

# HW-6 WRITE UP

Mohit Sai Gutha

[mohitsai@bu.edu](mailto:mohitsai@bu.edu)

U48519832

## Instructions to Run

### 1. Prerequisites

- Database generated after the completion of previous homework.
- A snapshot of the server vm from previous homework would be helpful as it has all the necessary dependencies related to sql that would be useful for this homework as well. (OPTIONAL)
- Knowledge on the database schema
- mysql instance should be running

### 1. Steps to Run

#### a) Creating and setting up a VM for HW-6 :

One of the requirements of the homework is to run the program on a vm that is created for this homework in particular.

The Server VM from the previous homework has all the necessary dependencies for the sql part of this homework, so creating a vm from the snapshot of this older vm instance would save a lot of time and resources.

The VM can also be created from scratch. After the instance is created we will need to install and set up the database client for mysql using the steps provided in Lecture 12 Slides - Slide 9. Install all the necessary dependencies given in slide 10.

Once the instance is set up to be able to connect to a mysql database we will need to install the dependencies to perform the model related tasks.

To install these dependencies copy requirements-hw6.txt file into the instance and run the following command :

```
pip install -r requirements-hw6.txt
```

#### b) Running the program :

Once the vm instance is set up. Copy hw6.py into the vm. Activate python virtual environment (optional) and then run the following command :

```
python hw6.py
```

#### c) Notes :

It is important to note that the db instance used for running the original program has been deleted (to conserve resources) and therefore to run the program on a different db client, we will need to change the values of the constants for the db credentials in the program.

## Design and Implementation (HW-6 Files)

### Additional resources used are:

Model training and evaluation : pandas; scikit-learn (sklearn);

### Key Components, Workflow and Design Decisions

#### 1. Database Connection

The code for database connection remains the same as the previous homework.

#### 2. Data retrieval and storage

The SQL query "SELECT \* FROM requests;" is executed to pull the data into a DataFrame. The data is saved to a CSV file (requests.csv) for quick reloading and reproducibility. This also helps us avoid repeated database queries and ensures offline access to data.

#### 3. Data Preprocessing

Label Encoding: The project uses LabelEncoder to convert categorical columns (client\_ip, client\_country, etc.) into numerical format, which is a requirement for most machine learning algorithms. This enables the models to process non-numeric data effectively.

Timestamp Conversion: The request\_time column is transformed from datetime format to seconds since the epoch to ensure it is suitable for numerical modeling.

#### 4. Model 1 : Predicting Client Country

The only feature used is client\_ip\_encoded to predict country\_encoded. (Part of the requirements of the hw).

Model Choice: A RandomForestClassifier is chosen for its robustness and ability to handle high variance. The choice was made after also trying out different models such as knn, xgboost and lightgbm which all failed to match the accuracy provided by the random forest model. GridSearchCV was used to find the following optimal hyperparameters for the model:

- n\_estimators=100: Specifies the number of trees.
- min\_samples\_split=5, min\_samples\_leaf=2, and max\_depth=30: These parameters control the tree's growth and complexity to avoid overfitting.
- bootstrap=False: Indicates that bootstrap sampling is not used.

#### 5. Model 2 : Predicting Income

Based on the Correlation matrix that I had constructed for the dataset for the income field, there seemed to be no correlation between income in any other fields except age. So initially I used only the age to predict target variable income. But this never yielded the desired accuracy of 40% and above. Then credit to piazza post @181 I realized that I need to use all fields as there are plenty of other spurious correlations.

The process of choosing the random forest model and hyperparameter tuning was similar to that of the previous model.

#### 6. Model Training and Evaluation

A 80-20 split is used for Model 1, while a 70-30 split is used for Model 2 as the train-test split of the dataset.

Evaluation: Accuracy is computed to gauge the models' performance and also since that is the desired parameter for the hw. The `evaluate_model()` function abstracts training and prediction steps, allowing code reuse and consistency.

