**Julius Bär Challenge**

Team Jawab

**1- Overview of the solution**

To predict the labels of each client, we decided to go with a 2-step approach: flagging and inference with LLM. This approach allows to pre-filter the obvious rejections from the dataset and then pass the more complicated ones into the second phase. Most of the flags used had no false negatives, this way we would have hard boundaries on features that are eliminatory.

**2- Flags**

We started with simple filters targeting obvious inconsistencies — for example, mismatched names across documents. To our surprise, this alone gave us a perfect rule: every client with such a mismatch was rejected. Encouraged by this, we explored other human-logical flags, including missing data (e.g., an empty passport number). This led to an iterative process of testing various inconsistency and incompleteness rules — keeping those that consistently matched only rejected clients. Therefore, each flag can be categorized as either searching for inconsistency or searching for incompleteness. By inconsistency, we mean that there is different information concerning the same phenomenon in different parts of the data. E.g., one example of inconsistency can be considered having different names in the passport and account form. By incompleteness, we mean that there is a missing value at the place we expect the information to be present. E.g., imagine if the passport number is missing. That way we a closer to replicating the human logic which evaluates the client information and can find such missing parts or inconsistent parts which will result in a Reject decision. Below are the examples of some such filters we applied (all the filters can be found in the file called final\_pipeline.ipynb, and all the queries perfectly filter out only rejected clients).

* **Address mismatch** between client\_profile and account\_form
* **Any missing field** in the passport section (i.e., empty string)
* **Incomplete inheritance details** (e.g., missing relationship or year) when the client has a non-zero inheritance value

The main advantage of this approach is that it gives us a feature that works reliably on the data we have, regardless of how we split it into train and test. Since the rules only target clients who were rejected, they’re guaranteed to hold across any subset. Instead of building a complex ML model to approximate this behavior, we’re able to capture it with a simple, perfectly accurate, rule-based system. It’s also fully explainable, unlike most machine learning models, which tend to be much harder (if not impossible) to interpret. After applying all our rules, we were able to correctly classify around 4,000 out of 5,000 rejected clients, without misclassifying a single accepted one.

**3- Inference**

For the inference approach after filtering the dataset with flags, we use LLMs for reasoning inference and inconsistency detection. By using prompt engineering and testing different models (qwen2.5:0.5b llama3.2:1b phi:latest deepseek-r1:1.5b deepseek-r1:8b llama3.2:latest

Gemma3:latest in zero-shot learning approach).

This approach was limited by two factors: Computing power (loading larger model is too expensive on single GPU), and processing time (larger model needing more time per client). Given the 30 min window to process the 1000 client validation set, we aimed for a maximum of 20 min of LLMs inference time (Approx 12 sec per client in a worst case scenario).

Comparing an extraction based approach (prompting for extraction) vs a reasoning based approach (prompting for a boolean), reasoning demonstrated overall better performance.

Additionnaly, a correlation between context string length and probability of returning false was observed, among other observation.

Performance of each LLMs was evaluated through the FPR, as the processing pipeline is sequential, to minimize misclassification accumulation.

The best performing mode in terms of time/performance was phi light, inferred on family background and client\_work\_history.

**4- Unsuccessful/Incomplete attempts**

After filtering the clients using rule-based approach, we were still left with the clients that were not filtere out (with approx. distribution 5000 accepted and 1000 rejected clients). We also tried to apply the tree-based gradient boosing model on this remaining set of clients to run a classififcation taks for a reject/accept decision (we tried Catboost since it especially good for handling categorical variables, and we have a lot of them). Some of the features we used are:

* Numerical:
  + Current salary, total\_aum, total property value, number of jobs in employment history, savings value, ratio of a property value to the cash value (inheritance +savings), ratio of inheritance value to cash value (inheritance +savings), total work experience (# of years from the start of the career till the finish of the career), number of effective work years (how many actual years did the client work), saving\_per\_annum (savings value divided by effective work years), salary to max salary ratio
* Categorical:
  + Gender, country code, country of domicile, nationality, marital status, higher education, investment risk profile, investment horizon, investment experience, type of mandate, currency, preferred markets

