Module Interface Specification for Natural Language Processing for Mental Health Risk Prediction

Team 13, The Cognitive Care Crew
Jessica Dawson
Michael Breau
Matthew Curtis
Benjamin Chinnery
Yaruo Tian

April 4, 2024

1 Revision History

Date	**Version**	**Notes**
January 17, 2024	1.0	Revision 0

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/MichaelBreau/nlp-mentalhealth/blob/main/docs/SRS/index.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for Natural Language Processing for Mental Health Risk Prediction

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/MichaelBreau/nlp-mentalhealth.

4 Notation

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Natural Language Processing for Mental Health Risk Prediction.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
list	[item, item,...]	a list of objects of the same type
dict	$< key : T, val : T >$	a dictionary of key value pairs
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Natural Language Processing for Mental Health Risk Prediction uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Natural Language Processing for Mental Health Risk Prediction uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**	**Level 3**
Hardware-Hiding	None	
3*Behaviour-Hiding		
	2*Eating Disorder IO	ED Input ED Output
4*Software Decision	5*Depression Pipeline	Depression Main Depression Data Processing Depression Feature Extraction Depression Output Depression Evaluation
	4*Anorexia Pipeline	Anorexia Main Anorexia Parser Anorexia Training Anorexia Predictor
	5*Eating Disorder Pipeline	ED Pipeline Manager ED Tokenizer ED Representation Model ED Prediction Model ED Transformation 1...n (represents multiple modules)

Table 1: Module Hierarchy

6 MIS for Search for Symptoms of Depression Module

6.1 depression_main

Task 1 - Search for Symptoms of Depression for User Sentences - Main Module

6.2 Uses

Used for combining the various module functionalities in a pipeline format.
Uses depression_process_data, depression_feature_extraction, depression_output, depression_accuracy_evaluation

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
N/A	N/A	N/A	-

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

None

6.4.5 Local Functions

None

6.5 depression_process_data

Task 1 - Search for Symptoms of Depression for User Sentences - Data Processing Module

6.6 Uses

Used for parsing user posts and golden truth files

6.7 Syntax

6.7.1 Exported Constants

None

6.7.2 Exported Access Programs

Name	In	Out	Exceptions
pathaddress	String	String	-
parsedata	String	Dictionary	-
processdata	String	Dictionary	-
goldentruthsentences	String	Dictionary	-

6.8 Semantics

6.8.1 State Variables

None

6.8.2 Environment Variables

None

6.8.3 Assumptions

- Data from user posts are in multiple trec file and the ground truth file is in a csv file and are present in the current task directory.

6.8.4 Access Routine Semantics

pathaddress():

- transition: None
- output: The path address for the data files
- exception: None

parsedata():

- transition: None
- output: Dictionary with usernames as keys connected to a list of strings containing user posts
- exception: None

parsedata():

- transition: None
- output: Dictionary with usernames as keys connected to a list of strings containing user posts, with gibberish information removed
- exception: None

goldentruthsentences():

- transition: None
- output: Dictionary with usernames as keys connected to an integer representing their depression symptom
- exception: None

6.8.5 Local Functions

None

6.9 depression_feature_extraction

Task 1 - Search for Symptoms of Depression for User Sentences - Feature Extraction Module

6.10 Uses

Used for predicting the top depression sentences that matches each symptom of depression

6.11 Syntax

6.11.1 Exported Constants

None

6.11.2 Exported Access Programs

Name	In	Out	Exceptions
getFeatureWords	List	Dictionary	-
feature_2_vec	Dictionary, Dictionary	Dictionary	-
getEmbeddings	Dictionary, Dictionary, Dictionary	Dictionary	-
tf_idf	Dictionary, Dictionary Integer	Dictionary	-
vectorizePosts	Dictionary, Dictionary Dictionary	Dictionary	-
cosine_sim	Dictionary, List	Dictionary	-

6.12 Semantics

6.12.1 State Variables

None

6.12.2 Environment Variables

None

6.12.3 Assumptions

None

6.12.4 Access Routine Semantics

getFeatureWords():

- transition: None
- output: A dictionary where each key represents a depression symptom, and the corresponding value is a list of strings representing feature words extracted from files.
- exception: None

feature_2_vec():

- transition: None
- output: Produces a dictionary where each key corresponds to a question number, and the associated value is a numpy array representing the vectorized features for that question.
- exception: None

getEmbeddings():

- transition: None
- output: Outputs a dictionary containing word embeddings, filtered based on words present in the data, feature sets, and a given set of word embeddings
- exception: None

tf_idf():

- transition: None
- output: Generates a dictionary where each key represents a document ID, and the associated value is a list of strings representing the top-k words based on TF-IDF scores.