## 7. Saving Models

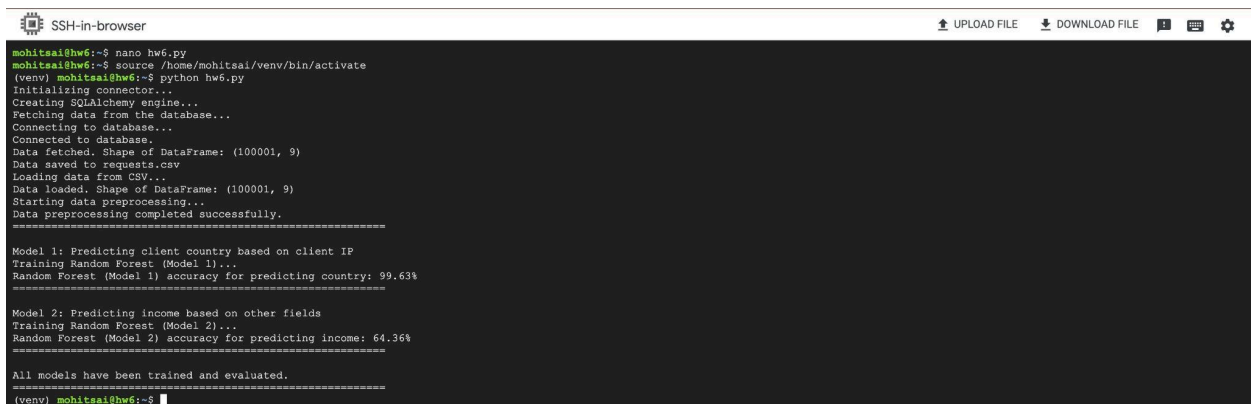
We save both the trained models as python objects in serialized format using pickle, to use it for future applications or for testing purposes.

## 8. Limitations :

- We used label encoding for the categorical columns but label encoding may introduce ordinal relationships that don't exist. This might be misleading for certain columns where order doesn't have a natural meaning. But it is justified in this context as the data relationships are not very complex and the size of the dataset isn't too big.
- Using only `client_ip_encoded` as a feature to predict the `client_country` may be oversimplified. Additional contextual features might improve performance. But this was done as it was one of the requirements.
- The current code isn't too scalable and may face performance issues with very large datasets due to in-memory processing.
- Along with saving the models, I also tried to save confusion matrices for the both the models to understand how well the model is working on different classes and potentially giving more data for classes that it was underperforming for but since the countries class had too many categories, I failed to achieve a legible confusion matrix for the first model. The confusion matrix for model 2 has been saved and it seems to perform almost consistently on all income categories.

