Whoop Fitness Tracker Application

Brett Mele

MSDS434, Northwestern University

March 12, 2023

Introduction

The goal of this project was to develop an application to serve machine learning predictions in an application to make inferences about the effect of behaviors like sleep performance and workout strain on daily recovery. To do so, we use the Whoop API to collect daily biometric data captured from a wearable fitness tracker for a specific user, create models to predict the Whoop recovery score and workout strain, and serve those predictions in a front end Rshiny dashboard hosted on Google Cloud Platform (GCP). The dashboard allows the user to interact with his or her own data with individualized model predictions.

Project Overview

All code for this project is stored in a Gtihub repository (link) and is set up using a microservices architecture, with all services hosted on GCP.

The first step in the project is to retrieve data from the Whoop API. We use a custom python package developed by Josh Hjeka for this purpose. This package has helpful wrappers around python http request methods specific to the Whoop API. To ingest the data, calls are made to the api to return data from the recovery, cycles, sleep and workout endpoints. Recovery data includes daily measurements surrounding recovery like Whoop's recovery score and heart rate variability. Sleep data includes total sleep time, time in the various sleep stages, respiratory rate and more. Cycles data includes overall strain metrics for each day. Most of this data is based on heart rate measurements, and includes things like overall strain for the day and resting heart rate. Finally, the workout data includes information about each of the users' workouts. To ingest the data, we create a python service to make a request to the api, store the returned data in temp csv files, and upload those files to cloud storage. A docker image for this service is created with Google Cloud build via a yaml file, and the container is hosted on google cloud run.

With the data hosted in cloud storage, the next step is to make the data accessible in a cloud data warehouse. For this project we use BigQuery as our cloud warehouse solution. A python microservice is used to get blobs from cloud storage and upload the raw data into tables in a BigQuery dataset. This service is run via a container created in the same manner as above.

After data is uploaded to the warehouse certain tables must be merged together in order to prepare the data for modeling. We use python to query the BigQuery tables, clean and apply transformations, merge data and upload to a new table. Transformations largely surround calculating weekly moving averages for each of the different metrics. Merging the data together is necessary to link workouts and previous day's information to the recovery data for each data. Once this is accomplished the service is containerized and deployed via cloud run.

Each of the above services are set up on a schedule with Cloud Scheduler. There are three cron jobs that run each of the microservices each morning sequentially, starting around 12am EST.

For modeling, we chose to predict the next day's recovery score and workout strain. Multiple model types were tested but we settled on ridge regression given the relatively small size of the dataset. Models are created in a jupyter notebook, written to pickle files and uploaded to individual cloud storage buckets. Then, each model is registered in VertexAI and set up as a REST api endpoint.

Finally, the front end dashboard is an RShiny application that displays recovery data for the past month and workout data for the past week along with predicted recovery and workout strain. The data is displayed in tabular format with a widget included to view actual vs predicted recovery trends for a specified time period. Once again, this application is containerized and hosted on GCP via cloud run.

Discussion

All source code is stored in Github, data stored in BigQuery, ML predictions served out with models set up as REST API endpoints, and an application hosted in GCP. As noted above, all code is deployed into a GCP environment. The application allows for ML inference via rest api endpoints and the frontend dashboard.

Unfortunately separate environments are not used given time constraints, but it likely would be trivial to set up development and production environments with how the application is structured. Comprehensive monitoring and alerts are available in GCP thanks to their Cloud Run monitoring interface. Containers can be monitored to see CPU and memory usage, and web traffic through the application can be seen as well

As far as security, I do believe the principle of least security applies. Each of the microservices is managed via a service account that only has access to what it needs for those services. For example, the Shiny application has a dedicated service account setup as a VertexAI agent that has permission to use everything necessary to interact with VertexAI, but nothing more. Additionally, the project uses environment variables to protect passwords and secrets, none of which are publicly accessible in the Github repository.

The project is not set up to include CI/CD. However, it could be done relatively easily via Google Cloud build. In order to do so, a yaml file would need to be created specifying what images to create, and then Cloud Build would need to be set up to link to the Github repository and deploy changes any time a pull request is merged. A trigger with Github actions workflows could also be set up to facilitate this process, similar to the steps outlined here.

Appendix

Github:
https://github.com/melebrett/WhoopApp

App:
https://whoop-app-oc5t7fetca-uc.a.run.app/