- exception: None

vectorizePosts():

- transition: None
- output: A dictionary where each key represents a sentence, and the corresponding value is a numpy array representing the vectorized representation of the sentence.
- exception: None

cosine_sim():

- transition: None
- output: A dictionary where each key represents a sentence, and the associated value is a float representing the cosine similarity between the sentence vector and a target vector.
- exception: None

6.12.5 Local Functions

None

6.13 depression_output

Task 1 - Search for Symptoms of Depression for User Sentences - Ouput Module

6.14 Uses

Used for generating output into a specific format

6.15 Syntax

6.15.1 Exported Constants

None

6.15.2 Exported Access Programs

Name	In	Out	Exceptions
write_to_text_file	Any, String	-	-
write_dict_to_file	Dictionary, String	-	-
get_questions	String	List	-
get_top_matchs	String, Dictionary	Dictionary	-

6.16 Semantics

6.16.1 State Variables

None

6.16.2 Environment Variables

None

6.16.3 Assumptions

None

6.16.4 Access Routine Semantics

write_to_text_file():

- transition: None
- output: None
- exception: None

write_dict_to_file():

- transition: None
- output: None
- exception: None

get_questions():

- transition: None
- output: A list of strings representing symptom names extracted from a JSON questionnaire file.
- exception: None

get_top_matchs():

- transition: None
- output: A dictionary where each key represents a question index, and the associated value is a list of top matches based on user responses.
- exception: None

6.16.5 Local Functions

None

6.17 depression_accuracy_evaluation

Task 1 - Search for Symptoms of Depression for User Sentences - Accuracy Evaluation Module

6.18 Uses

Used for determining the accuracy of generated results using various metrics

6.19 Syntax

6.19.1 Exported Constants

None

6.19.2 Exported Access Programs

Name	In	Out	Exceptions
golden_truth_sentence_data	String	Dictionary	-
get_average_precisions	Dictionary, Dictionary	List	-
print_accuracy	List, List, String	-	-

6.20 Semantics

6.20.1 State Variables

None

6.20.2 Environment Variables

None

6.20.3 Assumptions

None

6.20.4 Access Routine Semantics

golden_truth_sentence_data():

- transition: None
- output: A dictionary where each key represents a question index, and the associated value is a list of sentence IDs and their relevances based on a provided CSV file.
- exception: None

get_average_precisions():

- transition: None
- output: A list containing precision values and NDCG values calculated based on the provided sentence scores and golden truth values.
- exception: None

print_accuracy():

- transition: None
- output: None
- exception: None

6.20.5 Local Functions

None

7 MIS for Early Detection of Signs of Anorexia Module

7.1 anorexia_main

Task 2 - Early Detection of Signs of Anorexia using User Posts - Main Module

7.2 Uses

Used for combining the various module functionalities in a pipeline format

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
N/A	N/A	N/A	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

None

7.4.5 Local Functions

None

7.5 anorexia_parser

Task 2 - Early Detection of Signs of Anorexia using User Posts - Parser Module

7.6 Uses

Used for parsing user posts and golden truth files

7.7 Syntax

7.7.1 Exported Constants

None

7.7.2 Exported Access Programs

Name	In	Out	Exceptions
pathaddress	String	String	-
parsedata	String	Dictionary	-
goldentruth	String	Dictionary	-

7.8 Semantics

7.8.1 State Variables

None

7.8.2 Environment Variables

None

7.8.3 Assumptions

- Data from user posts are in an xml file and the ground truth file is in a txt file and are present in the current working directory.

7.8.4 Access Routine Semantics

pathaddress():

- transition: None
- output: The path address for the data files
- exception: None

parsedata():

- transition: None
- output: Dictionary with usernames as keys connected to a list of strings containing user posts
- exception: None

goldentruth():

- transition: None
- output: Dictionary with usernames as keys connected to an integer representing their anorexia status (0 or 1)
- exception: None

7.8.5 Local Functions

None

7.9 anorexia_training

Task 2 - Early Detection of Signs of Anorexia using User Posts - Training Module

7.10 Uses

Used for training bertopic model on parsed data using golden truth values

7.11 Syntax

7.11.1 Exported Constants

None

7.11.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
bert	Dictionary	List, List	-
predictor	List, List	Dictionary	-
calculateaccuracy	Dictionary, Dictionary	None	-

7.12 Semantics

7.12.1 State Variables

None

7.12.2 Environment Variables

None

7.12.3 Assumptions

- The model is trained on accurate data with correctly linked user-truth values.

