TAZI Software Engineer (Backend Developer) Interview Project

Project Description

In this project, you'll simulate a continuously growing data source and do some windowed calculations on the fly in a single application.

Part 1: Continuously Growing Data Source

You'll be given a CSV file, containing probability values of predictions from multiple machine learning models, which are trying solve a classification problem with two labels: A and B.

CSV file is **NOT** the datasource, its only purpose is to have something fixed when populating the simulated continuous data source.

csv schema:

id	given_label	model1_A	model1_B	model2_A	model2_B	...
1	A	0.3	0.7	0.2	0.8	...
2	B	0.21	0.79	0.1	0.9	...
3	B	0.0	1.0	0.25	0.75	...

id: id of the *instance*

given_label: actual label of the *instance* (real class)

modelX_Y: modelX's estimation of the probability that the instance is of the **class Y**

You may use any DB (relational, object) you prefer as the datasource: PostgreSQL, MongoDB, SQLite etc... Note: please **DO NOT** use Kafka or any other event streaming platform.

You're expected to populate the datasource **gradually** (500 instances per second for example) from the given CSV file to simulate data **growing** in time.

Calculations in Part 2 will use this growing data.

Part 2: Calculating Confusion Matrix

In a classification problem, there are multiple labels representing the *class* of the *instance*. For each *instance*, the system is trying to predict the label.

In this part, you'll be calculating **confusion matrices** and writing them back to the data storage.

https://en.wikipedia.org/wiki/Confusion_matrix

Here we have labels like **A** or **B** but in real world scenarios labels would be such as ["survived", "not survived"] or ["will divorce" "won't divorce"] or ["fake", "real"] or ["cat", "dog", "frog", "human"] etc...

Confusion matrix example: (Act: Actual, Prd: Predicted)

Act/Prd	A	B
A	124	12
B	11	321

(Above can be read as, in 124 cases actual label was A and the predicted label was A, whereas in 12 cases actual label vas 12 but predicted label was B, etc.)

To get *predictedLabel*, first you need to combine results from all models, using weighted average.

model weights are below:

model	weight
model1	0.5
model2	0.6
model3	0.7

You'll pick the label with the highest probability, that will be the predictedLabel.

You're expected to calculate confusion matrices with **sliding windows** of 1000

instances. (i.e. if data source contains 2000 instances, you'll calculate 1001 confusion matrices).

- from instance 1 to instance 1000
- from instance 2 to instance 1001
- ...
- from instance 1001 to instance 2000

Confusion matrices should be calculated whenever there are enough instances in the growing data source. Results should be written back to the data source continuously.

Important!

- You should **NOT** rely on your assumptions about the speed or size of the data source. In a real world scenario, your app may run after the source is heavily populated, fetching data may be slow, data source may grow irregularly etc... So, the app should run `as decoupled as possible` from the data source.
- You should **NOT** store all data in disk, trying to avoid using memory. In real life scenarios, data may be **huge**. (Simulated growing datasource will eventually store all the CSV in it, that's OK)
- You should **NOT** store *unnecessary* data in memory, as the original data source may be an **infinite** stream. (You should **NOT** load all the data into the memory at any stage.)
- You'll be continuously populating a data source (from CSV) and reading from the data source at the same time. Please use separate threads for populating data source and doing the calculations, code is expected to be concurrent.
- You should try to optimize reads/writes...
- You should **NOT** depend on some `priviledged user` or any other software that can access datasource's internals. Program should be able to work with minimal requirements (read-only access to datasource)

Other Notes

- You may use Scala, Java, C# or any other programming language.
- Please use a VCS (preferably `git`)
- You should write unit tests (just for the crucial parts)
- Please carefully choose suitable data structures.