# domain_classification_doc

Jesús Cid, Manuel A. Vázquez

Apr 04, 2022

# CONTENTS:

**1 Menu Navigator** **1**

**2 Base Task Manager** **3**

**3 Task Manager** **5**

**4 Data Manager** **9**

**5 Query Manager** **11**

**6 Classifier** **13**

**7 Custom Model** **15**

**8 Preprocessor** **17**

**9 Main Window** **23**

**10 Analyze Keywords Window** **27**

**11 Constants** **29**

**12 Get Keywords Window** **31**

**13 Get Topics List Window** **33**

**14 Messages** **35**

**15 Output Wrapper** **37**

**16 Util** **39**

**17 Worker Signals** **41**

**18 Worker** **43**

**19 Indices and tables** **45**

**Python Module Index** **47**

CHAPTER

ONE

# MENU NAVIGATOR

A generic class to manage the user navigation through a multilevel options menu.

Created on March. 04, 2019

**@author: Jesús Cid Sueiro** Based on former menu manager scripts by Jerónimo Arenas.

`class src.menu_navigator.menu_navigator.`**`MenuNavigator`**(*tm*, *path2menu*, *paths2data=None*)

Bases: `object`

A class to manage the user navigation through a multilevel options menu.

The structure of multilevel menu options with their associated actions and parameters should be defined in a yaml file.

A taskmanager class is required to take the selected actions with the given parameters.

**`__init__`**(*tm*, *path2menu*, *paths2data=None*)

Initializes a menu navigator.

**Parameters**

- **tm** (*object*) – A task manager object, that will be in charge of executing all actions selected by the user through the menu interaction. Thus, it must contain: (1) One action method per method specified in the menu structure (2) Data collection methods, required for some menus with dynamic options.
- **path2menu** (*str*) – The route to the yaml file containing the menu structure
- **paths2data** (*dict or None, optional (default=None)*) – A dictionary of paths to data repositories. The key is a name of the path, and the value is the path.

**`__weakref__`**

list of weak references to the object (if defined)

**`clear`**()

Cleans terminal window

**`front_page`**(*title*)

Prints a simple title heading the application user screen

**Parameters** **title** (*str*) – Title message to be printed

**`navigate`**(*option=None*, *active_options=None*)

Manages the menu navigation loop

**Parameters**

- **options** (*dict*) – A dictionary of options

- **active_options** (*list or None, optional (default=None)*) – List of option keys indicating the available options to print. If None, all options are shown.

`query_options`(*options*, *active_options=None*, *msg=None*, *zero_option='exit'*)

Prints a heading mnd the subset of options indicated in the list of active_options, and returns the one selected by the used

**Parameters**

- **options** (*dict*) – A dictionary of options
- **active_options** (*list or None, optional (default=None)*) – List of option keys indicating the available options to print. If None, all options are shown.
- **msg** (*str or None, optional (default=None)*) – Heading message to be printed before the list of available options
- **zero_option** (*str {'exit', 'up'}, optional (default='exit')*) – If 'exit', an exit option is shown If 'up', an option to go back to the main menu

**Returns** **option** – Selected option

**Return type** str

`request_confirmation`(*msg=' Are you sure?'*)

Request confirmation from user

**Parameters** **msg** (*str, optional (default=" Are you sure?")*) – Message printed to request confirmation

**Returns** **r** – User respones

**Return type** str {'yes', 'no'}

CHAPTER

TWO

# BASE TASK MANAGER

`class src.base_taskmanager.`**`baseTaskManager`**`(`*path2project*, *path2source=None*, *config_fname='parameters.yaml'*, *metadata_fname='metadata.yaml'*, *set_logs=True*`)`

Bases: `object`

Base Task Manager class.

This class provides the basic functionality to create, load and setup an execution project from the main application

The behavior of this class might depend on the state of the project, which is stored in dictionary self.state, with the followin entries:

- 'isProject' : If True, project created. Metadata variables loaded
- 'configReady' : If True, config file succesfully loaded and processed

**`__init__`**`(`*path2project*, *path2source=None*, *config_fname='parameters.yaml'*, *metadata_fname='metadata.yaml'*, *set_logs=True*`)`

Sets the main attributes to manage tasks over a specific application project.

**Parameters**

- **path2project** (*str or pathlib.Path*) – Path to the application project
- **path2source** (*str or pathlib.Path or None, optional (default=None)*) – Paht to the folder containing the data sources for the application. If none, no source data is used.
- **config_fname** (*str, optional (default='parameters.yaml')*) – Name of the configuration file
- **metadata_fname** (*str or None, optional (default='metadata.yaml')*) – Name of the project metadata file.
- **set_logs** (*bool, optional (default=True)*) – If True logger objects are created according to the parameters specified in the configuration file

**`__weakref__`**

list of weak references to the object (if defined)

**`create`**`()`

Creates an application project. To do so, it defines the main folder structure, and creates (or cleans) the project folder, specified in self.path2project

**`load`**`()`

Loads an existing project, by reading the metadata file in the project folder.

It can be used to modify file or folder names, or paths, by specifying the new names/paths in the f_struct dictionary.

`setup()`
Sets up the application projetc. To do so, it loads the configuration file and activates the logger objects.

CHAPTER

THREE

# TASK MANAGER

**class** src.task_manager.**TaskManager**(*path2project*, *path2source=None*, *path2zeroshot=None*, *config_fname='parameters.yaml'*, *metadata_fname='metadata.yaml'*, *set_logs=True*)

Bases: `src.base_taskmanager.baseTaskManager`

This class extends the functionality of the baseTaskManager class for a specific example application

This class inherits from the baseTaskManager class, which provides the basic method to create, load and setup an application project.

The behavior of this class might depend on the state of the project, in dictionary self.state, with the following entries:

- 'isProject' : If True, project created. Metadata variables loaded
- **'configReady'** [If True, config file succesfully loaded. Datamanager] activated.

**__init__**(*path2project*, *path2source=None*, *path2zeroshot=None*, *config_fname='parameters.yaml'*, *metadata_fname='metadata.yaml'*, *set_logs=True*)

Opens a task manager object.

