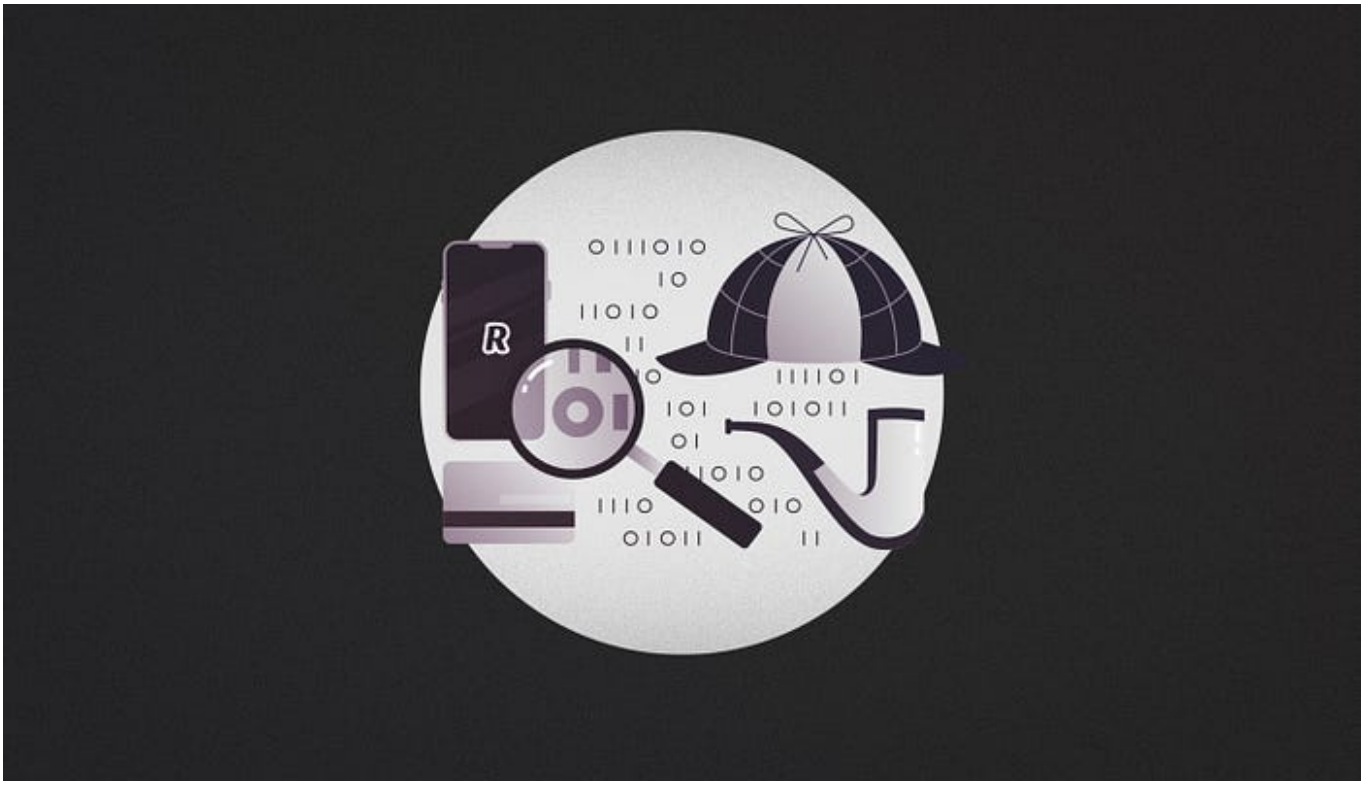
Building a state-of-the-art card fraud detection system in 9 months

https://medium.com/revolut/building-a-state-of-the-art-card-fraud-detection-system-in-9-months-96463d7f652d

Go deeper into our methodology for solving a business problem with machine learning



Have you ever experienced the dismay of getting a call from your bank's security team? The voice on the other end of the line reports that they detected a suspicious transaction.

> "Please confirm that it's you making a payment for $149.99."

You start nervously recalling your latest purchases:

> "Am I losing track of the things I buy, or is someone really trying to rob me?"

Your palms are sweating. It's not you, so you ask your bank to cancel the payment. For the next few hours you're overwhelmed with uneasy thoughts:

> “How did the perpetrator get my payment data? Is it going to happen again? What did they buy?”