## Demonstration of Implementation

Output of the program :



```
mohitsai@hw6:~$ nano hw6.py
mohitsai@hw6:~$ source /home/mohitsai/venv/bin/activate
(venv) mohitsai@hw6:~$ python hw6.py
Initializing connector...
Creating SQLAlchemy engine...
Fetching data from the database...
Connecting to database...
Connected to database.
Data fetched. Shape of DataFrame: (100001, 9)
Data saved to requests.csv
Loading data from CSV...
Data loaded. Shape of DataFrame: (100001, 9)
Starting data preprocessing...
Data preprocessing completed successfully.

Model 1: Predicting client country based on client IP
Training Random Forest (Model 1)...
Random Forest (Model 1) accuracy for predicting country: 99.63%

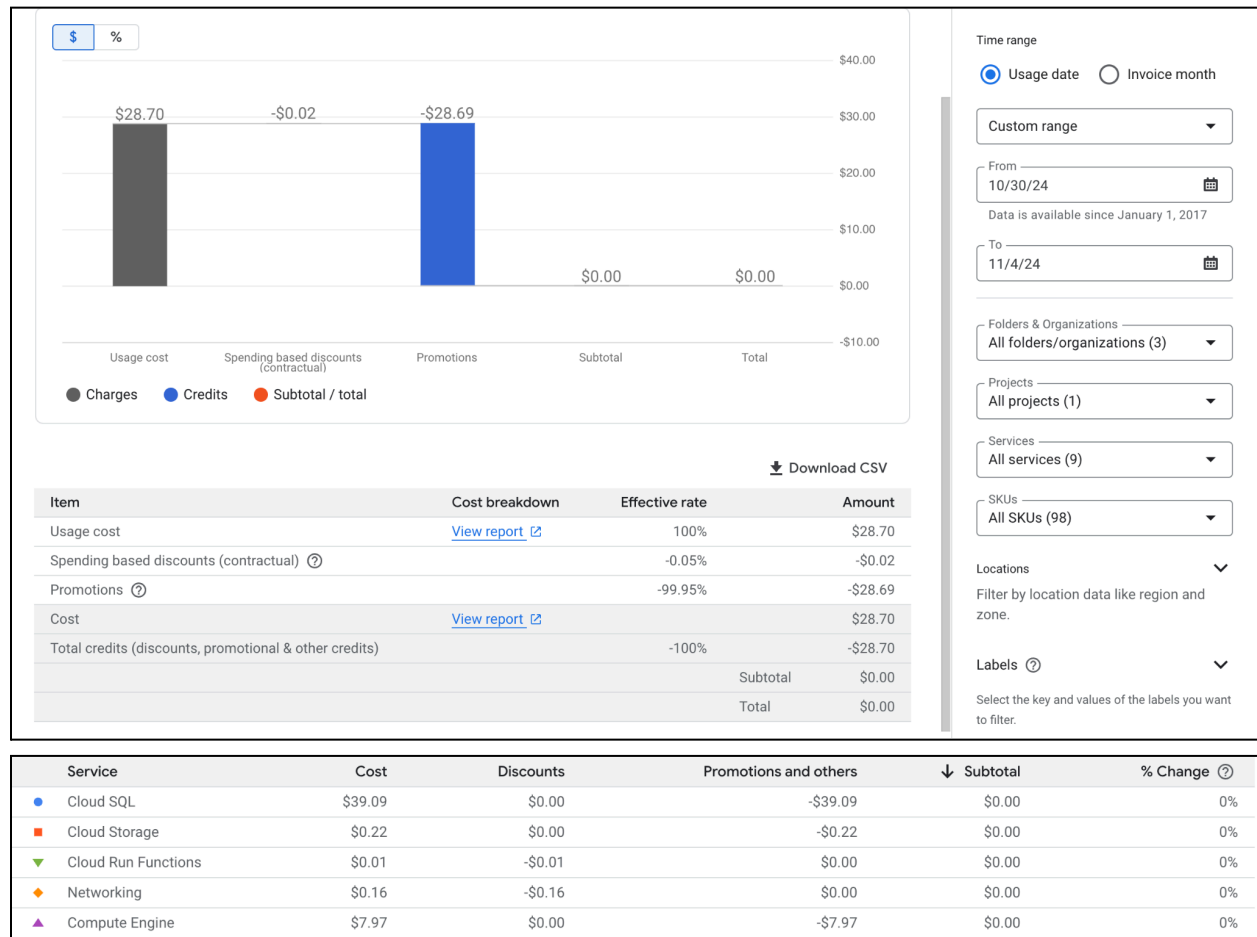
Model 2: Predicting income based on other fields
Training Random Forest (Model 2)...
Random Forest (Model 2) accuracy for predicting income: 64.36%

All models have been trained and evaluated.
(venv) mohitsai@hw6:~$
```