**Parameters**

- **path2project** (*pathlib.Path*) – Path to the application project
- **path2source** (*str or pathlib.Path or None (default=None)*) – Path to the folder containing the data sources
- **path2zeroshot** (*str or pathlib.Path or None (default=None)*) – Path to the folder containing the zero-shot-model
- **config_fname** (*str, optional (default='parameters.yaml')*) – Name of the configuration file
- **metadata_fname** (*str or None, optional (default=None)*) – Name of the project metadata file. If None, no metadata file is used.
- **set_logs** (*bool, optional (default=True)*) – If True logger objects are created according to the parameters specified in the configuration file

**analyze_keywords**(*wt=2*)

Get a set of positive labels using keyword-based search

**Parameters** **wt** (*float, optional (default=2)*) – Weighting factor for the title components. Keyword matches with title words are weighted by this factor

**evaluate_PUmodel**()

Evaluate a domain classifiers

`get_feedback()`
Gets some labels from a user for a selected subset of documents

`get_labels_by_keywords`(*wt=2*, *n_max=2000*, *s_min=1*, *tag='kwds'*)
Get a set of positive labels using keyword-based search

**Parameters**

- **wt** (*float, optional (default=2)*) – Weighting factor for the title components. Keyword matches with title words are weighted by this factor
- **n_max** (*int or None, optional (defaul=2000)*) – Maximum number of elements in the output list. The default is a huge number that, in practice, means there is no loimit
- **s_min** (*float, optional (default=1)*) – Minimum score. Only elements strictly above s_min are selected
- **tag** (*str, optional (default=1)*) – Name of the output label set.

`get_labels_by_topics`(*topic_weights*, *T*, *df_metadata*, *n_max=2000*, *s_min=1*, *tag='tpcs'*)
Get a set of positive labels from a weighted list of topics

**Parameters**

- **topic_weights** (*numpy.array*) – Weight of each topic
- **T** (*numpy.ndarray*) – Topic matrix
- **df_metadata** – Topic metadata
- **n_max** (*int or None, optional (defaul=2000)*) – Maximum number of elements in the output list. The default is a huge number that, in practice, means there is no loimit
- **s_min** (*float, optional (default=1)*) – Minimum score. Only elements strictly above s_min are selected
- **tag** (*str, optional (default=1)*) – Name of the output label set.

`get_labels_by_zeroshot`(*n_max=2000*, *s_min=0.1*, *tag='zeroshot'*)
Get a set of positive labels using a zero-shot classification model

**Parameters**

- **n_max** (*int or None, optional (defaul=2000)*) – Maximum number of elements in the output list. The default is a huge number that, in practice, means there is no loimit
- **s_min** (*float, optional (default=0.1)*) – Minimum score. Only elements strictly above s_min are selected
- **tag** (*str, optional (default=1)*) – Name of the output label set.

`get_labels_from_docs`(*n_docs*)
Requests feedback about the class of given documents.

**Parameters** **selected_docs** (*pands.DataFrame*) – Selected documents

**Returns** **labels** – Labels for the given documents, in the same order than the documents in the input dataframe

**Return type** list of boolean

`import_labels()`
Import labels from file

`load()`
Extends the load method from the parent class to load the project corpus and the dataset (if any)

`load_corpus`(*corpus_name*)
Loads a dataframe of documents from a given corpus.

**Parameters** **corpus_name** (*str*) – Name of the corpus. It should be the name of a folder in self.path2source

`load_labels`(*class_name*)
Load a set of labels and its corresponding dataset (if it exists)

**Parameters** **class_name** (*str*) – Name of the target category

`reevaluate_model`()
Evaluate a domain classifier

`reset_labels`(*labelset*)
Reset all labels and models associated to a given category

**Parameters** **labelset** (*str*) – Name of the category to be removed.

`retrain_model`()
Improves classifier performance using the labels provided by users

`train_PUmodel`(*max_imbalance=3*, *nmax=400*)
Train a domain classifiers

**Parameters**

- **max_imbalance** (*int or float or None, optional (default=None)*) – Maximum ratio negative vs positive samples. If the ratio in df_dataset is higher, the negative class is subsampled. If None, the original proportions are preserved
- **nmax** (*int or None (defautl=None)*) – Maximum size of the whole (train+test) dataset

`class src.task_manager.TaskManagerCMD`(*path2project*, *path2source=None*, *path2zeroshot=None*, *config_fname='parameters.yaml'*, *metadata_fname='metadata.yaml'*, *set_logs=True*)
Bases: *src.task_manager.TaskManager*

Provides extra functionality to the task manager, requesting parameters from users from a command window.

`__init__`(*path2project*, *path2source=None*, *path2zeroshot=None*, *config_fname='parameters.yaml'*, *metadata_fname='metadata.yaml'*, *set_logs=True*)
Opens a task manager object.

**Parameters**

- **path2project** (*pathlib.Path*) – Path to the application project
- **path2source** (*str or pathlib.Path or None (default=None)*) – Path to the folder containing the data sources
- **path2zeroshot** (*str or pathlib.Path or None (default=None)*) – Path to the folder containing the zero-shot-model
- **config_fname** (*str, optional (default='parameters.yaml')*) – Name of the configuration file
- **metadata_fname** (*str or None, optional (default=None)*) – Name of the project metadata file. If None, no metadata file is used.
- **set_logs** (*bool, optional (default=True)*) – If True logger objects are created according to the parameters specified in the configuration file

`analyze_keywords`()
Get a set of positive labels using keyword-based search

`get_labels_by_keywords()`
Get a set of positive labels using keyword-based search

`get_labels_by_topics()`
Get a set of positive labels from a weighted list of topics

`get_labels_by_zeroshot()`
Get a set of positive labels using keyword-based search

`get_labels_from_docs`(*selected_docs*)
Requests feedback about the class of given documents.

**Parameters** **selected_docs** (*pands.DataFrame*) – Selected documents

**Returns** **labels** – Labels for the given documents, in the same order than the documents in the input dataframe

**Return type** list of boolean

`train_PUmodel()`
Train a domain classifier

`class src.task_manager.TaskManagerGUI`(*path2project*, *path2source=None*, *path2zeroshot=None*, *config_fname='parameters.yaml'*, *metadata_fname='metadata.yaml'*, *set_logs=True*)

Bases: *src.task_manager.TaskManager*

Provides extra functionality to the task manager, to be used by the Graphical User Interface (GUI)

`get_feedback`(*idx*, *labels*)
Gets some labels from a user for a selected subset of documents

### Notes

In comparison to the corresponding parent method, STEPS 1 and 2 are carried out directly through the GUI

`get_labels_by_keywords`(*keywords*, *_tag*)
Get a set of positive labels using keyword-based search through the MainWindow

`get_suggested_keywords()`
Get the list of suggested keywords to showing it in the GUI.