7.12.4 Access Routine Semantics

bert():

- transition: None
- output: List 1: List of usernames, List 2: List of topics generated from the bertopic model with the indexes corresponding to the usernames from the first list
- exception: None

predictor():

- transition: None
- output: Dictionary with usernames as keys and the predicted 0 or 1 integer linked
- exception: None

calculateaccuracy():

- transition: None
- output: Accuracy scores comparing golden truth to predictor values

$$Precision(P) = \frac{|u \in U : d = g = 1|}{|u \in U : d = 1|}$$

$$Recall(R) = \frac{|u \in U : d = g = 1|}{|u \in U : g = 1|}$$

$$F - Score(F) = \frac{2 * P * R}{P + R}$$

- exception: None

7.12.5 Local Functions

None

7.13 anorexia_predictor

Task 2 - Early Detection of Signs of Anorexia using User Posts - Predictor Module

7.14 Uses

Used for predicting new documents unseen by training

7.15 Syntax

7.15.1 Exported Constants

None

7.15.2 Exported Access Programs

Name	In	Out	Exceptions
predict_new	String	Integer	-

7.16 Semantics

7.16.1 State Variables

None

7.16.2 Environment Variables

None

7.16.3 Assumptions

None

7.16.4 Access Routine Semantics

predict_new():

- transition: None
- output: Integer 0 or 1 corresponding to the model prediction of anorexia negative or positive
- exception: None

7.16.5 Local Functions

None

8 MIS for Measuring the Severity of the Signs of Eating Disorders

8.1 Module ED Parser

8.2 Uses

None

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
parse_json	*String*	*Dict*	*IncorrectFileFormat*

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

None

8.4.3 Assumptions

The input to parse_json will be a path to a file on the disk.

8.4.4 Access Routine Semantics

parse_json($pathaddress : String$):

- transition: None
- input: a path to a file on the disk

- output: $Dict\ of < username :< labels$: [questionnaire scores], $posts$: [user's post history] $>>$
- exception: If the input file is not correctly formatted

  format of input file should have one user on each line and each user should be stored as:

  {"id": "subjectID", "label": ["0-6", "0-6", ..., "0-6" (22 entries)], "posts": [{"date": "YYYY-MM-DD HH:MM:SS", "title": "title text", "body": "body text"}, more posts]}

8.4.5 Local Functions

None

8.5 Module ED Cleaner

8.6 Uses

Parser

8.7 Syntax

8.7.1 Exported Constants

None

8.7.2 Exported Access Programs

Name	In	Out	Exceptions
clean_data	*Dict*	*Dict*	None

8.8 Semantics

8.8.1 State Variables

None

8.8.2 Environment Variables

None

8.8.3 Assumptions

The parser is working correctly

8.8.4 Access Routine Semantics

clean_data($pathaddress : String$):

- transition: None
- input: $Dict\ of < username :< labels$: [questionnaire scores]$, posts$: [user's post history] $>>$
- output: $Dict\ of < username :< labels$: [questionnaire scores]$, posts$: [user's post history] $>>$ where the user's posts no longer have a title and body but a single text field instead
- exception: None

8.8.5 Local Functions

None

8.9 Module ED Output

8.10 Uses

ED Data Storage

8.11 Syntax

8.11.1 Exported Constants

None

8.11.2 Exported Access Programs

Name	In	Out	Exceptions
output	$DataStorage$	None	$NoPredictions$

8.12 Semantics

8.12.1 State Variables

None

8.12.2 Environment Variables

None

8.12.3 Assumptions

The rest of the pipeline up to this point will function correctly

8.12.4 Access Routine Semantics

output(ds : *DataStorage*):

- transition: writes predictions in *ds* to a file where each line represents predictions for a user and is of the format "username: 0-6, 0-6, ..., 0-6" with twenty to answers of 0 to 6.
- output: None
- exception: *NoPredictions*: raised if *ds* does not have predictions stored

