WORKSHOP DAY ONSITE
OCTOBER 4, 2023

CONFERENCE DAY ONSITE
OCTOBER 5, 2023

WORKSHOP DAY ONLINE
OCTOBER 6, 2023

Fast and Scalable Machine Learning Model Deployment

Vikramjit Sidhu

SUMMIT.DATAMASS.IO

Contents

1. Data Scientist Workflow
1. Deployment Pipeline
1. Challenges

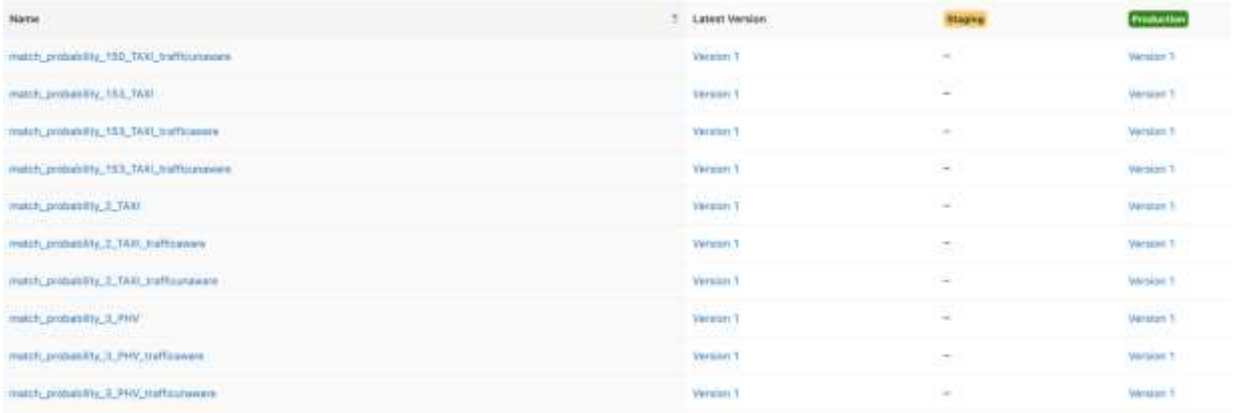
Data Scientist Workflow

Data Scientist Workflow

- Model training and experimentation performed on Databricks

- Metric logging in MLFlow

- Model Logging in MLFlow



Data Scientist Workflow

- Model Schema in MLFlow

Schema

Name	Type
Inputs (5)	
route_decision_score	double
farevalueinminor	double
offer_throughput_rate	double
eventtime	double
backtoback	double
Outputs (1)	
-	Tensor (dtype: float64, shape: [-1])

Data Scientist Workflow

- Smaller ML models work better than a large complex model
- Each city where FreeNow operates usually has a model associated with it for a business use case

Deployment Pipeline

Pipeline Requirements

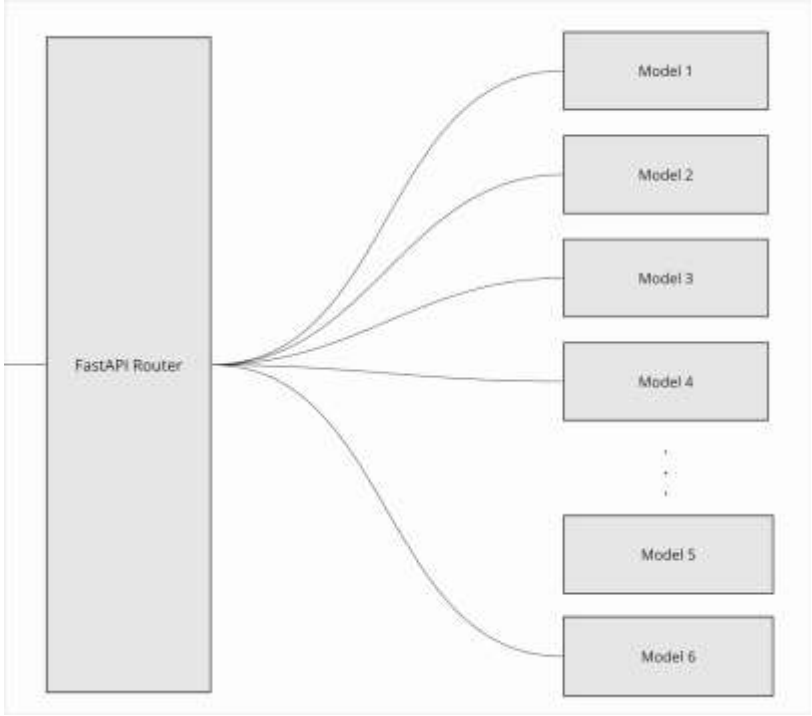
- Easy to use
- Should be able to handle an arbitrary number of models
- Handle A/B tests for the models
- Optimize memory usage of the model
- Enforce data quality by using the model schema

