I would like to thank Olivier, Dominique and the monetization team for proving me this opportunity. It was a roller-coaster ride for me to complete this challenge in the last 4 days and I learned so much!

Pipeline description and design choices

Attached a README.pdf with a detailed pipeline description and design choices. Please review the same, also, the README is available in the package as markdown.

Performance evaluation of the model

Attached the training and validation logs in a csv format.

Results:

- train-logloss:0.57310
- train-auc:0.79693
- validation-logloss:0.57595
- validation-auc:0.72821
- actual-target-mean:0.01223
- predicted-target-mean: 0.4071 (Extremely high bias, need to fix this later)

Setup:

- training: 80% validation 20%
- 500 rounds of training at 0.05 eta on GPU
- Only experimented max_delta_step. Scale_pos_weight helped eliminate the class imbalance. Ideally, I would have built a shallow NNet with fastai (categorical variables turned to embeddings) and trained with a custom log-loss where I would use the same imbalance ratio. I think LR schedules and increased latent space can improve the AUC further.
- I also experimented with RF (with class balance), and BalancedRadomForest from imblearn, but found the XGBoost results best. Also, the xgboost model serializes easily and is 10x smaller and faster (with GPU inference), so I used XGBoost.

Scaling

I chose to learn kubernetes (and associated tools) from scratch keeping scalability and downstream tasks in mind. *My solution works well and I tested it by scaling pods* to see if it can scale properly in the real situation, I have provided a better explanation around it's scaling capabilities and how it can be tested later in the document. I will need some expert guidance to adjust the setup to billions of data-points *but the capability* *itself is provided with the current implementation*. Please pardon me if I'm missing something, and feel free to educate me.

Future work

(some parts are repeated in the README file)

- For model/file versioning and experiments tracking, I can include MLFlow in individual components and integrate experimentation with kubeflow pipeline and seldon deployment.
- For monitoring, scaling, and logging, I can design a streaming service that pulls 5K records per sec from the test data and requests our deployment to generate predictions. We can ustilize all the analytics using the seldon-analytics (Prometheus + Grafana). *Once this is implemented, we can test the scaling capacilities of our solution.*
- For deployment testing, once the streaming simulation is in place, we can easily change our deployment yamls to test different model deployment strategies like cananry, blue-green, shadow, and also A/B/N test our models. Seldon natiely supports Kafka, we can also write our consumers on topics that produced this data.
- Seldon also offers explainer and feedback, which can help us improve models (via features) and create a reinforcement loop to update models on-the-fly based on rewards. This can be easily connected with our kubeflow pipeline or Github CI.
- Current setup required offline notebook experimentation and training to finalize the components. These can become a part of the training pipeline with databricks (directly) or designing an internal notebook based framework like fastai did.
- On the modeling side, there is a high prediction bias that needs to be fixed.
- MLflow setup can be used in conjunction with the RapidsAI cuDF support for better training on GPUs. XGBoost natively works well on GPUs but Rapids can help improve by a good margin.

Source code is provided in the zip file.

Additional notes

- The training and experimentation notebooks are not included in the submission (for framework brevity) and all the distilled information is transfered into the components. Please let me know in case it is needed.
- The pipeline is designed to fullfil the exercise requirements, at several points, I had to manually connect to online hosting/resources and add environment variables on ad-hoc basis. I have included samples in the config folder but they aren't perfectly integrated with the pipeline.
- My approach to emulate kubernetes clusters using minikube may not be the best, in the given time constraint, I felt it was the most convinient to use amongst many.
- I am not providing a `run_setup.sh` (which I ideally should) because the dependencies span outside simple pip and system installs, and I have marked the same in the requirements file. For this reason, I'll try my best to write a setup process that should help you reproduce the exact results, and your

expertise would help my case. In case the setup isn't working, I have also attached the proofs of a working setup in the README file.

Setup details

- We need to first install docker, minikube, kubeflow and helm. Assuming an ubuntu-based system, here are the steps.
    - Docker: https://docs.docker.com/engine/install/ubuntu/
    - minikube: https://minikube.sigs.k8s.io/docs/start/
    - kubeflow: https://www.kubeflow.org/docs/started/workstation/minikube-linux/ # Do run kfctl apply -V -f ${CONFIG_URI}
    - helm: https://helm.sh/docs/intro/install/
- Once the above setup is done and you can see the kubeflow deployments working in minikube dashboard, follow the below steps.
    - Ingress with Istio
        - Create namespace `kubectl create namespace seldon`
        - run in terminal `helm install seldon-core-operator seldon-core --set istio.enabled=true --repo https://storage.googleapis.com/seldon-charts --set usageMetrics.enabled=true --namespace seldon`
    - Configure Istio Ingress
        - Using the seldon-gateway.yaml file in the config folder, run `kubectl create -f config/seldon-gateway.yaml`
        - Label our namespace so that istio can sidecar it, run `kubectl label namespace seldon istio-injection=enabled`
        - Run in terminal `kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}'`
        - IN A SEPERATE TERMINAL, Run `kubectl port-forward $(kubectl get pods -l istio=ingressgateway -n istio-system -o jsonpath='{.items[0].metadata.name}') -n istio-system 8003:80`
    - Deploy our model server
        - Using the serve.yaml file in the deploy folder, run `kubectl apply -f serve.yaml`
    - Test our server
        - Open ipthon or notebook and run the following
            - `from seldon_core.seldon_client import SeldonClient`
            - `sc = SeldonClient(deployment_name="xgboost",namespace="seldon")`
            - prepare a data row as numpy ndarray in shape (1,56) and name is (let's say D)
            - `sc.predict(gateway="ambassador",transport="rest",shape=(1,56), data=D)`
            - You should see the model output from the deployemt.