**Returns** **suggested_keywords** – List of suggested keywords

**Return type** list of str

`get_topic_words`(*n_max*, *s_min*)
Get a set of positive labels from a weighted list of topics

`train_PUmodel`(*max_imabalance*, *nmax*)
Train a domain classifier

CHAPTER

FOUR

# DATA MANAGER

`class src.data_manager.DataManager`(*path2source*, *path2labels*, *path2datasets*, *path2models*, *path2embeddings=None*)

Bases: `object`

This class contains all read / write functionalities required by the domain_classification project.

It assumes that source and destination data will be stored in files.

`__init__`(*path2source*, *path2labels*, *path2datasets*, *path2models*, *path2embeddings=None*)

Initializes the data manager object

**Parameters**

- **path2source** (*str or pathlib.Path*) – Path to the folder containing all external source data
- **path2labels** (*str or pathlib.Path*) – Path to the folder containing sets of labels
- **path2datasets** (*str or pathlib.Path*) – Path to the folder containing datasets
- **path2embeddings** (*str or pathlib.Path*) – Path to the folder containing the document embeddings

`__weakref__`

list of weak references to the object (if defined)

`get_corpus_list()`

Returns the list of available corpus

`get_dataset_list()`

Returns the list of available datasets

`get_keywords_list()`

Returns a list of IA-related keywords

**Returns** **keywords** – A list of keywords

**Return type** list

`get_labelset_list()`

Returns the list of available labels

`get_model_list()`

Returns the list of available models

`import_labels`(*ids_corpus=None*, *tag='imported'*)

Loads a subcorpus of positive labels from file.

**Parameters**

- **ids_corpus** (*list*) – List of ids of the documents in the corpus. Only the labels with ids in ids_corpus are imported and saved into the output file.
- **tag** (*str, optional (default="imported")*) – Name for the category defined by the positive labels.

**Returns** **df_labels** – Dataframe of labels, with two columns: id and class. id identifies the document corresponding to the label. class identifies the class. All documents are assumed to be class 1

**Return type** pandas.DataFrame

`load_corpus`(*corpus_name*)

Loads a dataframe of documents from a given corpus.

**Parameters** **corpus_name** (*str*) – Name of the corpus. It should be the name of a folder in self.path2source

`load_dataset`(*tag=''*)

Loads a labeled dataset of documents in the format required by the classifier modules

**Parameters** **tag** (*str, optional (default="")*) – Name of the dataset

`load_labels`(*tag=''*)

Loads a set or PU labels

`load_topics`()

Loads a topic matrix for a specific corpus

`reset_labels`(*tag=''*)

Delete all files related to a given class

**Parameters** **tag** (*str, optional (default="")*) – Name of the class to be removed

`save_dataset`(*df_dataset*, *tag=''*, *save_csv=False*)

Save dataset in input dataframe in a feather file.

**Parameters**

- **df_dataset** (*pandas.DataFrame*) – Dataset to save
- **tag** (*str, optional (default="")*) – Optional string to add to the output file name.
- **save_csv** (*boolean, optional (default=False)*) – If True, the dataset is saved in csv format too

CHAPTER

FIVE

# QUERY MANAGER

class src.query_manager.**QueryManager**

Bases: `object`

This class contains all user queries needed by the datamanager.

**`__init__`**()

Initializes the query manager object

**`__weakref__`**

list of weak references to the object (if defined)

**`ask_keywords`**(*kw_library=None*)

Ask the user for a list of keywords.

**Parameters** **kw_library** (*list*) – A list of possible keywords

**Returns** **keywords** – List of keywords

**Return type** list of str

**`ask_label`**()

Ask the user for a single binary label

**Returns** **label** – Binary value read from the standard input

**Return type** int

**`ask_label_tag`**()

Ask the user for a tag to compose the label file name.

**Returns** **keywords** – List of keywords

**Return type** list of str

**`ask_labels`**()

Ask the user for a weighted list of labels related to some documents

**Returns** **labels** – List of labels

**Return type** list of int

**`ask_topics`**(*topic_words*)

Ask the user for a weighted list of topics

**Parameters** **topic_words** (*list of str*) – List of the main words from each topic

**Returns** **tw** – Dictionary of topics: weights

**Return type** dict

`ask_value`(*query='Write value: '*, *convert_to=<class 'str'>*, *default=None*)

Ask user for a value

**Parameters**

- **query** (*str, optional (default value="Write value: ")*) – Text to print
- **convert_to** (*function, optional (default=str)*) – A function to apply to the selected value. It can be used, for instance, for type conversion.
- **default** (*optional (default=None)*) – Default value to return if an empty value is returned

**Returns** The returned value is equal to conver_to(x), where x is the string introduced by the user (if any) or the default value.

**Return type** value

`confirm`()

Ask the user for confirmation

**Return type** True if the user inputs 'y', False otherwise

CHAPTER

SIX

# CLASSIFIER

**class** `src.domain_classifier.classifier.`**`CorpusClassifier`**(*df_dataset*, *path2transformers='.'*, *use_cuda=True*)

Bases: `object`

A container of corpus classification methods

**`AL_sample`**(*n_samples=5*)

Returns a given number of samples for active learning (AL)

**Parameters** **n_samples** (*int, optional (default=5)*) – Number of samples to return

**Returns** **df_out** – Selected samples

**Return type** pandas.dataFrame

**`__init__`**(*df_dataset*, *path2transformers='.'*, *use_cuda=True*)

Initializes a preprocessor object

**Parameters**

- **df_dataset** (*pandas.DataFrame*) – Dataset with text and labels. It must contain at least two columns with names "text" and "labels", with the input and the target labels for classification.
- **path2transformers** (*pathlib.Path or str, optional (default=".")*) – Path to the folder that will store all files produced by the simpletransformers library. Default value is ".".
- **use_cuda** (*boolean, optional (default=True)*) – If true, GPU will be used, if available.

## Notes

Be aware that the simpletransformers library produces several folders, with some large files. You might like to use a value of path2transformers other than '.'.

