Cloud Functions and Storage

Applications of Cloud Computing and Big Data - ECON 446

Bella Rakhlina
Lora Yovcheva
Mauricio Vargas-Estrada

Master Of Quantitative Economics
University of California - Los Angeles

In [ ]:
```
import requests
import os
import joblib
import subprocess
import pandas as pd
import numpy as np
import json

from google.cloud import storage
from io import StringIO, BytesIO
from toolz import pipe


from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
```

First Cloud Function

a.)

Post a cloud function that takes in a string of numbers and returns a json file that contains the the sum of all of the single digit numbers.

```
Example : input ="12345"
output = 1+2+3+4+5 = 15
returns({"answer":15})
```

The source code is available in the following GitHub repository:

ECON446-ORG/first-cloud-function

The cloud functions is deployed directly from the GitHub repository to Google Cloud Platform using Cloud Build and a trigger tracking commits to the `main` branch.

b.)

Query your cloud function using requests for example input "012937", "2" and "9999999999999".

In [ ]:
```
url = 'https://us-west2-my-first-function-422520.cloudfunctions.net/first_cloud_function'
```

In [ ]:
```
print(requests.post(url, data='012937').content.decode('utf-8'))
```

```
{"numbers": "012937", "sum": 22}
```

In [ ]:
```
print(requests.post(url, data='2').content.decode('utf-8'))
```

```
{"numbers": "2", "sum": 2}
```

In [ ]:
```
print(requests.post(url, data='9999999999999').content.decode('utf-8'))
```

```
{"numbers": "9999999999999", "sum": 117}
```

c.)

Add srborghese@g.ucla.edu to your cloud project via IAM.

User added in `iam` with `Viewer` role.

Automated Webscraping

a.)

Find a website that is scrapable with Beautiful soup that updates with some frequency. Build a cloud function to programatically scrape the useful content.

The source code is available in the following GitHub repository:

ECON446-ORG/automated-webscraping

The cloud functions is deployed directly from the GitHub repository to Google Cloud Platform using Cloud Build and a trigger tracking commits to the `main` branch.

The webscraped data is directly stored in a Google Cloud Storage bucket, and a scheduler is used to trigger the cloud function every day at 12:00 UTC.

b.)

Query your stored files.

In [ ]:
```
url = 'https://us-central1-automated-webscraping.cloudfunctions.net/main'
```

In [ ]:
```
print(requests.get(url).content.decode('utf-8'))
```

```
Webscraping Old School RuneScape Gold Prices
========================================
Alltime: From 2023-06-04 21:16:00 to 2024-06-03 18:00:00
90 Day: From 2024-03-05 21:15:00 to 2024-06-03 18:00:00
30 Days: From 2024-05-04 21:15:00 to 2024-06-03 18:00:00
7 Days: From 2024-05-27 21:15:00 to 2024-06-03 18:00:00
1 Day: From 2024-06-02 21:15:00 to 2024-06-03 18:00:00
========================================
```

In [ ]:
```
df = pipe(
    storage.Client(),
    lambda x: x.bucket('main_webscraping'),
    lambda x: x.blob('1 Day.csv'),
    lambda x: x.download_as_string(),
    lambda x: pd.read_csv(StringIO(x.decode('utf-8'))),
)
```

In [ ]:
```
print(df)
```

```
                 date  price
0  2024-06-02 21:15:00  0.197
1  2024-06-02 23:00:00  0.204
2  2024-06-03 02:45:00  0.197
3  2024-06-03 03:45:00  0.192
4  2024-06-03 05:00:00  0.186
5  2024-06-03 13:45:00  0.192
6  2024-06-03 18:00:00  0.204
```

c.)

State how this could be useful in a business setting.