8.12.5 Local Functions

None

8.13 Template Module ED Data Storage

8.14 Uses

ED Cleaner

8.15 Syntax

8.15.1 Exported Constants

None

8.15.2 Exported Access Programs

Name	In	Out	Exceptions
initialize	*Dict*, *Int*	*DataStorage*	None
get_num_posts	None	*Int*	None
get_num_users	None	*Int*	None
get_embeddings	None	*List*	*NoEmbeddings*
set_embeddings	*List*	None	*WrongSize*
get_texts	None	*List*	*NoText*
set_texts	*List*	None	*WrongSize*
get_post_user_labels	None	*2dList*	*NoLabels*
get_post_labels	None	*2dList*	*NoLabels*
set_post_labels	*Int*, *List*	None	*WrongSize*
get_user_labels	None	*Dict*	*NoLabels*
get_user_predictions	None	*Dict*	*NoLabels*
set_user_predictions	*String*, *List*	None	*UserDoesNotExist*
get_user_embeddings	*String*	*List*	*NoEmbeddings*, *UserDoesNotExist*
get_user_texts	*String*	*List*	*NoText*, *UserDoesNotExist*
get_user_post_labels	*String*	*2dList*	*NoLabels*, *UserDoesNotExist*
get_user_post_label_counts	*String*, *Int*	*Tuple*	*NoLabels*, *UserDoesNotExist*
get_state	None	*Dict*	None
set_state	*Int*, *Bool*	None	*NotASplit*

8.16 Semantics

8.16.1 State Variables

Split : *Int* stores the current train/test split the DataStorage is focused on, must be between -1 and the number of splits specified at initialization minus 1. -1 means the DataStorage is focused on the entirety of the data.
Train : *Bool* stores whether the DataStorage is currently focused on the train split or the test split
TheData : Potentially multiple different data structure and lists storing the data, includes space to store a post history for each user, expected scores for each user, predicted scores for each user, embeddings for each post, and scores for each post. All of these fields can be changed and viewed through getters and setters

8.16.2 Environment Variables

None

8.16.3 Assumptions

ED Cleaning is working correctly

8.16.4 Access Routine Semantics

initialize(data : *Dict*, num_splits : *Int*)

- input: data is a dictionary of the form produced by ED Cleaning
- transition: constructs a DataStorage() object, stores the post history and expected scores of each user in TheData, produces num_splits train/test splits on the data, and sets the current split to -1
- output: the DataStorage() object constructed
- exception: None

get_num_posts()

- input: None
- transition: None
- output: the number of posts stored in the DataStorage
- exception: None

get_num_users()

- input: None
- transition: None
- output: the number of users stored in the DataStorage
- exception: None

get_embeddings()

- input: None
- transition: None
- output: a list of embeddings for the currently focused split
- exception: *NoEmbeddings* thrown if the DataStorage does not currently have embeddings stored

set_embeddings(embeddings : *List*)

- input: a list of embeddings
- transition: sets the embeddings for the current split to the input
- output: None
- exception: *WrongSize* thrown if embeddings are not the same length as the currently focused split

get_texts()

- input: None
- transition: None
- output: a list of texts for the currently focused split
- exception: *NoTexts* thrown if the DataStorage does not currently have texts stored

set_texts(texts : *List*)

- input: a list of texts
- transition: sets the texts for the current split to the input
- output: None
- exception: *WrongSize* thrown if texts are not the same length as the currently focused split

get_post_user_labels()

- input: None

- transition: None
- output: a list of user labels for each post, the label for each post is the score of the author of said post. Gets these labels for the currently focused split
- exception: $NoLabels$ thrown if the DataStorage does not currently have user labels stored

get_post_labels()

- input: None
- transition: None
- output: a list of labels for each post for the currently focused split
- exception: $NoLabels$ thrown if the DataStorage does not currently have post labels stored

set_post_labels(question : *int*, labels : *List*)

- input: question is the questionnaire question the labels are for, labels is the labels to set
- transition: sets the post labels for question to labels for the current split
- output: None
- exception: $WrongSize$ thrown if labels are not the same length as the currently focused split

get_user_labels()

- input: None
- transition: None
- output: a dictionary of the form $<$ "username" : [actual scores for each question] $>$ that maps a user's username to the scores they filled on the EDE-Q
- exception: $NoLabels$ thrown if the DataStorage does not currently have user labels stored

get_user_predictions()