**`__weakref__`**

list of weak references to the object (if defined)

**`annotate`**(*idx*, *labels*, *col='annotations'*)

Annotate the given labels in the given positions

**Parameters**

- **idx** (*list of int*) – Rows to locate the labels.
- **labels** (*list of int*) – Labels to annotate
- **col** (*str, optional (default = 'annotations')*) – Column in the dataframe where the labels will be annotated. If it does not exist, it is created.

`eval_model`(*tag_score='score'*)
Compute predictions of the classification model over the input dataset and compute performance metrics.

**Parameters** **tag_score** (*str*) – Prefix of the score names. The scores will be save in the columns of self.df_dataset containing these scores.

### Notes

The use of simpletransformers follows the example code in https://towardsdatascience.com/simple-transformers-introducing-the- easiest-bert-roberta-xlnet-and-xlm-library-58bf8c59b2a3

`load_model`()
Loads an existing classification model

**Return type** The loaded model is stored in attribute self.model

`load_model_config`()
Load configuration for model.

If there is no previous configuration, copy it from simpletransformers ClassificationModel and save it.

`retrain_model`()
Re-train the classifier model using annotations

`train_model`(*epochs=3*, *evaluate=True*)
Train binary text classification model based on transformers

### Notes

The use of simpletransformers follows the example code in https://towardsdatascience.com/simple-transformers-introducing-the- easiest-bert-roberta-xlnet-and-xlm-library-58bf8c59b2a3

`train_test_split`(*max_imbalance=None*, *nmax=None*, *train_size=0.6*, *random_state=None*)
Split dataframe dataset into train an test datasets, undersampling the negative class

**Parameters**

- **max_imbalance** (*int or float or None, optional (default=None)*) – Maximum ratio negative vs positive samples. If the ratio in df_dataset is higher, the negative class is subsampled If None, the original proportions are preserved
- **nmax** (*int or None (defautl=None)*) – Maximum size of the whole (train+test) dataset
- **train_size** (*float or int (default=0.6)*) – Size of the training set. If float in [0.0, 1.0], proportion of the dataset to include in the train split. If int, absolute number of train samples.
- **random_state** (*int or None (default=None)*) – Controls the shuffling applied to the data before splitting. Pass an int for reproducible output across multiple function calls.

**Returns**

- *No variables are returned. The dataset dataframe in self.df_dataset is*
- *updated with a new columm 'train_test' taking values* – 0: if row is selected for training 1: if row is selected for test -1: otherwise

CHAPTER

SEVEN

# CUSTOM MODEL

A custom classifier based on the RobertaModel class from transformers.

Created on February 2022

@author: José Antonio Espinosa

`class src.domain_classifier.custom_model.`**`CustomClassificationHead`**(*classifier_dropout=0.1*, *hidden_dropout_prob=0.1*, *hidden_size=768*)

Bases: `torch.nn.modules.module.Module`

Copy of class RobertaClassificationHead (from transformers.models.roberta.modeling_roberta) Head for sentence-level classification tasks.

**`__init__`**(*classifier_dropout=0.1*, *hidden_dropout_prob=0.1*, *hidden_size=768*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

**`forward`**(*features: torch.Tensor*)

**Parameters** **features** (*Tensor*) – The encoded text

`class src.domain_classifier.custom_model.`**`CustomDataset`**(*df*)
Bases: `torch.utils.data.dataset.Dataset`

**`__init__`**(*df*)

`class src.domain_classifier.custom_model.`**`CustomEncoderLayer`**(*hidden_act='gelu'*, *hidden_size=768*, *intermediate_size=3072*, *layer_norm_eps=1e-05*, *num_attention_heads=12*, *num_hidden_layers=1*)

Bases: `torch.nn.modules.module.Module`

Custom encoder layer of transformer for classification.

**`__init__`**(*hidden_act='gelu'*, *hidden_size=768*, *intermediate_size=3072*, *layer_norm_eps=1e-05*, *num_attention_heads=12*, *num_hidden_layers=1*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

**`forward`**(*features: torch.Tensor*, *mask: torch.Tensor*)

**Parameters**

- **features** (*Tensor*) – The sequence to the encoder
- **mask** (*Tensor*) – The mask for the src keys per batch

`class src.domain_classifier.custom_model.CustomModel`(*config*, *path_model*)

Bases: `torch.nn.modules.module.Module`

Copy of class RobertaClassificationHead (from transformers.models.roberta.modeling_roberta) Head for sentence-level classification tasks.

`__init__`(*config*, *path_model*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

`create_data_loader`(*df*, *batch_size=8*)

Creates a DataLoader from a DataFrame to train/eval model

`eval_model`(*df_eval*, *device='cuda'*)

Evaluate trained model

`forward`(*features: torch.Tensor*, *mask: torch.Tensor*)

**Parameters**

- **features** (*Tensor*) – The sequence to the encoder
- **mask** (*Tensor*) – The mask for the src keys per batch

`load_embeddings`()

Load embeddings layer.

If there is no previous configuration, copy it from simpletransformers ClassificationModel and save it.

`train_model`(*df_train*, *device='cuda'*)

Train the model

**Parameters**

- **df_train** (*DataFrame*) – Training dataframe
- **epochs** (*int*) – Number of epochs to train model

CHAPTER

EIGHT

# PREPROCESSOR

**class** `src.domain_classifier.preprocessor.`**`CorpusDFProcessor`**(*df_corpus*, *path2embeddings=None*, *path2zeroshot=None*)

Bases: `object`

A container of corpus processing methods. It assumes that a corpus is given by a dataframe of documents.

Each dataframe must contain three columns: id: document identifiers title: document titles description: body of the document text

**`__init__`**(*df_corpus*, *path2embeddings=None*, *path2zeroshot=None*)

Initializes a preprocessor object

**Parameters**

- **df_corpus** (*pandas.dataFrame*) – Input corpus.
- **path2embeddings** (*str or pathlib.Path or None, optional (default=None)*) – Path to the folder containing the document embeddings. If None, no embeddings will be used. Document scores will be based in word counts
- **path2zeroshot** (*str or pathlib.Path or None, optional (default=None)*) – Path to the folder containing the pretrained zero-shot model If None, zero-shot classification will not be available.

