README - IPB OPTLEARN ALGORITHM

Overview

This code is designed to select the next question to be asked on the platform based on the user's knowledge level, response history, current data and user action. It utilizes a caching mechanism to avoid redundant requests to Google Sheets and optimizes question selection using a random forest model.

Dependencies

The code requires the following dependencies:

- pygsheets
- pandas
- numpy
- random
- RF_functions

Ensure that these dependencies are installed in your Python environment before running the code.

Class: Cache_operations

This class serves as a singleton cache for storing and retrieving data from Google Sheets. It contains methods for loading data, getting authorization, retrieving data from Google Sheets, and fetching questions and answers. It also includes a method to create a table of questions and their corresponding keywords, levels, correct answers, and correct percentages.

Class Attributes

- google_key_file: The file path to the Google Sheets key file.
- question_levels: The file path to the local file containing question levels.
- data_sheet_question: The name of the worksheet in the Google Sheets that contains the questions.
- data_sheet_answer: The name of the worksheet in the Google Sheets that contains the answers.
- data_cache: A cache to store the Google Sheets data.
- cache_questions: A DataFrame to store the cached questions.
- cache_answer: A DataFrame to store the cached answers.
- count: A counter to keep track of the number of API calls made to Google Sheets.

Class Methods

__init__(self): The class constructor, set as private to prevent incorrect usage of the singleton class.

get_instance(cls): Returns the singleton instance of the class.

load(self): Loads the initial configurations and questions.

__get_google_authorization(self): Retrieves authorization to access the Google Sheets.

get_data(self): Retrieves the Google Sheets data and caches it.

get_questions(self, update=False): Retrieves the questions from Google Sheets and merges them with the local file of question levels. Returns a DataFrame with the columns [question, answer, level].

create_table(self): Creates a table of questions with their keywords, levels, correct answers, and correct percentages.

get_answers(self, update=False): Retrieves all the answers from the specified worksheet in Google Sheets, processes the data, and returns a DataFrame with columns [email, id, test, correct].

create_table(self): Creates a table of questions with their keywords, levels, correct answers, and correct percentages.

Function: format_current_data(current_data)

This function takes the current data from the platform and formats it into the user's email and a DataFrame representing the user's answers. It filters out skipped questions and merges the answers with the cached questions. It returns the email and the formatted DataFrame.

Function: get_user_history(user_email)

This function takes the user's email and retrieves their response history from the cache. It returns a filtered DataFrame with columns [question, correct].

Function: get_last_question(current_data, user_history)

This function takes the current data and the user's response history. It combines these two sources to determine the last question answered by the user. It returns a DataFrame with the columns [question, correct].

Function: get_user_level(sheet_last_question)

This function takes the last question answered by the user and determines their question level based on the correctness of the answer. It returns an integer representing the user's question level.

Function: is_first_question(current_data)

This function checks if the current question is the first question asked by the user. It returns a boolean value.

Function: debugg_function(email, level, hist, next)

This function is used for debugging purposes. It prints out relevant information such as the user's email, question level, last question answered, correctness of the last question, and the next question to be asked.

Function: next_question_choose(level, historic, current_data, platform_user_action)

This function selects the next question to ask based on the user's question level, response history, current data, and user action on the platform. It uses a random forest model calling the giuliaRF function to do the question selection. It returns the ID of the next question to be asked with the highest probability of getting it right.

Function: giulia_RF(plattaform_user_action, list_questions, level)

This function is used within the next_question_choose function. It receives the user action data, a list of available questions, and the user's question level. It calls a random forest model to predict the probabilities of each question being the next question. It returns the ID of the next question to be asked.

Function: entrypoint(plattaform_user_action)

This is the main entry point function that receives the user action data from the platform. It formats the current data, checks if it's the first question, retrieves the user's response history, gets the last question answered, determines the user's question level, combines the user's full history, and selects the next question to be asked. It returns the ID of the next question. Its input parameter comes from

"app.py", which is responsible for making the web interface for questions and capturing user actions. Your call is on line 218.

Example Usage

The data that the function expects are:

- Name;
- Surname;
- College;
- Email;
- First answer to the satisfaction question;
- Second answer to the satisfaction question;
- Third answer to the satisfaction question;
- Fourth answer to the satisfaction question;
- First question;
- Answer to the first question;
- Second question;
- Answer to the second question;
- Third question;
- Answer to Third Question;
- Fourth question;
- Answer to the fourth question;
- Fifth question;
- Answer to the fifth question.

This information must be passed as a list. In the following way:

['Name','Surname','College','Email','-1','-1','-1','-1','-1','-1','-1','-1','-1','-1','-1','-1','-1','-1']

This way, the return of the function will be an integer value corresponding to the ID of a question chosen for the user based on all the parameters of importance chosen and analyzed.

Example Usage:

app.py

```
def algorithm_for_new_question(output_row):
 return proto_three_GiuliaIPB_TabelaPorcentagem.entrypoint(output_row)
```

proto_three_GiuliaIPB_TabelaPorcentagem.py

```
def entrypoint(plattaform_user_action)
```

```
.
.
.
return next_question
```