- input: None
- transition: None
- output: a dictionary of the form $<$ "username" : [predicted scores for each question] $>$ that maps a user's username to the scores the model predicts they have on the EDE-Q
- exception: $NoLabels$ thrown if the DataStorage does not currently have predicted user labels stored

set_user_predictions(user : $String$, preds : $List$)

- input: user is the user to set predictions for, preds are the predictions to set
- transition: sets the predicted user labels for user to preds
- output: None
- exception: $UserDoesNotExist$ thrown if user does not exist in the DataStorage

get_user_embeddings(user : $String$)

- input: the user to get embeddings for
- transition: None
- output: list of embeddings for user
- exception: $NoEmbeddings$ thrown if the DataStorage does not currently have embeddings stored, $UserDoesNotExist$ thrown if user does not exist in the DataStorage

get_user_texts(user : $String$)

- input: the user to get texts for
- transition: None
- output: list of texts for user

- exception: *NoTexts* thrown if the DataStorage does not currently have texts stored, *UserDoesNotExist* thrown if user does not exist in the DataStorage

get_user_post_labels(user : *String*)

- input: the user to get post labels for
- transition: None
- output: list of post labels for user
- exception: *NoLabels* thrown if the DataStorage does not currently have post labels stored, *UserDoesNotExist* thrown if user does not exist in the DataStorage

get_user_post_label_counts(user : *String*, question : *String*)

- input: the user and question to get counts of post labels for
- transition: None
- output: a tuple of (number of negative scores in post history for user and question, number of positive scores, proportion of positive scores)
- exception: *NoLabels* thrown if the DataStorage does not currently have post labels stored, *UserDoesNotExist* thrown if user does not exist in the DataStorage

get_state()

- input: None
- transition: None
- output: a dictionary with entries: "split" : number of the current split, and "tr/te" : the string "train" if currently focusing train "test" otherwise
- exception: None

set_state(split : *Int*, train : *Bool*)

- input: split to set focus on and whether to focus train or test

- transition: sets state variables split and train to input variables split and train, methods that reference a focused split will now act on the specified split
- output: None
- exception: *NotASplit* thrown if input split refers to an invalid split

8.16.5 Local Functions

gen_splits(num : *Int*, train_prop : *Float*)

- input: number of train/test splits to create and what proportion of the data should be reserved for training in each split
- transition: creates splits on TheData
- output: None
- exception: None

8.17 Module ED Pipeline

8.18 Uses

ED Parser, ED Cleaner, ED Output, ED Data Storage, ED Rep Model, ED Relabeler, ED Pred Model, ED Aggregator, ED Evaluator

8.19 Syntax

8.19.1 Exported Constants

None

8.19.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
run	None	None	None

8.20 Semantics

8.20.1 State Variables

None

8.20.2 Environment Variables

None

8.20.3 Assumptions

None

8.20.4 Access Routine Semantics

run():

- input: None
- transition: runs the full pipeline
- output: None
- exception: None

8.20.5 Local Functions

None

8.21 Template Module ED Rep Model

8.22 Uses

ED Data Storage

8.23 Syntax

8.23.1 Exported Constants

None

8.23.2 Exported Access Programs

Name	In	Out	Exceptions
initialization	None	None	None
represent	*DataStorage*	None	None

8.24 Semantics

8.24.1 State Variables

Rep Model : stores the rep model in running memory

8.24.2 Environment Variables

None

8.24.3 Assumptions

DataStorage has text stored

8.24.4 Access Routine Semantics

initialize():

- input: None
- transition: loads the rep model from disk
- output: a RepModel() object
- exception: None

represent(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: stores a list of embeddings representing *ds*'s text for the current split
- output: None
- exception: None

8.24.5 Local Functions

None

8.25 Template Module ED Relabeler

8.26 Uses

ED Rep Model, ED Data Storage

8.27 Syntax

8.27.1 Exported Constants

None

8.27.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
initialization	*DataStorage*	*Relabeler*	None
relabel_all	*DataStorage*, Args	None	None
relabel_question	*DataStorage*, *Int*, Args	None	None

8.28 Semantics

8.28.1 State Variables

None

8.28.2 Environment Variables

None

8.28.3 Assumptions

DataStorage has embeddings and user labels stored

8.28.4 Access Routine Semantics

initialize(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: initializes the relabeler to use the current split of *ds* to relabel posts
- output: a *Relabeler* object
- exception: None

relabel_all(ds : *DataStorage*, HyperParameters : set of Args):