**`__weakref__`**

list of weak references to the object (if defined)

**`compute_keyword_stats`**(*keywords*, *wt=2*)

Computes keyword statistics

**Parameters**

- **corpus** (*dataframe*) – Dataframe of corpus.
- **keywords** (*list of str*) – List of keywords

**Returns**

- **df_stats** (*dict*) – Dictionary of document frequencies per keyword df_stats[k] is the number of docs containing keyword k
- **kf_stats** (*dict*) – Dictionary of keyword frequencies df_stats[k] is the number of times keyword k appers in the corpus
- **wt** (*float, optional (default=2)*) – Weighting factor for the title components. Keyword matches with title words are weighted by this factor

**`filter_by_keywords`**(*keywords*, *wt=2*, *n_max=1e+100*, *s_min=0*)

Select documents with a significant presence of a given set of keywords

**Parameters**

- **keywords** (*list of str*) – List of keywords
- **wt** (*float, optional (default=2)*) – Weighting factor for the title components. Keyword matches with title words are weighted by this factor. Not used if self.path2embeddings is None
- **n_max** (*int or None, optional (defaul=1e100)*) – Maximum number of elements in the output list. The default is a huge number that, in practice, means there is no loimit
- **s_min** (*float, optional (default=0)*) – Minimum score. Only elements strictly above s_min are selected

**Returns** **ids** – List of ids of the selected documents

**Return type** list

`filter_by_topics`(*T*, *doc_ids*, *topic_weights*, *n_max=1e+100*, *s_min=0*)
Select documents with a significant presence of a given set of keywords

**Parameters**

- **T** (*numpy.ndarray or scipy.sparse*) – Topic matrix.
- **doc_ids** (*array-like*) – Ids of the documents in the topic matrix. doc_ids[i] = '123' means that document with id '123' has topic vector T[i]
- **topic_weights** (*dict*) – Dictionary {t_i: w_i}, where t_i is a topic index and w_i is the weight of the topic
- **n_max** (*int or None, optional (defaul=1e100)*) – Maximum number of elements in the output list. The default is a huge number that, in practice, means there is no loimit
- **s_min** (*float, optional (default=0)*) – Minimum score. Only elements strictly above s_min are selected

**Returns** **ids** – List of ids of the selected documents

**Return type** list

`get_top_scores`(*scores*, *n_max=1e+100*, *s_min=0*)
Select documents from the corpus whose score is strictly above a lower bound

**Parameters**

- **scores** (*array-like of float*) – List of scores. It must be the same size than the number of docs in the corpus
- **n_max** (*int or None, optional (defaul=1e100)*) – Maximum number of elements in the output list. The default is a huge number that, in practice, means there is no loimit
- **s_min** (*float, optional (default=0)*) – Minimum score. Only elements strictly above s_min are selected

`make_PU_dataset`(*df_labels*)
Returns the labeled dataframe in the format required by the CorpusClassifier class

**Parameters**

- **df_corpus** (*pandas.DataFrame*) – Text corpus, with at least three columns: id, title and description
- **df_labels** (*pandas.DataFrame*) – Dataframe of positive labels. It should contain column id. All labels are assumed to be positive

**Returns** **df_dataset** – A pandas dataframe with three columns: id, text and labels.

**Return type** pandas.DataFrame

`make_pos_labels_df`(*ids*)

Returns a dataframe with the given ids and a single, all-ones column

**Parameters** **ids** (*array-like*) – Values for the column 'ids'

**Returns** **df_labels** – A dataframe with two columns: 'id' and 'class'. All values in class column are equal to one.

**Return type** pandas.DataFrame

`remove_docs_from_topics`(*T*, *df_metadata*, *col_id='id'*)

Removes, from a given topic-document matrix and its corresponding metadata dataframe, all documents that do not belong to the corpus

**Parameters**

- **T** (*numpy.ndarray or scipy.sparse*) – Topic matrix (one column per topic)
- **df_metadata** (*pandas.DataFrame*) – Dataframe of metadata. It must include a column with document ids
- **col_id** (*str, optional (default='id')*) – Name of the column containing the document ids in df_metadata

**Returns**

- **T_out** (*numpy.ndarray or scipy.sparse*) – Reduced topic matrix (after document removal)
- **df_out** (*pands.DataFrame*) – Metadata dataframe, after document removal

`score_by_keyword_count`(*keywords*, *wt=2*)

Computes a score for every document in a given pandas dataframe according to the frequency of appearing some given keywords

**Parameters**

- **corpus** (*dataframe*) – Dataframe of corpus.
- **keywords** (*list of str*) – List of keywords
- **wt** (*float, optional (default=2)*) – Weighting factor for the title components. Keyword matches with title words are weighted by this factor

**Returns** **score** – List of scores, one per documents in corpus

**Return type** list of float

`score_by_keywords`(*keywords*, *wt=2*)

Computes a score for every document in a given pandas dataframe according to the frequency of appearing some given keywords

**Parameters**

- **keywords** (*list of str*) – List of keywords
- **wt** (*float, optional (default=2)*) – Weighting factor for the title components. Keyword matches with title words are weighted by this factor This input argument is used if self.path2embeddings is None only

**Returns** **score** – List of scores, one per documents in corpus

**Return type** list of float

`score_by_topics`(*T*, *doc_ids*, *topic_weights*)

Computes a score for every document in a given pandas dataframe according to the relevance of a weighted list of topics

**Parameters**

- **T** (*numpy.ndarray or scipy.sparse*) – Topic matrix (one column per topic)
- **doc_ids** (*array-like*) – Ids of the documents in the topic matrix. doc_ids[i] = '123' means that document with id '123' has topic vector T[i]
- **topic_weights** (*dict*) – Dictionary {t_i: w_i}, where t_i is a topic index and w_i is the weight of the topic

**Returns** **score** – List of scores, one per documents in corpus

**Return type** list of float

`score_by_zeroshot`(*keyword*)

Computes a score for every document in a given pandas dataframe according to the relevance of a given keyword according to a pretrained zero-shot classifier

**Parameters** **keyword** (*str*) – Keywords defining the target category

**Returns** **score** – List of scores, one per documents in corpus

**Return type** list of float

`class src.domain_classifier.preprocessor.CorpusProcessor`(*path2embeddings=None*, *path2zeroshot=None*)

Bases: `object`

A container of corpus preprocessing methods It provides basic processing methods to a corpus of text documents The input corpus must be given by a list of strings (or a pandas series of strings)

`__init__`(*path2embeddings=None*, *path2zeroshot=None*)

Initializes a preprocessor object

**Parameters**

- **path2embeddings** (*str or pathlib.Path or None, optional (default=None)*) – Path to the folder containing the document embeddings. If None, no embeddings will be used. Document scores will be based in word counts
- **path2zeroshot** (*str or pathlib.Path or None, optional (default=None)*) – Path to the folder containing the pretrained zero-shot model If None, zero-shot classification will not be available.