The RuneScape is a free-to-play massively multiplayer online role-playing game. We web scraped the price of OSRS (Old School RuneScape) Gold, which serves as the primary means to facilitate trades, elevate skill levels, obtain powerful equipment, and indulge in a multitude of entertaining activities in Old School RuneScape. According to Jagex, the developer of RuneScape, selling gold and account names is illegal, but the demand and supply for these goods still exist. On the supply side, Venezuelans, who are facing a severe socioeconomic and political crisis that has led to hyperinflation, are playing RuneScape for up to 10 hours a day. This is because mining gold in the game for 10 hours can be more profitable than working two weeks in their local economy. For gold sellers, staying updated on the daily prices is crucial. This information can help them maximize profits by timing their sales when prices are high. Additionally, businesses that operate in the secondary market for virtual goods can use this data to make informed decisions about buying, selling, and trading OSRS gold. Knowing the trends and fluctuations in gold prices can help them optimize their inventory, set competitive prices, and anticipate market movements.

Machine Learning Model

a.)

Build some machine learning model using scikit learn and make it queriable using cloud functions.

Fitting the Model

In [ ]:
```
pd.options.display.float_format = '{:,.0f}'.format
```

In [ ]:
```
data = pd.read_csv('data/healthcare-dataset-stroke-data.csv')
```

In [ ]:
```
print(data.head(5))
```

```
      id  gender  age  hypertension  heart_disease ever_married  \
0   9046    Male   67             0              1          Yes
1  51676  Female   61             0              0          Yes
2  31112    Male   80             0              1          Yes
3  60182  Female   49             0              0          Yes
4   1665  Female   79             1              0          Yes

       work_type Residence_type  avg_glucose_level  bmi   smoking_status  \
0        Private          Urban                229   37  formerly smoked
1  Self-employed          Rural                202  NaN     never smoked
2        Private          Rural                106   32     never smoked
3        Private          Urban                171   34           smokes
4  Self-employed          Rural                174   24     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1
```

In [ ]:
```
data['age'] = data['age'].round()
data['avg_glucose_level'] = data['avg_glucose_level'].round()
data['bmi'] = data['bmi'].round()
data = data[data['gender'] != 'Other']
```

In [ ]:
```
value_counts = data['smoking_status'].value_counts()
print("count of each unique value in 'smoking_status' column:")
print(value_counts)
```

```
count of each unique value in 'smoking_status' column:
smoking_status
never smoked       1892
Unknown            1544
formerly smoked     884
smokes              789
Name: count, dtype: int64
```

In [ ]:
```
data['gender'] = data['gender'].replace({'Male': 0, 'Female': 1})
data['smoking_status'] = data['smoking_status'].replace(
    {'never smoked': 1, 'Unknown': 2,'formerly smoked': 3, 'smokes': 4 }
)
```

In [ ]:
```
imputer = KNNImputer(n_neighbors=5)
data[['bmi']] = imputer.fit_transform(data[['bmi']])
```

In [ ]:
```
data_select = data[
    [
        "age",
        "gender",
        "heart_disease",
        "avg_glucose_level",
        "bmi",
        "smoking_status",
        "stroke"
    ]
]
```

In [ ]:
```
y = data[["stroke"]]
X = data_select.drop(columns=["stroke"])
```

In [ ]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)
clf = MLPClassifier(
    hidden_layer_sizes=(10,100,100,),
    max_iter=1000,
    random_state=42
)
clf.fit(X_train,y_train)
```

```
/home/m4wnn/anaconda3/envs/web-env/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for exampl
e using ravel().
  y = column_or_1d(y, warn=True)
```

Out[ ]:
```
MLPClassifier(hidden_layer_sizes=(10, 100, 100), max_iter=1000, random_state=42)
```

Querying the model from the cloud storage bucket.

In [ ]:
```
def load_scikit_model(file_name):
    bucket_name = "stroke_prediction123"
    source_blob = "stroke/" + file_name

    os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "Gcredentials.json"
```

```
    client = storage.Client()

    bucket = client.get_bucket(bucket_name)
    blob = bucket.blob(source_blob)

    model_data = blob.download_as_string()

    model = joblib.load(BytesIO(model_data))
    return(model)
```

In [ ]:
```
model = load_scikit_model("stroke_NN.sav")
```

In [ ]:
```
preproc = load_scikit_model("stroke_scaler.sav")
```

Deployed Cloud Function source code

The entry point is the `stroke_presence` function, which receives a JSON object with the following keys: `age`, `gender`, `heart_disease`, `avg_glucose_level`, `bmi`, and `smoking_status`. The function loads the model and the preprocessor from the cloud storage bucket, preprocesses the input data, and returns the prediction and the probability of a stroke.