- input: a *DataStorage* object and some number of hyper parameters used to calibrate the relabeling process (depends on implementation)
- transition: calculates and stores a list of post labels in the current *ds* split
- output: None
- exception: None

relabel_question(ds : *DataStorage*, question : *Int*, HyperParameters : set of Args):

- input: a *DataStorage* object, an int specify the questionnaire question to relabel, and some number of hyper parameters used to calibrate the relabeling process (depends on implementation)
- transition: calculates and stores a list of post labels in the current *ds* split for the specified question
- output: None
- exception: None

8.28.5 Local Functions

None

8.29 Template Module ED Pred Model

8.30 Uses

ED Relabeler, ED Rep Model, ED Data Storage

8.31 Syntax

8.31.1 Exported Constants

None

8.31.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
initialize	*DataStorage*	None	None
train	None	None	None
predict	*DataStorage*	None	None

8.32 Semantics

8.32.1 State Variables

None

8.32.2 Environment Variables

None

8.32.3 Assumptions

The DataStorage should have embeddings and post labels to train on

8.32.4 Access Routine Semantics

initialize(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: initializes a *PredModel* object to use the current ds split to train on

- output: the initialized *PredModel*
- exception: None

train():

- input: None
- transition: trains the pred model on the ds split it was initialized on, saves the trained model to disk
- output: None
- exception: None

predict(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: loads the previously trained pred model from disk, predicts scores for the currently focused ds split, and stores the predictions in ds
- output: None
- exception: None

8.32.5 Local Functions

None

8.33 Template Module ED Aggregator

8.34 Uses

ED Relabeler, ED Pred Model, ED Data Storage

8.35 Syntax

8.35.1 Exported Constants

None

8.35.2 Exported Access Programs

Name	In	Out	Exceptions
initialization	*DataStorage*, Args	*Aggregator*	None
aggregate_all	*DataStorage*, Args	None	None
aggregate_question	*DataStorage*, *Int*, Args	None	None
score_users	*DataStorage*, Args	None	None

8.36 Semantics

8.36.1 State Variables

None

8.36.2 Environment Variables

None

8.36.3 Assumptions

DataStorage has embeddings and user labels stored

8.36.4 Access Routine Semantics

initialize(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: initializes the aggregator to use the current split of *ds* to aggregate posts
- output: an *Aggregator* object
- exception: None

aggregate_all(ds : *DataStorage*, HyperParameters : set of Args):

- input: a *DataStorage* object and some number of hyper parameters used to calibrate the aggregating process (depends on implementation)
- transition: initializes the aggregator to score users based on ds and HyperParameters

- output: None
- exception: None

aggregate_question(ds : *DataStorage*, question : *Int*, HyperParameters : set of Args):

- input: a *DataStorage* object, the question to initialize on, and some number of hyper parameters used to calibrate the aggregating process (depends on implementation)
- transition: initializes the aggregator to score users based on ds and HyperParameters for the specified question
- output: None
- exception: None

score_users(ds : *DataStorage*, HyperParameters : set of Args):

- input: a *DataStorage* object and some number of hyper parameters used to calibrate the aggregating process (depends on implementation)
- transition: calculates and stores user labels in the current *ds* split based off of the *ds*'s post labels
- output: None
- exception: None

8.36.5 Local Functions

None

8.37 Module ED Trainer

8.38 Uses

ED Parser, ED Cleaner, ED Output, ED Data Storage, ED Rep Model, ED Relabeler, ED Pred Model, ED Aggregator, ED Evaluator

8.39 Syntax

8.39.1 Exported Constants

None

8.39.2 Exported Access Programs

Name	In	Out	Exceptions
train	None	None	None

8.40 Semantics

8.40.1 State Variables

None

8.40.2 Environment Variables

None

8.40.3 Assumptions

DataStorage has embeddings and user labels stored

8.40.4 Access Routine Semantics

train():

- input: None
- transition: finds the optimal hyperparameters for the different parts of the pipeline and writes them to a file
- output: None
- exception: None

8.40.5 Local Functions

None

8.41 Template Module ED Evaluator

8.42 Uses

ED Aggregator, ED Metrics, ED Data Storage

8.43 Syntax

8.43.1 Exported Constants

None

8.43.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
initialization	*DataStorage*	*Evaluator*	None
evaluate	*DataStorage*	None	None
get_baselines	*DataStorage*	*Tuple*	None