`__weakref__`

list of weak references to the object (if defined)

`compute_keyword_stats`(*corpus*, *keywords*)

Computes keyword statistics

**Parameters**

- **corpus** (*list (or pandas.Series) of str*) – Input corpus.
- **keywords** (*list of str*) – List of keywords

**Returns**

- **df_stats** (*dict*) – Dictionary of document frequencies per keyword df_stats[k] is the number of docs containing keyword k

- **kf_stats** (*dict*) – Dictionary of keyword frequencies df_stats[k] is the number of times keyword k appers in the corpus

`get_top_scores`(*scores*, *n_max=1e+100*, *s_min=0*)
Select the elements from a given list of numbers that fulfill some conditions

**Parameters**

- **n_max** (*int or None, optional (defaul=1e100)*) – Maximum number of elements in the output list. The default is a huge number that, in practice, means there is no loimit
- **s_min** (*float, optional (default=0)*) – Minimum score. Only elements strictly above s_min are selected

`score_docs_by_keyword_count`(*corpus*, *keywords*)
Computes a score for every document in a given pandas dataframe according to the frequency of appearing some given keywords

**Parameters**

- **corpus** (*list (or pandas.Series) of str*) – Input corpus.
- **keywords** (*list of str*) – List of keywords

**Returns** **score** – List of scores, one per document in corpus

**Return type** list of float

`score_docs_by_keywords`(*corpus*, *keywords*)
Computes a score for every document in a given pandas dataframe according to the frequency of appearing some given keywords

**Parameters**

- **corpus** (*list (or pandas.Series) of str*) – Input corpus.
- **keywords** (*list of str*) – List of keywords

**Returns** **score** – List of scores, one per document in corpus

**Return type** list of float

`score_docs_by_zeroshot`(*corpus*, *keyword*)
Computes a score for every document in a given pandas dataframe according to a given keyword and a pre-trained zero-shot classifier

**Parameters**

- **corpus** (*list (or pandas.Series) of str*) – Input corpus.
- **keyword** (*str*) – Keyword defining the target category

**Returns** **score** – List of scores, one per document in corpus

**Return type** list of float

CHAPTER

NINE

# MAIN WINDOW

@author: lcalv

**class** src.graphical_user_interface.main_window.**MainWindow**(*project_folder*, *source_folder*, *tm*, *widget*, *stdout*, *stderr*)

Bases: `PyQt5.QtWidgets.QMainWindow`

Class representing the main window of the application.

**__init__**(*project_folder*, *source_folder*, *tm*, *widget*, *stdout*, *stderr*)

Initializes the application's main window based on the parameters received from the application's starting window.

**Parameters**

- **project_folder** (*pathlib.Path*) – Path to the application project
- **source_folder** (*pathlib.Path*) – Path to the folder containing the data sources
- **tm** (*TaskManager*) – TaskManager object associated with the project
- **widget** (*QtWidgets.QStackedWidget*) – Window to which the application's main window is attached to
- **stdout** (*sys.stdout*) – Output file object
- **stderr** (*sys.stderr*) – Standard Error file object

**append_text_evaluate**(*text*)

Method to redirect the stdout and stderr in the "text_edit_results_eval_pu_classifier" while the evaluation of a PU model is being performed.

**append_text_retrain_reval**(*text*)

Method to redirect the stdout and stderr in the "text_edit_results_reval_retrain_pu_model" while the retraining of a PU model is being performed.

**append_text_train**(*text*)

Method to redirect the stdout and stderr in the "text_logs_train_pu_model" while the training of a PU model is being performed.

**center**()

Centers the window at the middle of the screen at which the application is being executed.

**clicked_change_predicted_class**(*checkbox*)

Method to control the checking or unchecking of the QCheckboxes that represented the predicted class that the user has associated to each of the documents to annotate.

**clicked_evaluate_PU_model**()

Method that controls the actions that are carried out when the button "eval_pu_classifier_push_button" is clicked by the user.

`clicked_get_labels()`
Method for performing the getting of the labels according to the method selected for it.

`clicked_get_labels_option()`
Method to control the functionality associated with the selection of each of the QRadioButtons associated with the labels' getting. Only one QRadioButton can be selected at a time.

`clicked_give_feedback()`
Method that controls the actions that are carried out when the button "give_feedback_user_push_button" is clicked by the user.

`clicked_load_corpus()`
Method to control the selection of a new corpus by double-clicking one of the items of the corpus list within the selected source folder, as well as its loading as dataframe into the TaskManager object. Important is that the corpus cannot be changed inside the same project, so if a corpus was used before me must keep the same one.

`clicked_load_labels()`
Method for controlling the loading of the labels into the session. It is equivalent to the "_get_labelset_list" method from the TaskManager class

`clicked_reevaluate_model()`
Method that controls the actions that are carried out when the button "retrain_pu_model_push_button" is clicked by the user.

`clicked_reset_labels()`
Method for controlling the resetting of the current session's labels.

`clicked_retrain_model()`
Method that controls the actions that are carried out when the button "retrain_pu_model_push_button" is clicked by the user.

`clicked_train_PU_model()`
Method that controls the actions that are carried out when the button "train_pu_model_push_button" is clicked by the user.

`clicked_update_ndocs_al()`
Updates the AL parameter that is going to be used for the resampling of the documents to be labelled based on the value specified by the user and shows the id, title, abstract and predicted class (if available) of each of the documents in a QTextEdit widget.

`do_after_evaluate_pu_model()`
Method to be executed after the evaluation of the PU model has been completed.

`do_after_give_feedback()`
Method to be executed after annotating the labels given by the user in their corresponding positions

`do_after_import_labels()`
Function to be executed after the labels' importing has been completed.

`do_after_load_corpus()`
Method to be executed after the loading of the corpus has been completed.

`do_after_reevaluate_model()`
Method to be executed once the reevaluation of the model based on the feedback of the user has been completed.

`do_after_retrain_model()`
Method to be executed once the retraining of the model based on the feedback of the user has been completed.

