riskified

Data Science Home Assignment

Task:

Riskified is an AI platform powering the eCommerce revolution. We work with the world's largest brands - from airlines to luxury fashion houses to gift card marketplaces. We have a wide range of data points and models working to analyze the orders of these merchants and make a decision on whether an order should be approved or declined. From time to time we receive alerts from the Operations team saying that there is a deterioration in the performance of the models, and it is our responsibility to understand what is causing the problem.

In the dataset attached, you will find a set of orders from some of our merchants, representing the population of one of our models. We received an alert from the Operations team saying that there seems to be a problem with the performance of this model. A short preliminary analysis raises the suspicion that the problem is a result of a population drift that happened in at least one of the features that we have in the dataset. When and in which features it happened isn't clear yet.

Task 1

Explore the data and identify when and which features show some kind of population drift. Add plots to support your findings.

Task 2

Our goal is to automate the alerting process, so we are not dependent on manual analysis of the Operations team in future similar events. Create a model that will allow us to detect future feature population drifts; this should be a POC. Try to be as general as possible, as drifts might also appear in different features, times, and intensities. Also, take into account that we would like this algorithm to detect the drift as close as possible to its inception.

Use the test set to validate your model and report when and which features are involved in the drift.

Task 3

Now is the time to move to production. Design the implementation of the model in production; including the end-to-end flow and which technology you will use in each step and why. The deliverable of this task should be a flow chart showing the design and a text file that describes the entire flow and each step in more depth.

Your solution should meet the following criteria:

1. It can work at scale.
2. Ability to know when things don't work well and to easily debug and understand why they went wrong.
3. Ability to control how many alerts will be shown to the Operations team - we can't show them thousands of alerts.
4. In the future we would like to improve the algorithm - your design should persist the important data that will enable us to design or train a new algorithm.
5. Ability to easily revert if we introduce a new change and things fail.

Assumptions**:**

1. An order (with its data) is being submitted to us through an API by a merchant.
2. We have dedicated microservices that calculate the features values and publish the results to a "features" topic which later is consumed to create classifications by our chargeback models (which give a score between 0-1 and a decision).
3. The model classifications, feature values, and the order's details are being consumed by stream processing and saved in S3 bucket.
4. The data in S3 is being ETLed on a scheduled basis to a data warehouse in a relational form.
5. There are over 100M orders per day generated by thousands of merchants.
6. We have dozens of models serving in production. Models and merchants have many to many relationships (one chargeback model can serve multiple merchants, one order can be classified by multiple models).
7. The alerts that your algorithm generates will be consumed by analysts in the Operations team and will be manually reviewed by them through a web UI - you shouldn't care much about the web UI.

Good luck :)

Please keep in mind that all the data was synthetically generated and does not represent a real-world scenario. Our goal is to see your full work process and approach to the problem. The accuracy of the model is less important.

<u>Variables</u>

- *order_id = unique identifier of the transaction (**not** chronological)*
- *merchant (categorical) = merchant name*
- *created_at (date) = date of transaction*
- *score (float) = the machine learning model score*

Features used by the model:

- *price (float) = amount paid at checkout*
- *ip_score (float) = risk score of the IP used by the customer, received from an external data source*
- *external_email_score (float) = risk score of the email used by the customer, received from an external data source*
- *address_score (float) = risk score of the address used by the customer, received from an external data source*
- *bill_ship_distance (float) = the distance between the billing and shipping addresses*
- *avs_match (categorical) = a matching level between the billing address (which was inserted by the person making the transaction) and the payment's address information*
- *payment_method (categorical) = payment method used by the customer*
- *history_count (integer) = number of orders previously made by the customer*