**Model 1** surpassed the desired accuracy benchmark of 99% and gave an accuracy of **99.63%**.

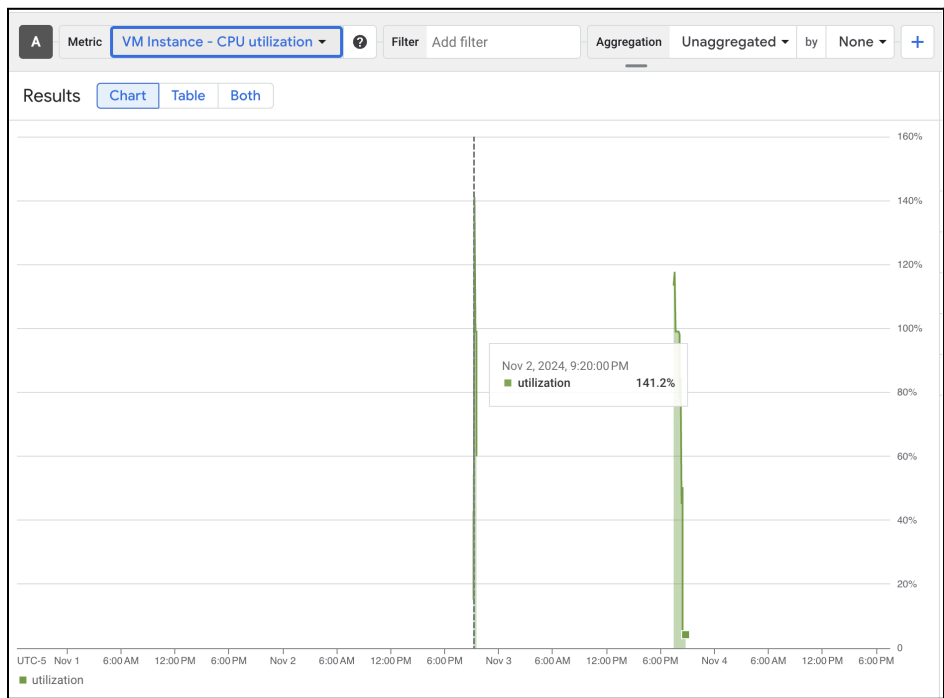
**Model 2** surpassed the desired accuracy benchmark of 40% and gave an accuracy of **64.36%**.

Billing report generated for use related to this homework :

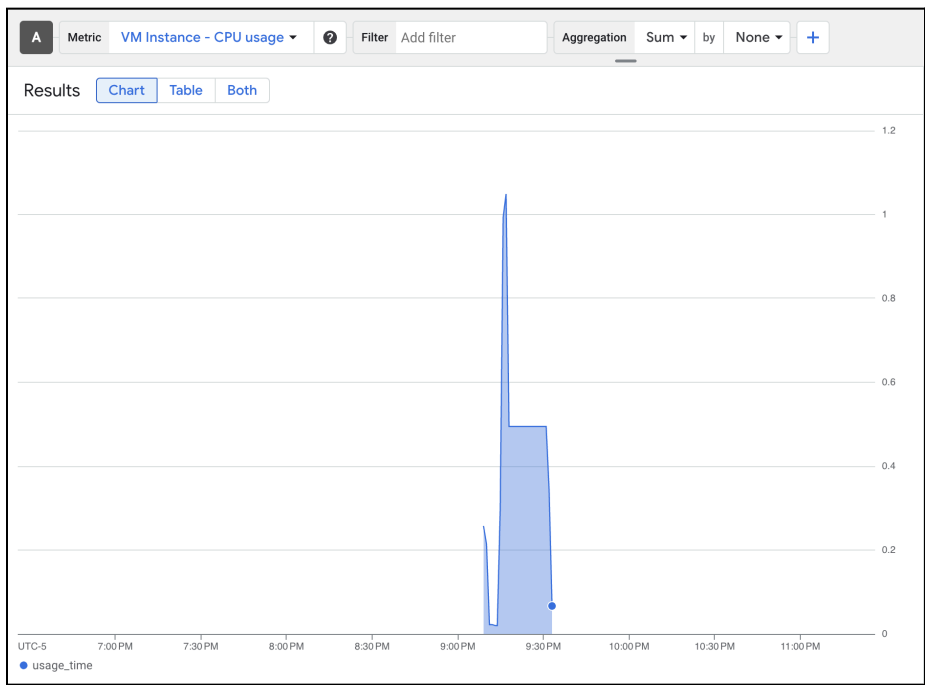


For Compute Engine, I incurred a cost of \$7.97 and I incurred \$39.09 for cloud sql. This was definitely more than what was warranted for the task. I failed to turn off the db instance in time and therefore incurred high costs for running the instance and also sending multiple queries to the instance for the program. Also the high charges for the compute engine is due to the high computing resources used by training and testing these models. I need to be wary of the costs moving forward to make sure I do not go beyond the allotted \$200.

The following graphs show the high compute engine usage for this particular hw :



The above graph shows the high utilization of the vm instance cpu resource while training and evaluating the model.



The high CPU usage also signifies the substantial computational power that was required for this hw.