Tools

- We encapsulate our applications within Docker containers
- The Gitlab Continuous Integration (CI) tool builds the Docker images, performs the tests and manages deployments
- We use Kubernetes to orchestrate and manage the container deployments at scale, this also allows us to be agnostic to the cloud platform

Design

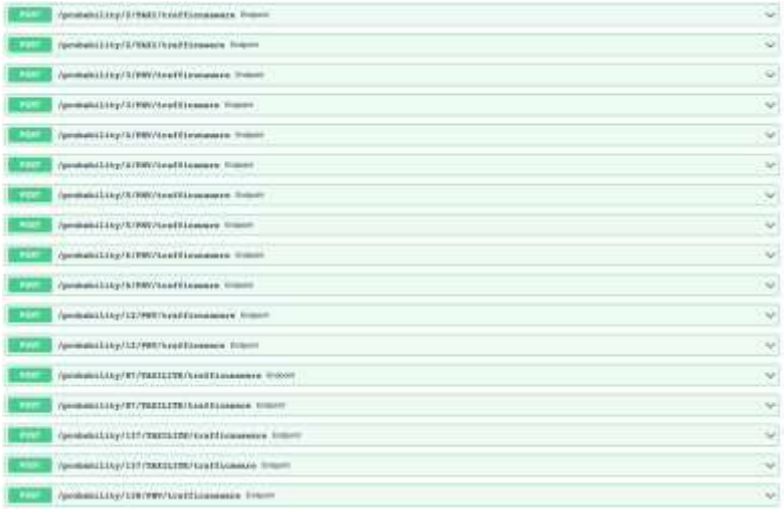
- Each model is deployed within a Docker container as a microservice
- The microservice exposes an endpoint which can be queried for predictions
- The Fast API app runs in a separate Docker container
- It routes incoming traffic to the relevant model



Defining the Models

- The models are defined in a YAML file
- A parameter can be added if the model needs to be A/B tested

```yaml
- name: match_probability_2_TAXI_trafficunaware
  version: 1
  endpoint: probability
  parameters: [2, TAXI, trafficunaware]
```



Gitlab CI Deployment - Parent Pipeline

- The parent pipeline defined in gitlab-ci.yml is first triggered

- The create_model_folders job runs performs the following:
  - Create a folder for each model using a predefined template (**model templating**)
  - Store the created files as Gitlab artifacts so that it can be accessed later in the CI pipeline
  - Trigger a **dynamic child pipeline** which then builds, tests and deploys the models

Model Templating

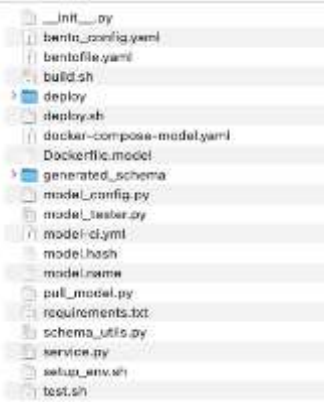
- Template files are created, specific strings are defined within them which are replaced as per the model

```
model_mlflow_name = "MODEL_MLFLOW_NAME_REPLACE_STRING"
model_version = "MODEL_MLFLOW_VERSION_REPLACE_STRING"
model_name = "MODEL_NAME_REPLACE_STRING"
model_route = "MODEL_ROUTE_REPLACE_STRING"
batch_it = "MODEL_BATCH_IT_REPLACE_STRING"
```

→

```
model_mlflow_name = "match_probability_2_taxi_trafficaware"
model_version = "1"
model_name = "match_probability_2_taxi_trafficaware"
model_route = "probability/2/TAXI/trafficaware"
batch_it = "False"
```

- For each model we have the final folder structure as seen in the image:



