Machine Learning Engineer Deployment Exercise Write Up

Zach Zastrow

08/12/2020

Brief Summary

For the deployment exercise, I split the project into three separate Python files. The first file, StateFarmModel.py, preps the data and trains the model based on the given jupyter notebook file. The next file, RestApi.py, handles the API call to predict the business outcome. The RestApi.py file trains the model by calling the functionality from the StateFarmModel.py file when RestApi.py is first run. Once trained, the model will listen for any incoming post requests. The RestApiUnitTests.py file tests the API calls.

Preparation for Production

To prepare for production, I made several changes to the existing code as well as adhered to various standards and practices while developing the API. First, I broke up the data preparation and model training up into several functions. This eliminates repeating code, and will help with making future changes. In terms of code standards, I made a conscious effort to make my variable and function names self-documenting so that other members of the team can better follow what each variable and function does. I also included comments to clarify my reasons for certain blocks of code. Lastly, as evident by the RestApiUnitTest.py file, I included unit tests that can be run at any time.

API Description

For the rest api, I considered multiple libraries to use. It ultimately came down to FastAPI vs. Flask. FastAPI would have been my first choice due to having better performance benchmarks and async support. I ultimately decided on Flask, however, due to the requirement that the API call requires both individual and batch processing, and Flask was better to work with in terms of handling either in the same call. When a request comes in, it will determine the predicted probability based on the training data, and then based on that decimal value, it will determine the predicted outcome based on the 75^{th} percentile cutoff that was mentioned in the jupyter notebook file. The program then alphabetizes the columns and returns the pandas dataframe as a json object in the response, assuming the input was valid.

Docker Image

The Docker image for this project is located at https://hub.docker.com/r/zzastrow1/project. Running the run_api.sh script will build the Docker image from the Dockerfile and create run the application.

Optimization

Going forward, the api call can be optimized for scalability in multiple ways. The first priority would be to implement rate limiting that would throttle the amount of calls coming in if it receives a high amount of traffic. This would prevent the program from crashing and would help it run more reliably during the periods. Next, given that the training dataset is relatively small, using pandas is fine for now, but if a more accurate model with more training data is required, it would be worth considering migrating to pyspark to take advantage of parallelizing the data cleaning and training process.