IDS 594

Special Topics in IDS MLOps (44420)

2021 Spring

Final Project Report

Persistent model pipeline : Design, Implementation and Docker based Deployment

Submitted by :

George Peter

UIN : 659371163

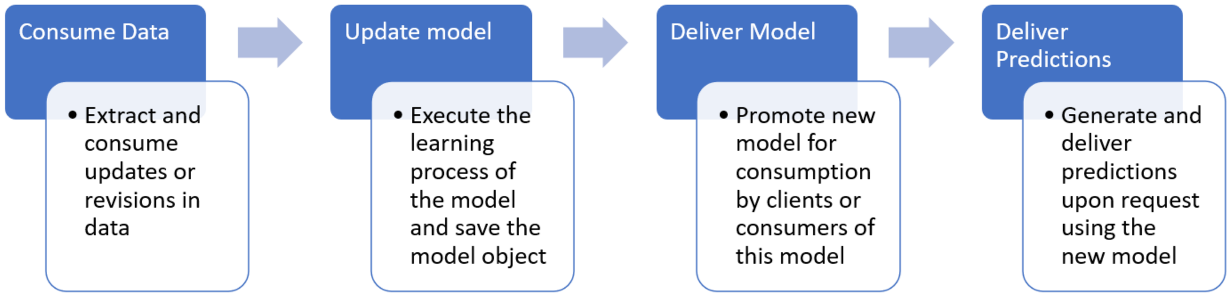
gpeter7@uic.edu

Goals of the Project

This project aims to implement a mechanism to deploy a machine learning model from one environment to another. For example, the lower environment can be considered as development or staging. Testing or production environments could be considered for targets respectively. The mechanism must demonstrate the following characteristics

Scope

Use cases chosen for implementation in this project represents end to end scope items in the life cycle of a machine learning operation. Specifically, the scope areas to be addressed in each execution of the proposed project are as follows


Consume Data
• Extract and consume updates or revisions in data
Update model
• Execute the learning process of the model and save the model object
Deliver Model
• Promote new model for consumption by clients or consumers of this model
Deliver Predictions
• Generate and deliver predictions upon request using the new model

Automation

A manual intervention can create significant dependencies for the process of deployment to move through the steps of the process. Accordingly, it is very important to automate various elements of the operational process designed to address the items in scope of this project. The use case considered for this project considered that updated data is available in regular intervals. Therefore, the operational process designed for this project would need to kick off once the new data / updated data is available. The model should proceed through the steps automatically unless a decision has to be made for addressing exceptional circumstances; such cases would require a manual inversion.

Flexibility

The model would be packaged for deployment in a container aligned to the target platform on cloud. The packaging process will structure the model including all applicable dependencies to one single container. The resulting package would need to be portable to another area (such as staging or production. When deployed in the target environment, specifications of the environment as prescribed at the time of packaging should be installed / configured for the specific container in which the model is expected to run.

Solution Design

A fundamental design choice made in the design is the persistent model of pipeline. This mechanism generates and saves the model in a process mutually exclusive from prediction.

Pros

- Allows the prediction to continue uninterrupted while model is being re-generated
- Scope of work during prediction is limited to read model and apply it
- Better performance given scope is limited to predicting against a few data points

