Name: Wang, Wentao Date: 12/21/2022

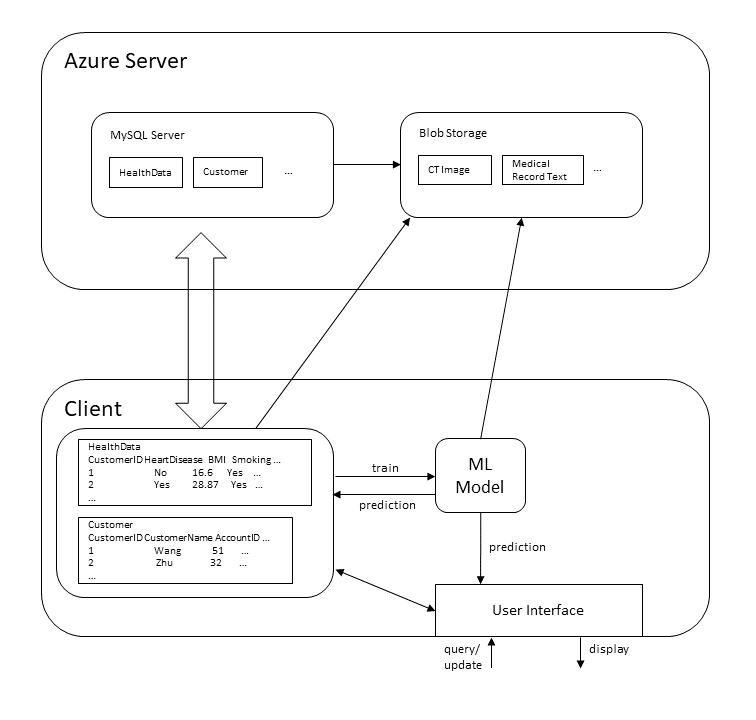
NYU ID: N13813465

# Project Report

## Reference architecture

The goal of this project is to manage data about heart disease, which could be helpful for an insurance company to estimate the health risk of customers and develop/adapt their insurance products.

The reference architecture is presented in the following diagram:



The client program interacts with both the user (through the user interface) and the Azure server (upload/download data and model, database management). The user interface will check and ensure the well-formedness of the input data, therefore data quality is ensured. All the data and model is stored not only on the local machine but also on the Azure server, and accessing the Azure server will require authentication, so the prevention of data losses and leakage is also achieved.

Structured data, like those in tabular forms, are stored in both Pandas DataFrames on the client machine and tables in the SQL database on the Azure MySQL server. Unstructured data accompanying the structure ones, like image, text and model parameters, could be stored in blob storage on the Azure server, and linked by a unique blob id in the structured data. (Though I currently have no unstructured data in my illustration.)

The logical schema of the database is adaptable to the actual data you use. In fact, the columns of each table is transformed from the .csv data used by the user. For example, Yes/No datatype will be transformed to bool in Python and BOOL in SQL, and age range will be transformed to integer values preserving the age order. I will present the specific definition of the table created from the data I use.

The Machine Learning model is used to provide insights into the data. It can predict the chance of a customer to get heart disease given his/her heath data. It is trained on the data we have in the database, and will be retrained after accumulating enough updates on data.

Through the user interface, the user can query, add, and update the data. The result will be presented to the user after the operation is completed. As mentioned above, the input data is checked by the client program, and if the data is faulty, the user will have to re-input the data.

## Details

The code in the GitHub repository: <https://github.com/wwt17/Database-Project>. You may check out the README to find out how to prepare for the environments and run the client program.

Specifically, I use Python to implement the client program. The dataset I use for illustration is a kaggle dataset named “Personal Key Indicators of Heart Disease” (<https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease>). This dataset is stored in a .csv table, which should be manually downloaded by you and will be loaded by my program into a Pandas DataFrame in the memory of the client program, for both uploading to the Azure server initial tuples and training the Machine Learning model. Other datasets in different forms are also possible. The data is automatically uploaded to and downloaded from the Azure server according to the operations requested by the user through the interface. When the client program is uploading the data, it will first create a table for this data. If you run the program, you will find it is executing the following SQL command, in which datatypes are automatically inferred and transformed to SQL datatypes:

CREATE TABLE data (

id serial PRIMARY KEY,

HeartDisease BOOL,

BMI DOUBLE,

Smoking BOOL,

AlcoholDrinking BOOL,

Stroke BOOL,

PhysicalHealth DOUBLE,

MentalHealth DOUBLE,

DiffWalking BOOL,

Sex ENUM('Female', 'Male'),

AgeCategory SMALLINT,

Race ENUM('American Indian/Alaskan Native', 'Asian', 'Black', 'Hispanic', 'Other', 'White'),

Diabetic ENUM('No', 'No, borderline diabetes', 'Yes', 'Yes (during pregnancy)'),

PhysicalActivity BOOL,

GenHealth SMALLINT,

SleepTime DOUBLE,

Asthma BOOL,

KidneyDisease BOOL,

SkinCancer BOOL);

The Machine Learning model is simply a two-layer perceptron, implemented in *models.py* using PyTorch. Training the model can take a lot of time and compute, so it would be costly to retrain the model every time the database is changed. Therefore, only after accumulating certain amount of updates on the database will the program automatically retrain the model on all existing data. After training, the model parameters are also uploaded to the Azure blob storage for backup.