- The final model folder is stored as an artifact in Gitlab

Gitlab CI Deployment - Dynamic Child Pipeline

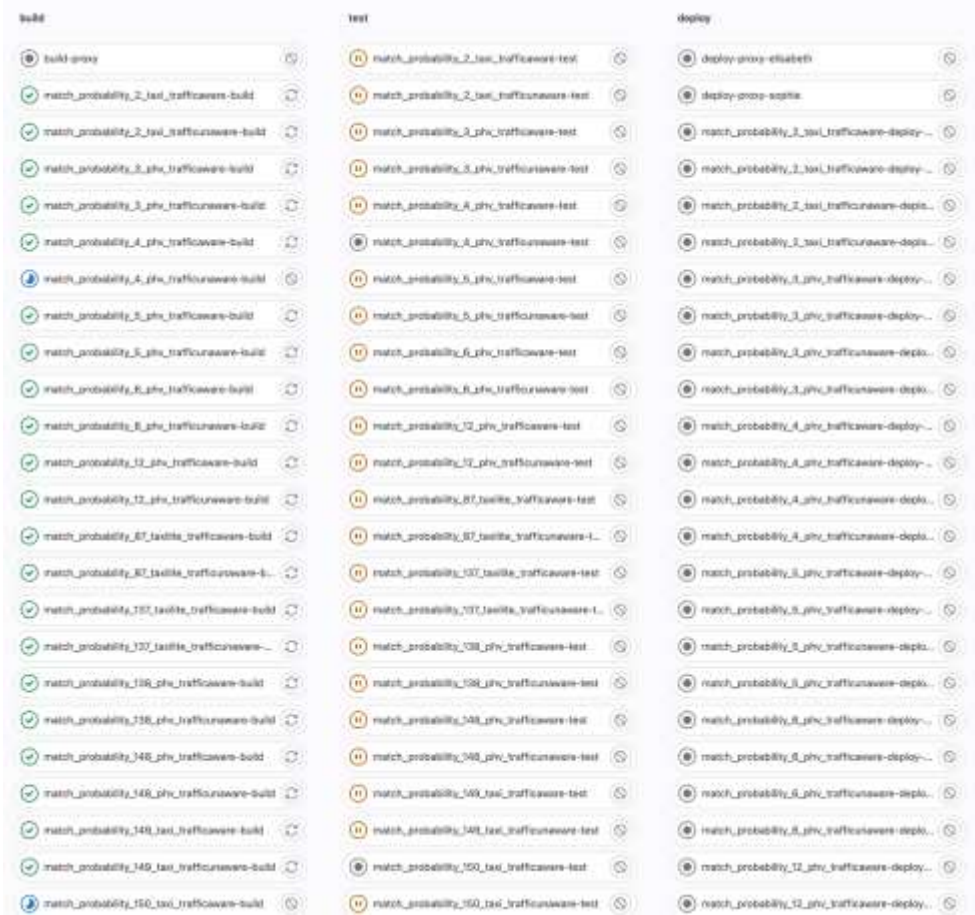
- Child pipelines are a feature in the Gitlab CI, they are CI pipelines triggered by the default (parent) CI pipeline
- We have a dynamic child pipeline which is created during runtime using a template

```
MODEL_NAME_REPLACE_STRING-build:
  stage: build
  needs:
    - pipeline: $PARENT_PIPELINE_ID
      job: create-model-folders
      artifacts: true
  variables:
    MODEL_FOLDER: MODEL_FOLDER_REPLACE_STRING
  script:
    - cd $MODEL_FOLDER
    - docker-compose -f docker-compose-model.yaml up --build model-builder
  artifacts:
    untracked: false
    when: on_success
    expire_in: "2 hrs"
    paths:
      - $MODEL_FOLDER/generated_schema/model_schema.py

MODEL_NAME_REPLACE_STRING-test:
  stage: test
  needs:
    - pipeline: $PARENT_PIPELINE_ID
      job: create-model-folders
      artifacts: true
    - job: MODEL_NAME_REPLACE_STRING-build
      artifacts: true
  variables:
    MODEL_FOLDER: MODEL_FOLDER_REPLACE_STRING
  script:
    - cd $MODEL_FOLDER
    - docker-compose -f docker-compose-model.yaml up --build model-tester
  artifacts:
    untracked: false
    when: on_success
    expire_in: "2 hrs"
    paths:
      - $MODEL_FOLDER/deploy/*
```

Dynamic Child Pipeline

- The dynamic child pipeline is responsible for building, testing and deploying the machine learning models



Model Building via BentoML

- This step builds a docker container which will be used to serve the model via an endpoint
- BentoML is an open-source tool which is used to build the container for the model
- BentoML Features
  - Can read models from a variety of sources (e.g. MLFlow) and deploy to various targets (Docker, Kubernetes)
  - Provides Adaptive Batching to improve the model performance

Model Test and Deploy

- We spin up the built docker container with the model endpoint and send requests to it with randomly generated values
- We measure the memory usage of the model during this process
- The measured memory usage is then updated in the Kubernetes file used to deploy the model

Challenges and Limitations

Redeployments

- The CI/CD pipeline relies heavily on storing artifacts in Gitlab
- These artifacts are sometimes deleted by Gitlab or they expire
- Due to this, the whole pipeline must be run again

Incompatible Libraries

- Because the libraries we are using have an extremely fast development time, it can cause incompatibilities between some of them
- This causes deployments to break and can take some time to debug

Thanks! Questions?