It’s irritating, to say the least, and I’ve been there myself. That’s why when I received an offer to build a fraud prevention solution at Revolut, I was happy to embrace the opportunity. What’s more, I was truly excited about solving this business problem using supervised machine learning methods.

In this article, I’ll tell you:

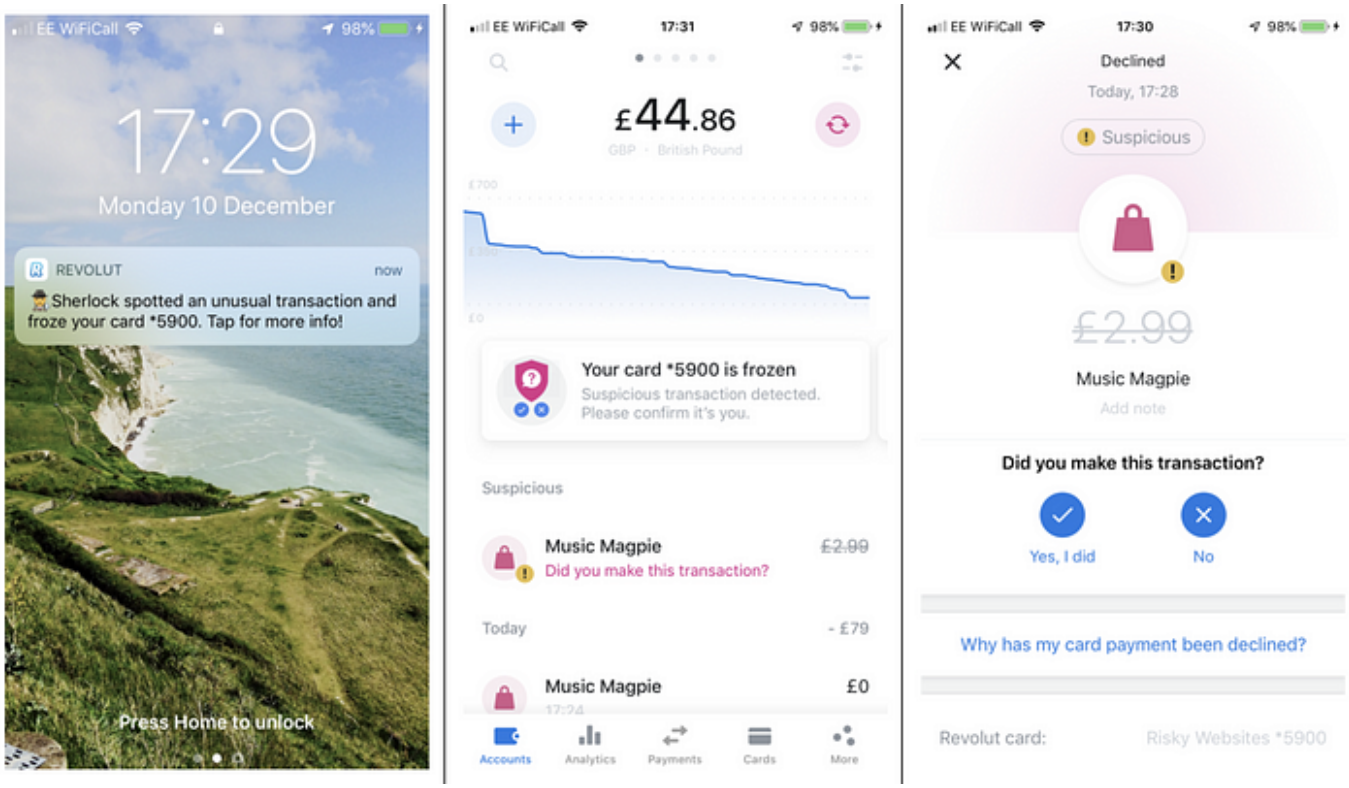
- about the fraud prevention system we’ve created,
- the technologies we used,
- the results we’ve achieved by now,
- and where we’re heading.

I’ll lead you through all the stages of the system creation, from definition to deployment. But first things first.

Meet Sherlock

Sherlock is our card fraud prevention system based on machine learning. It continuously and autonomously monitors Revolut users’ transactions. While a web store or a terminal at Starbucks is displaying the “Processing Payment” animation, in under 50 ms Sherlock evaluates your transaction.

If Sherlock finds it suspicious, it blocks the purchase and freezes your card. A follow-up push notification prompts you to confirm if it was a fraudulent payment, or not. If you respond that it was legit, the card is unblocked, and you simply repeat the purchase. If, however, you don’t recognise the transaction, your card gets terminated, and you can order a free replacement.



