a functional association with the disease, if any are found

**Script: “app.R”**

The only functions in “app.R” are the *ui* and *server* functions that R Shiny uses.