`do_after_train_classifier()`
: Method to be executed after the training of the classifier has been completed.

`execute_evaluate_pu_model()`
: Method to control the execution of the evaluation of a classifier on a secondary thread while the MainWindow execution is maintained in the main thread.

`execute_give_feedback()`
: Method to control the annotation of a selected subset of documents based on the labels introduced by the user on a secondary thread while the MainWindow execution is maintained in the main thread.

`execute_import_labels()`
: Imports the labels by invoking the corresponding method in the Task Manager object associated with the GUI.

`execute_load_corpus()`
: Method to control the execution of the loading of a corpus on a secondary thread while the MainWindow execution is maintained in the main thread.

`execute_reevaluate_model()`
: Method to control the execution of the reevaluation of a classifier on a secondary thread while the MainWindow execution is maintained in the main thread.

`execute_retrain_model()`
: Method to control the execution of the retraining of a classifier on a secondary thread while the MainWindow execution is maintained in the main thread.

`execute_train_classifier()`
: Method to control the execution of the training of a classifier on a secondary thread while the MainWindow execution is maintained in the main thread.

`init_feedback_elements()`
: Method for showing the documents to be annotated in the FEEDBACK TAB. The number of documents that is shown depends on the value assigned to self.n_docs_al; the remaining widgets that exist for showing documents until Constants.MAX_N_DOCS are hided while they are not used. The widgets that represent the document are displayed as empty spaces until the conditions for the annotation of the documents are met, i.e. a corpus and a set of labels have been selected

`init_ndocs_al()`
: Initializes the AL parameter in the text edit within the third tab of the main GUI window, i.e. n_docs. The default configuration of this parameter is read from the configuration file '/config/parameters.default.yaml'.

`init_params_train_pu_model()`
: Initializes the classifier parameters in the parameters' table within the second tab of the main GUI window, i.e. max_imbalance and nmax. The default configuration of these parameters is read from the configuration file '/config/parameters.default.yaml'.

`init_ui()`
: Configures the elements of the GUI window that are not configured in the UI, i.e. icon of the application, the application's title, and the position of the window at its opening.

`reset_params_train_pu_model()`
: Resets the PU model training parameters to its default value based on the values that were read initially from the configuration file

`show_corpora()`
: List all corpora contained in the source folder selected by the user.

`show_labels()`
: Method for showing the labels associated with the selected corpus.

`show_sampled_docs_for_labeling()`
Visualizes the documents from which the user is going to give feedback for the updating of a model.

`update_params_train_pu_model()`
Updates the classifier parameters that are going to be used for the training of the PU model based on the values read from the table within the second tab of the main GUI window that have been specified by the user.

CHAPTER

TEN

# ANALYZE KEYWORDS WINDOW

@author: lcalv

**class** `src.graphical_user_interface.analyze_keywords_window.`**`AnalyzeKeywordsWindow`**(*tm*)

Bases: `PyQt5.QtWidgets.QDialog`

Class representing the window that is used for the analysis of the presence of selected keywords in the corpus

**`__init__`**(*tm*)

Initializes a "AnalyzeKeywordsWindow" window.

**Parameters** **tm** (*TaskManager*) – TaskManager object associated with the project

**`center`**()

Centers the window at the middle of the screen at which the application is being executed.

**`do_analysis`**()

Performs the analysis of the keywords based by showing the "Sorted document scores", "Document frequencies" and "Keyword frequencies" graphs.

**`initUI`**()

Configures the elements of the GUI window that are not configured in the UI, i.e. icon of the application, the application's title, and the position of the window at its opening.

# CHAPTER ELEVEN

# CONSTANTS

@author: lcalv

`class src.graphical_user_interface.constants.Constants`

Bases: `object`

Class containing a series of constants for GUI configuration.

**`__weakref__`**

list of weak references to the object (if defined)

CHAPTER

# TWELVE

# GET KEYWORDS WINDOW

@author: lcalv

`class src.graphical_user_interface.get_keywords_window.GetKeywordsWindow(tm)`

Bases: `PyQt5.QtWidgets.QDialog`

Class representing the window that is used for the attainment of a subcorpus from a given list of keywords, this list being selected by the user.

`__init__(tm)`

Initializes a "GetKeywordsWindow" window.

**Parameters** **tm** (*TaskManager*) – TaskManager object associated with the project

`center()`

Centers the window at the middle of the screen at which the application is being executed.

`clicked_select_keywords()`

Method to control the actions that are carried out at the time the "Select keywords" button of the "Get keywords window" is pressed by the user.

`init_params()`

Initializes the keywords parameters in the parameters' table within the GUI's "Get keywords" window, i.e. wt, n_max, and s_min. The default configuration of these parameters is read from the configuration file '/config/parameters.default.yaml'.

`init_ui()`

Configures the elements of the GUI window that are not configured in the UI, i.e. icon of the application, the application's title, and the position of the window at its opening.

`show_suggested_keywords()`

Displays the corresponding keywords based on the configuration parameters selected by the user on the top QTextEdit "text_edit_show_keywords".

`update_params()`

Updates the keywords parameters that are going to be used in the getting of the keywords based on the values read from the table within the GUI's "Get keywords" window that have been specified by the user.

# CHAPTER THIRTEEN

# GET TOPICS LIST WINDOW

@author: lcalv

**class** `src.graphical_user_interface.get_topics_list_window.`**`GetTopicsListWindow`**(*tm*)

Bases: `PyQt5.QtWidgets.QDialog`

Class representing the window in charge of getting a subcorpus from a given list of topics, such a list being specified by the user.

**`__init__`**(*tm*)

Initializes a "GetTopicsListWindow" window.

**Parameters** **tm** (*TaskManager*) – TaskManager object associated with the project

**`center`**()

Centers the window at the middle of the screen at which the application is being executed.

**`clicked_get_topic_list`**()

Method to control the actions that are carried out at the time the "Select weighted topic list" button of the "Get topics window" is pressed by the user.

**`initUI`**()

Configures the elements of the GUI window that are not configured in the UI, i.e. icon of the application, the application's title, and the position of the window at its opening.

**`init_params`**()

Initializes the topics parameters in the parameters' table within the GUI's "Get topic list" window, i.e. n_max and s_min. The default configuration of these parameters is read from the configuration file '/config/parameters.default.yaml'.