Cons

- Stale model, since the model is static and learned a while ago

Given below are the solution components of this solution

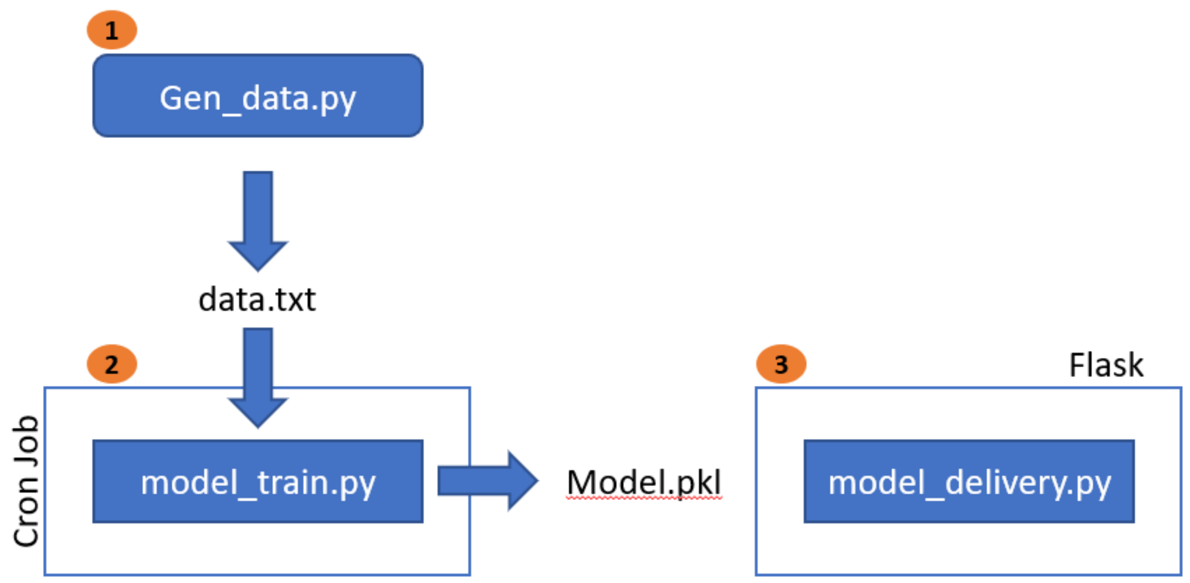
Area	Implementation	Summary of the function	Industry Point of view
Data Gen	Gen_data.py	Generate input. As implemented, a random set of data points are generated.	A feed file generated from data warehouse / ETL process
Model Gen	model_train.py	Consume input data and re-generate the model. The model is implemented as linear regression in this project. Model is saved into a pickle file.	The model itself can be mode complex, for example a neural net. Saving to file process would change to align to the new model, for example the target can be database.
Scheduling	Cron Job	Wakes every 1 min to run model generator step	Time period can vary depending on circumstances
Model Delivery	model_delivery.py	Consumes the saved model and generates a prediction against the data provided. A flask based architecture has been used to deliver the prediction result.	The mechanism of delivery can vary depending on factors such as client (requester) and integration platform specifications
Packaging	Docker Image gen and dockerfile	Create a docker image of the model delivery. Package contains model pickle file, model delivery flask app and all applicable dependencies	A more complex model system can require more items to be included in the docker model
Deployment	Docker Image Deploy	Apply the docker image to docker platform	Actions should be similar

Other Possible Solutions

1. Scheduling : Scheduling process can be implemented in any tool that caters to a job scheduling platform. Examples include Airflow, MLFlow, Luigi etc. Ultimately, the use case should accomplish the regular trigger to actions set in the job scheduling platform, and must be viable for the packaging and deployment process as designed.
2. Packaging and Deployment : Several tools are available as alternatives to docker. As our target platform is AWS, ECS and other similar products can be other options.

Technical Architecture

Diagram below shows the technical architecture of the project.



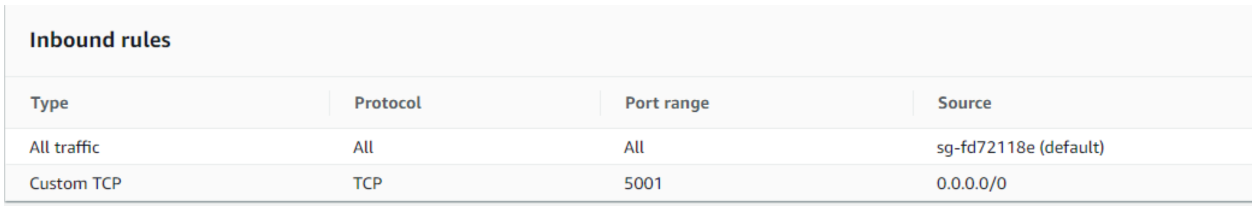
Note : The deployment process of the model is not depicted above. Refer to the Dockerfile section for more details.