8.44 Semantics

8.44.1 State Variables

None

8.44.2 Environment Variables

None

8.44.3 Assumptions

DataStorage has embeddings and user labels stored

8.44.4 Access Routine Semantics

initialize(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: initializes the evaluator to use the current split of *ds* to calculate baselines posts

- output: an *Evaluator* object
- exception: None

evaluate(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: calculates accuracy metrics and prints a comparison between a set of baselines and the predictions stored in ds to the console
- output: None
- exception: None

get_baselins(ds : *DataStorage*):

- input: a *DataStorage* object
- transition: calculates accuracy metrics for a set of baselines
- output: a tuple where each entry is a list of metric scores for a baseline
- exception: None

8.44.5 Local Functions

None

8.45 Module ED Metrics

8.46 Uses

None

8.47 Syntax

8.47.1 Exported Constants

None

8.47.2 Exported Access Programs

Name	In	Out	Exceptions
all_metrics	*List*, *List*	*List*	None
MAE	*List*, *List*	*Float*	None
MZOE	*List*, *List*	*Float*	None
RS	*List*, *List*	*Float*	None
ECS	*List*, *List*	*Float*	None
SCS	*List*, *List*	*Float*	None
WCS	*List*, *List*	*Float*	None
GED	*List*, *List*	*Float*	None

8.48 Semantics

8.48.1 State Variables

None

8.48.2 Environment Variables

None

8.48.3 Assumptions

DataStorage has embeddings and user labels stored

8.48.4 Access Routine Semantics

all_metrics(preds : *List*, expects : *List*):

- input: predicted values and expected values
- transition: None
- output: returns all relevant error metrics evaluated for preds and expects
- exception: None

MAE(preds : *List*, expects : *List*):

- input: predicted values and expected values
- transition: None
- output: returns MAE calculated for preds and expects
- exception: None

MZOE(preds : $List$, expects : $List$):

- input: predicted values and expected values
- transition: None
- output: returns MAE calculated for preds and expects
- exception: None

RS(preds : $List$, expects : $List$):

- input: predicted values and expected values
- transition: None
- output: returns MAE calculated for preds and expects
- exception: None

ECS(preds : $List$, expects : $List$):

- input: predicted values and expected values
- transition: None
- output: returns MAE calculated for preds and expects
- exception: None

SCS(preds : $List$, expects : $List$):

- input: predicted values and expected values
- transition: None
- output: returns MAE calculated for preds and expects

- exception: None

WCS(preds : *List*, expects : *List*):

- input: predicted values and expected values
- transition: None
- output: returns MAE calculated for preds and expects
- exception: None

GED(preds : *List*, expects : *List*):

- input: predicted values and expected values
- transition: None
- output: returns MAE calculated for preds and expects
- exception: None

8.48.5 Local Functions

None

9 Appendix

9.1 Reflection

9.1.1 Limitations of the proposed solution.

One of the primary limitations recognized in our design is that the performance of the various risk detectors are all turtlenecked with the spread of training data available to them. This primarily being enforced by biases present in the data which may skew results incorrectly, a prominent issue with all projects that rely on artificial intelligence to help predict an outcome. This bias can present itself as a decreased effectiveness for variations in cultural and linguistic backgrounds of the analyzed user data.

Additionally, this ties into the overall limitation of the system of how this tool is intended to be assistive and supplementary to trained mental health professionals, it is not meant to be the sole source of a diagnosis. Biases and limitations of the code structure will always offer an amount of machine error, which is why this program is to be used under the guidance of professionals who can help corroborate the results using their situation context and emotional intelligence, an ability that cannot ever truly be given to a computer.

9.1.2 Benefits and trade-offs of other potential solutions.

Early in the design and planning process, the team experimented with the idea of using large language models similar to ChatGPT for diagnosis. This would allow us to leverage well documented and existing technologies in our efforts, providing a strong jumping off point for our workflow. Unfortunately, the team was able to soon discover research which showed that models like ChatGPT tend to draw undesirable parallels with their training data, resulting in the system unknowingly fabricating information in the input data which would provide incorrect results.

Something that was discovered while coding the project was that certain modules that could be represented as libraries benefited from being made abstract data types as the stored state of ADTs could be used to cache information in between calls to the module. This helped improve performance in various areas like when tuning a models parameters where a module needs to be run over and over.