```
import warnings
import google
import joblib
import pandas as pd
import requests
import sklearn
from urllib.parse import parse_qs
from google.cloud import storage
import os
from io import StringIO
from joblib import load
from io import BytesIO
from flask import jsonify

def stroke_presence(request):
    print("Models")
    try:
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", UserWarning)
            model = load_scikit_model("stroke_NN.sav")
            preproc = load_scikit_model("stroke_scaler.sav")
            print("Models Loaded!")


            print(request)
            dictionary = request.get_json()
            print(dictionary)


            required_keys = ['age', 'gender', 'heart_disease', 'avg_glucose_level', 'bmi', 'smoking_status']
            missing_keys = [key for key in required_keys if key not in dictionary]
            if missing_keys:
                raise ValueError(f"Missing required parameter(s): {', '.join(missing_keys)}")

            age = float(dictionary['age'])
            gender = int(dictionary['gender'])
            heart_disease = int(dictionary['heart_disease'])
            avg_glucose_level = float(dictionary['avg_glucose_level'])
            bmi = float(dictionary['bmi'])
            smoking_status = int(dictionary['smoking_status'])

            print("Variables Set")


            X = preproc.transform([[age, gender, heart_disease, avg_glucose_level, bmi, smoking_status]])
            predictions = model.predict(X)[0]
            probability = str(round(model.predict_proba(X)[0][1] * 100, 2)) + "%"
            print("Probabilities Calculated")
            print(predictions)
            print(probability)

            return jsonify({
                "prediction": int(predictions),
                "status": 200,
                "prob_of_stroke": probability
            })
    except Exception as e:
        print(e)
        return jsonify({"status": "error", "message": str(e)})

def load_scikit_model(file_name):
    bucket_name = "stroke_prediction123"
    source_blob = "stroke/" + file_name

    os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "Gcredentials.json"
    client = storage.Client()
```

```
    print("Client Created")
    bucket = client.get_bucket(bucket_name)
    blob = bucket.blob(source_blob)

    model_data = blob.download_as_bytes()
    model = joblib.load(BytesIO(model_data))
    return model
```

Testing the API for the model

In [ ]:
```
url = "https://us-central1-spring-cloud-econ-446.cloudfunctions.net/stroke_function"
```

In [ ]:
```
r  = requests.post(url,
    json={
        "age":90,
        "gender":1,
        "heart_disease":1,
        "avg_glucose_level":90,
        "bmi":10,
        "smoking_status":4
    }
)
```

In [ ]:
```
print(r.content.decode('utf-8'))
```

b.)

Make a user-friendly input page that takes the inputs to your ML model via widgets and displays the output. Upload a seperate .ipynb that makes this easy to use. (Next Assignment you will have tot urn this into a shareable webpage).

The widget is included in the file `widget.ipynb`.

c.)

Think of a company that would use the ML app you just built. What employees could use this app what would they use it for? Write a short paragraph.

Americans. Our machine learning model is designed to predict the probability of a patient experiencing a stroke based on input parameters such as age, gender, presence of heart disease, BMI, glucose levels, and smoking status. Healthcare providers, including doctors, nurses, and medical researchers, could use this app to identify high-risk patients early on. By leveraging larger datasets, the app can improve the accuracy of predictions, facilitating the development of targeted prevention strategies and personalized treatment plans. Also, insurance companies and public health officials could utilize this model to assess population health risks, allocate resources more effectively, and implement preventive health measures on a broader scale.