Explanation against each section is provided below.

1	Generate the data and save it to file	Create delimited list of X and Y values
2	Run Cron, train new model, Save model	Consume data file and generate the model. Cron runs the model every 1 min and saves the model to pickle file.
3	Deliver prediction via Flask	Load the model from the pickle file and generate predicted y using input value of X. Deliver the result via a flask app.

AWS Network Port Configurations

The Flask application is configured to run over port 5001. Accordingly, AWS platform requires this port to be turned on in network configuration so that a browser can access the application via the internet. Please refer to the screenshot below for the actual configuration.



Inbound rules

Type	Protocol	Port range	Source
All traffic	All	All	sg-fd72118e (default)
Custom TCP	TCP	5001	0.0.0.0/0

Cron Job

Cron is currently configured as noted below

```
*/1 * * * * /home/ubuntu/miniconda3/envs/datasci-38/bin/python
/home/ubuntu/project/model_train.py >> /home/ubuntu/project/logfile.log
```

This configuration allows the job to run a model training every single min, and deliver output into a log file.

Dockerfile

Given below is the content of docker file

```
FROM python:3.8-slim-buster
COPY . .
RUN pip3 install -r requirements.txt
CMD [ "python", "model_delivery.py", "run", "--host=0.0.0.0"]
```

Requirements.txt refers to the dependencies, file content noted below

```
flask
requests
```

Dockerfile Image Creation and Deployment

Image created successfully with command below

```
sudo docker build -t "prediction_service" .
```

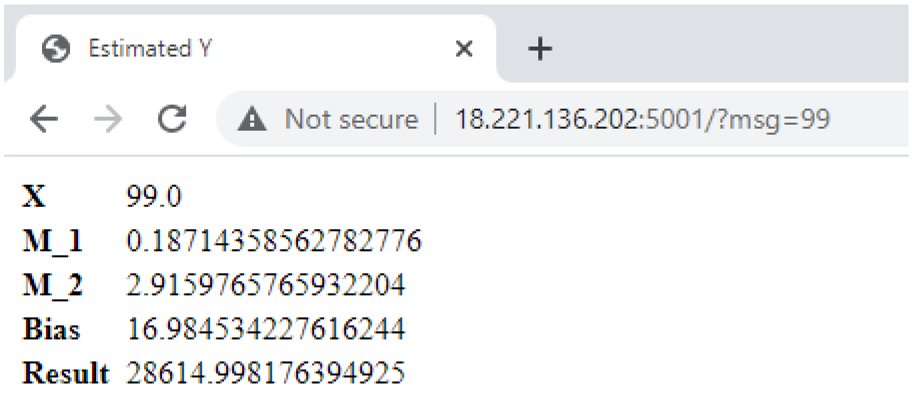
Image Deployed using command below

```
docker run --publish 5001:5001 prediction_service
```

Note : --publish option essentially maps port 5001 of container to port 5001 of the OS. This means when we navigate to http://18.221.136.202:5001/?msg=100, the call is forwarded to the container machine we have set up, which in turn is picked up by our FLASK app listening at that port.

Final Output Screen

It follows the flask app coding noted in project file : model_delivery.py



```
X       99.0
M_1     0.18714358562782776
M_2     2.9159765765932204
Bias    16.984534227616244
Result  28614.998176394925
```

Further scope of improvement

The solution designed as part of this project can be improved further with the following steps

1. Scalability : Alternate models for FLASK platform may improve the scalability of this application and manage performance under load.

2. Testing mechanism : An automated testing process can be devised against the learned model as part of the process. The results from testing can be an indication if the new model should be deployed or not.

3. Better avenues for integration : The results can be delivered on JSON framework information to allow better integration with calling applications and clients.