After running the model we were able to slightly improve the classification accuracy compared to the naive model where accept everyone: ~87% accuracy of the catboost compared to 84% accuracy of naive model evaluated on the standalone test set. In order to select the optimal hyperparameters we ran the Grid search on the hyperparamers space using 5-fold cross validation procedure. The optimal set of parameters we arrived at can be seen below, as well as a confusion matrix on the test set for catboost model:



It has a slight improvement that can still be considered to be added on top of previously described implemented rule-based and LLM procedures. We didn’t have time to add it to the pipeline, but one of the possible improvements of the approach can be seen as adding this type of model on top.

We also tried an unsupervised approach with isolation forest since we knew the contamination rate, ie, number of clients to reject from the remaining dataset post-filtering. The accuracy was very poor. Accepting all clients would have yielded a higher accuracy due to the low proportion of clients to reject in the training dataset.

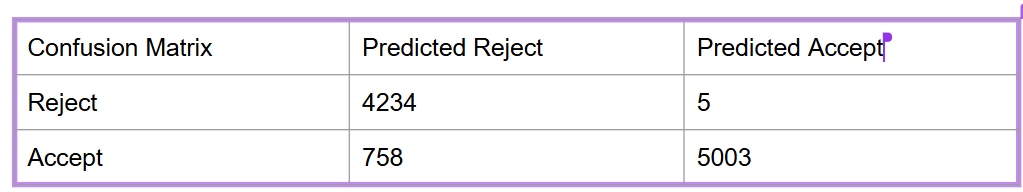
We also tried training with a StackingClassifier on the remaining clients post-filtering via flags. The StackingClassifier used as base estimators:

* Random Forest Classifier
* Gradient Boosting Classifier
* AdaBoost Classifier
* LinearSVC
* Perceptron

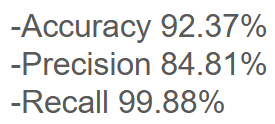
Logistic regression was used as the final estimator. Some hyperparameter tuning was done with randomized search cross validation with stratification. The results were an improvement compared to isolation forest but still did not outperform simply accepting all remaining clients post-filtering.

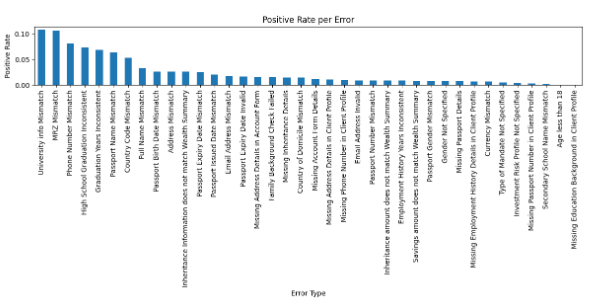
**5- Results**

In the end we were able to arrive at the following confusion matrix on all the datapoints:

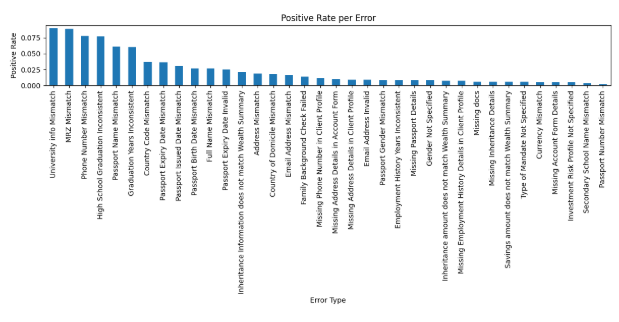


So the achieved accuracy on the provided data is 92.37%.





Training anomaly rates



Validation anomaly rates

Match shows validation of the pipeline on test set.