Mapper Package

Last Edited By: Devon Leadman

Last Edited On: 16 May 2022

**CLASS Map**

VARIABLES

**nlp**

SpaCy NLP class used for matching and tokenization

**matcher**

SpaCy PhraseMatcher object used for matching the input and the output data

**attr**

Lets nlp know how it will match the phrases, defaulted to lowercase

**matches**

Stores information on the matches found during phrasematching

**false\_positives**

A FalsePositives object

**root**

Store the current working directory

**system**

Identifies the system that the program is running on

**path\_char**

Single character for use in making file paths based on the current OS

**isWindows**

Bool value to identify if the current OS is Windows

**gardObj**

Stores the GARD database data for matching

**dataObj**

Stores the other input in which the user wants to match the GARD data to

**batchsize**

Stores how many batches the current system can manage

METHODS

**calc\_batch(none):int**

**PARAMETERS**

**none**

Used to calculate the optimal number of batches and threads based on the current systems resources

**batch\_thread(obj,function,int)**

**PARAMETERS**

**obj = data object in which you want to batch up**

**funct = function that you want to be ran in each thread**

**size = size of batch**

Takes a data object and splits them into batches and then each batch will be put on their own thread to run the chosen function

**setup\_nlp(none):none**

**PARAMETERS**

**none**

Sets up the tokenizer, phrasematcher, and loads the English language package for use in the code

**make\_patterns(none):none**

**PARAMETERS**

**none**

Part of the setup of the NLP, creates a list of all the phrases (GARD names and synonyms) that will be used to match with the user input data

**\_create\_path(str,bool): str**

**PARAMETERS**

**filename = name of the file**

**input\_file = is True when you are creating a path to the input folder; else it creates a path to the output folder**

Returns a file path to either the input or output folders for I/O work

**\_clean\_gard(str): dict**

**PARAMETERS**

data = GARD database JSON data

Converts the JSON data into a python dictionary

**\_custom\_tokenizer(Language()): Tokenizer()**

**PARAMETERS**

**nlp = Language object used for NLP**

Returns the customized tokenizer for use in phrasematching

**\_loadGard(str): none**

**PARAMETERS**

**datafile\_name = file path to be loaded into the Map class**

Stores the file path to the GARD database data

**\_loadData(str): none**

**PARAMETERS**

**datafile\_name = file path to be loaded into the Map class**

Stores the file path to the user input file

**\_normalize(str): str**

**PARAMETERS**

**text = string to be edited**

Gets a string and makes various edits to the string to fit a specific format then returns it

**CLASS RedditMap**

**\_clean(none): none**

**PARAMETERS**

**none**

Cleans both GARD and input data for use in the object

**\_convert\_data(list): none**

**PARAMETERS**

**chunk = list of data entries**

Converts the list of data entries into a list of tuples

**\_clean\_input(list): none**

**PARAMETERS**

**chunk = list of data entries**

Converts the list of data entries into a list of tuples using batching and threading

**\_match(str, str): none**

**PARAMETERS**

**inputFile = file name to user input file**

**gardFile = file name to GARD data file**

Starts the phrasematching process between the two selected files

**\_process\_doc(list): none**

**PARAMETERS**

**batch = list of user input data tuples**

Gets metadata for each subreddit and stores them into their own SpaCy Doc object

**append\_match\_dict(Doc()): none**

**PARAMETERS**

**doc = SpaCy doc object for the specific subreddit**

Gathers information on each match during phrasematching

**\_find\_matches(none): none**

**PARAMETERS**

**none**

Converts all the phrase match results to a python dictionary

**\_get\_matches(none): dict**

**PARAMETERS**

**none**

Returns the subreddit JSON dictionary

**\_find\_match(list): dict**

**PARAMETERS**

**hit\_list = 2D list of matches and respective data**

Pairs match results to their respective GARD disease

**\_get\_subreddit\_data(str): str**

**PARAMETERS**

**subreddit = name of subreddit**

Gets metadata for a specific subreddit

**\_get\_true\_positives(none): dict**

**PARAMETERS**

**none**

Returns matches that are not false positives

**\_display\_results(none): none**

**PARAMETERS**

**none**

Displays match results relative to GARD data and saves subreddit matches to file

**CLASS AbstractMap**

**\_clean(none): none**

**PARAMETERS**

**none**

Normalizes Abstract text and stores it into a list of tuples

**\_match(str, str): none**

**PARAMETERS**

**inputFile = file name to user input file**

**gardFile = file name to GARD data file**

Starts the phrasematching process between the two selected files

**\_process\_doc(list): none**

**PARAMETERS**

**batch = list of user input data tuples**

Gets metadata for each Abstract

**append\_match\_dict(Doc()): none**

**PARAMETERS**

**doc = SpaCy doc object for the specific Abstract**

Gathers information on each match during phrasematching

**\_clean\_csv(str): none**

**PARAMETERS**

**df = Pandas Dataframe object of the match result table**

Adds calculated columns to dataframe and cleans up output before saving it as a CSV

**CLASS FalsePositives**

VARIABLES

**false\_positives**

List of strings that contain all of the object’s false positive phrases

**false\_positives\_acronyms**

List of strings that contain all of the object’s false positive acronyms

METHODS

**\_clear(bool): none**

**PARAMETERS**

**acronyms = is True when false\_positive\_acronyms is to be cleared**

Clears the entire false\_positive or false\_positive\_acronyms list, the bool parameter selects which list you are editing

**\_getall(bool): none**

**PARAMETERS**

**acronyms = is True when false\_positive\_acronyms is to be returned**

Returns the entire false\_positive or false\_positive\_acronyms list, the bool parameter selects which list you are returning

**\_remove(str,bool): none**

**PARAMETERS**

**word = string value to be removed**

**acronyms = is True when false\_positive\_acronyms is to be removed from**

Removes a string from the false\_positive or false\_positive\_acronyms list, the bool parameter selects which list you are removing from

**\_pop(int,bool): none**

**PARAMETERS**

**index = list index where element is to be removed**

**acronyms = is True when false\_positive\_acronyms is to be removed from**

Removes a string from the false\_positive or false\_positive\_acronyms list based on index; the bool parameter selects which list you are removing from

**\_add(str,bool): none**

**PARAMETERS**

**word = string value to be added to the list**

**acronyms = is True when false\_positive\_acronyms is to be added to**

Adds a string to the false\_positive or false\_positive\_acronyms list, the bool parameter selects which list you are adding to