**`show_topics`**()

Configures the "table_widget_topic_list" and "table_widget_topics_weight" tables to have the appropriate number of columns and rows based on the available topics, and fills out the "table_widget_topic_list" table with the id and corresponding chemical description of each of the topics.

**`update_params`**()

Updates the topics parameters that are going to be used in the getting of the keywords based on the values read from the table within the GUI's "Get topic list" window that have been specified by the user.

**`updated_topic_weighted_list`**()

Generates the topic weighted list based on the weights that the user has introduced on the "table_widget_topics_weight" table

CHAPTER

# FOURTEEN

# MESSAGES

@author: lcalv

`class src.graphical_user_interface.messages.`**`Messages`**

Bases: `object`

Class containing the majority of the messages utilized in the GUI to facilitate the readability of the code.

**`__weakref__`**

list of weak references to the object (if defined)

CHAPTER

FIFTEEN

# OUTPUT WRAPPER

@author: lcalv

**class** `src.graphical_user_interface.output_wrapper.`**`OutputWrapper`**(*parent*, *stdout=True*)

Bases: `PyQt5.QtCore.QObject`

Module that overrides the "sys.stderr" and "sys.stdout" with a wrapper object that emits a signal whenever output is written. In order to account for other modules that need "sys.stdout" / "sys.stderr" (such as the logging module) use the wrapped versions wherever necessary, the instance of the OutputWrapper are created before the TaskManager object.

It has been created based on the analogous class provided by: https://stackoverflow.com/questions/19855288/duplicate-stdout-stderr-in-qtextedit-widget

**`__getattr__`**(*self*, *str*) → object

**`__init__`**(*parent*, *stdout=True*)

CHAPTER

# SIXTEEN

# UTIL

@author: lcalv

`src.graphical_user_interface.util.`**`change_background_color_text_edit`**`(`*text_edit*, *prediction*`)`

Method to that changes the border color of the QTextEdit associated with the documents to be annotated, based on the predicted class selected by the user.

**Parameters**

- **text_edit** (*QTextEdit*) – QTextEdit (document) whose border color is going to be updated based on the
- **prediction** (*int (0 or 1)*) – Predicted class specified by the user. If prediction == 1, the document's associated QTextEdit's border color is set to #6A7288; if prediction == 0, it is set to #DB8678.

`src.graphical_user_interface.util.`**`execute_in_thread`**`(`*gui*, *function*, *function_output*, *progress_bar*`)`

Method to execute a function in the secondary thread while showing a progress bar at the time the function is being executed if a progress bar object is provided. When finished, it forces the execution of the method to be executed after the function executing in a thread is completed. Based on the functions provided in the manual available at: https://www.pythonguis.com/tutorials/multithreading-pyqt-applications-qthreadpool/

**Parameters**

- **function** (*UDF*) – Function to be executed in thread
- **function_output** (*UDF*) – Function to be executed at the end of the thread
- **progress_bar** (*QProgressBar*) – If a QProgressBar object is provided, it shows a progress bar in the main thread while the main task is being carried out in a secondary thread

`src.graphical_user_interface.util.`**`signal_accept`**`(`*progress_bar*`)`

Makes the progress bar passed as an argument visible and configures it for an event whose duration is unknown by setting both its minimum and maximum both to 0, thus the bar shows a busy indicator instead of a percentage of steps.

**Parameters** **progress_bar** (*QProgressBar*) – Progress bar object in which the progress is going to be displayed.

`src.graphical_user_interface.util.`**`toggle_menu`**`(`*gui*, *max_width*`)`

Method to control the movement of the Toggle menu located on the left. When collapsed, only the icon for each of the options is shown; when expanded, both icons and name indicating the description of the functionality are shown. Based on the code available at: https://github.com/Wanderson-Magalhaes/Toggle_Burguer_Menu_Python_PySide2/blob/master/ui_functions.py

**Parameters**

- **gui** (*MainWindow*) – MainWindow object to which the toggle menu will be appended.

- **maxWidth** (*int*) – Maximum width to which the toggle menu is going to be expanded.

CHAPTER

# SEVENTEEN

# WORKER SIGNALS

Created on Tue Mar 2 13:19:34 2021 @author: lcalv

`class src.graphical_user_interface.worker_signals.WorkerSignals`

Bases: `PyQt5.QtCore.QObject`

Module that defines the signals that are available from a running worker thread, the supported signals being "finished" (there is no more data to process), "error", "result" (object data returned from processing) and "progress" (a numerical indicator of the progress that has been achieved at a particular moment). It has been created based on the analogous class provided by: https://www.pythonguis.com/tutorials/multithreading-pyqt-applications-qthreadpool/

# CHAPTER EIGHTEEN

## WORKER

Created on Tue Mar 2 13:19:34 2021 @author: lcalv

`class src.graphical_user_interface.worker.Worker(fn, *args, **kwargs)`

Bases: `PyQt5.QtCore.QRunnable`

Module that inherits from QRunnable and is used to handler worker thread setup, signals and wrap-up. It has been created based on the analogous class provided by: https://www.pythonguis.com/tutorials/multithreading-pyqt-applications-qthreadpool/.

`__init__(fn, *args, **kwargs)`

Initializes the application's main window based on the parameters received from the application's starting window.

**Parameters**

- **callback** (*UDF*) – The function callback to run on this worker thread. Supplied args and kwargs will be passed through to the runner.
- **callback** (*UDF*) – Function
- **args** (*list*) – Arguments to pass to the callback function
- **kwargs** (*dict*) – Keywords to pass to the callback function

`run()`

Initialises the runner function with passed args, kwargs.

CHAPTER

# NINETEEN

## INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

S

src.base_taskmanager, 3
src.data_manager, 9
src.domain_classifier.classifier, 13
src.domain_classifier.custom_model, 15
src.domain_classifier.preprocessor, 17
src.graphical_user_interface.analyze_keywords_window, 27
src.graphical_user_interface.constants, 29
src.graphical_user_interface.get_keywords_window, 31
src.graphical_user_interface.get_topics_list_window, 33
src.graphical_user_interface.main_window, 23
src.graphical_user_interface.messages, 35
src.graphical_user_interface.output_wrapper, 37
src.graphical_user_interface.util, 39
src.graphical_user_interface.worker, 43
src.graphical_user_interface.worker_signals, 41
src.menu_navigator.menu_navigator, 1
src.query_manager, 11
src.task_manager, 5