The user interacts with the client program through the user interface. The user can request following operations through this interface:

* use SQL to query the database;
* use the Machine Learning model to predict the chance of getting the disease given other health data of a customer;
* add health data of a customer to the database;
* delete health data of a customer from the database;
* reinitialize everything: reset the database state to the initial state (and re-train the Machine Learning model);
* explicitly request a re-train of the Machine Learning model.

The detail usage of these functions are explained in the following section.

## Interface usage

The interface I provided is CLI, which is mainly due to my limited time. However, I think my CLI interaction is good enough for you to explore my client program and find some insights into the project.

As the README suggested, in order to setup everything needed, you should first use pip to install required Python packages. You should also download the .csv dataset file from the [kaggle website](https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease) (and rename it to *data.csv* or any other name you like). Any other csv files are also possible, but my client program will assume that the first column of the csv file is the disease label, which is to be predicted by the Machine Learning model.

After having the proper environment, you can run

python main.py [-–csv-file CSV\_FILE] …

to start the client program. The CSV\_FILE, which default to *data.csv*, is exactly the csv file you just downloaded and want to use in your case. More command line options and detailed explanations of the usage can be shown by running python main.py –h.

After starting the program, it will load data from the csv file into a Pandas DataFrame, identify the columns, and automatically recognize and adapt the datatype of each column. For example, Yes/No values will be converted to Booleans, and categorical labels like gender will be converted to categorical datatypes, which correspond to ENUM datatypes in SQL.

Then, it will build connections to the Azure servers, including both the blob container and the MySQL database server. For the former, I already included in my code the authentication, but the MySQL database server would prompt you to input the password, which is “DataBase2022”. This step illustrates the data security mechanism I provided.

After setup connections to the Azure servers, it will then build and (if --load\_model is not specified) train the Machine Learning models on the data it has. Uploading tuples to the Azure MySQL database is time-consuming, so I have already uploaded some initial number of tuples to the database. If you want to reset everything (the database and the model), you may use the reinit operation later.

After all these initialization steps, the client program will turn into an interactive mode. In each turn you are prompted to input the operation (query/predict/add/delete/reinit/retrain/quit) you want to perform. Here I explain each of them in detail:

* query: You will be prompted to enter an SQL query on the table, like “SELECT \* FROM data;”. The interaction will feels like you are interacting with an MySQL client. The results of the query will be displayed in a Pandas DataFrame form.
* predict: You can call the Machine Learning model to predict the chance of getting the disease given the health data of a customer. To provide the customer data, you will prompted to input either
  + all health data fields (not predicted), or
  + simply an integer number indexing the DataFrame from the csv file.

The latter option is to provide an easier way for illustration, since inputting all the fields is tiresome. However, you can still try out the first option, which will interactively check the data you input. This is the way I ensure the well-formedness of the data and data quality. If you want to terminate the data inputting process, you can send an EOF (Ctrl-D), so this operation is canceled.

The client will show the prediction of the probability of getting the disease. If an integer index is provided, it will also show the ground-truth label in the tuple. Due to the small amount of data, the very unbalanced data distribution, and the simple architecture of the model, the prediction is reasonable but not very accurate.

* add: You can add health data of a customer, which is a tuple, to the table in the database. The way you input the data is similar to the way in the predict operation. After providing the data, the program will display the tuple and ask to you confirm the add operation. You may later use query operation to find out the change.
* delete: You are asked to complete an SQL DELETE command by completing the condition of tuples to delete. Usually, if you want to delete only one tuple, you can use condition like “id=?”. You may also later use query operation to find out the change.
* retrain: This explicitly requests a re-train of the Machine Learning model. The program will automatically re-train the model after enough number of updates (add/delete). The number can be specified by --retrain-interval option. However, as mentioned earlier, training the model is time and compute-consuming, so the number should be set to a large value if you want to save time.
* reinit: As I mentioned earlier, this will reset the state of the database and the model to their initial state, which is uploading several number of initial tuples to the database and train the model on them.
* quit: You can quit the client program. Also, sending an EOF (Ctrl-D) has the same function. This is similar to any interactive program in CLI.