Sherlock in action

Fraudsters exploit advanced technologies, they learn fast, but so does Sherlock. Every night, Sherlock's machine learning models are retrained to account for any missed fraudulent and incorrectly declined transactions.

Sherlock in numbers:

- over $3M saved during the year in production
- just 1c out of $100 is lost due to fraud
- 96% of fraudulent transactions are caught
- 30% of Sherlock's fraud predictions turn out to be correct

What these numbers mean to our clients is the crucial difference between an unforgettable holiday and a holiday marred by being robbed and having to make do in a foreign country.

Defining project goals, metrics, and talent

Above all, we aimed to *minimise Revolut fraud losses* and the losses caused by incorrectly blocked transactions. Banks have dedicated departments for transaction analysis with hundreds of people calling their clients in order to prevent unauthorised transactions.

It really takes a huge effort to reduce the damage caused by scammers. We wanted to find a way to detect and prevent fraudulent card transactions efficiently with fewer resources. We needed to learn to predict which transactions a user may demand to chargeback. While diving deeper into the problem we realised the importance of *catching the first instance in a sequence of fraudulent transactions*.

Basically, we're solving a binary classification problem here, i.e. identifying whether a given transaction is fraudulent or not. Thus, *precision*, *recall, and the number of false-positive detections* are our primary metrics. Adding metrics for detecting the first fraud in a row helped us tweak the machine learning performance during its training.

In the early days of our research, we needed an expert to perform *data wrangling, data analysis, data visualisation, and machine learning*. I took on the responsibilities of a data scientist and a backend engineer to build Sherlock, and later I used the help of my colleagues — mobile developers and platform engineers — to integrate it into the Revolut product and develop UI.

Discovering. Data sources

Having the right data is much more important than choosing a machine learning algorithm. At Revolut, we've had a table of transactions reported fraudulent since the early days. It was stored in a PostgreSQL database and was not suitable for doing the analysis.

To simplify the data processing and data visualisation, I set up a nightly delta dump from PostgreSQL into BigQuery — a warehouse database running on Google Cloud. It became easier to join multiple tables and perform an initial analysis of millions of transactions happening daily. Besides, the serverless nature of BigQuery relieved me from the need to maintain the infrastructure. My focus was on the ETL code only.

Design. Envisioning the solution

Having the right data in hand, we moved on to envisioning the end solution. It was important to decide on the architecture at such an early stage of the project to make sure that our solution would fit the pace and conditions of real-time processing.

For the task of detecting fraudulent card transactions, we decided to implement the lambda architecture. *We designed a real-time transaction scoring system paired with a nightly batch data processing pipeline*. The nightly job is aimed at fixing any inaccuracies that might have appeared throughout the day in user profiles.

We also wanted to *increase the accuracy of data scoring and automate the fraud-prevention flow entirely*. Therefore we decided to collect users' feedback on declined suspicious transactions within the mobile app. Besides, this logic would empower our users to take control over their funds and make their own decision — they can either confirm a transaction and unblock their card, or terminate their card quickly and order a new one.

Development

Choosing the language

We've chosen Python both for development and production as it's the most widely used language in the data science community.

Feature engineering

This is the most challenging and creative part of any machine learning project. At this stage, we investigated how we could use what we knew about customer behaviour and merchants to identify fraud patterns in real-time; what kinds of deviations should be considered risky.

Features are attributes that data scientists have to come up with based on the available data. Features are combined into a vector to serve as an input to a machine learning algorithm.

We categorised our features as follows:

1. **Features with no or minimum pre-processing:**
    - Merchant's name, city, category, currency, etc.
    - USD amount of the transaction
    - Transaction time of day
2. **User-focused features that compare the values of the given transaction against the historical transaction data for the given user:**
    - How quickly does the user make the transaction?
    - How much does the USD transaction amount differ from the average spending of this user with the given merchant and the category code, using this payment method, and at this time of day?
    - Is it the first transaction of the user with this merchant? If yes, this merchant is monitored for the next few hours
3. ***Merchant-focused features*** **that compare the current transaction with all the previous non-fraudulent and fraudulent transactions at this merchant or merchant**

category:

- How many users transacted with this merchant before? How many of them reported fraud with this merchant?
- How much and how often do users transact with this merchant?
- How long ago have we seen this merchant for the first time at Revolut?

Getting to scale

During the early stages of my research, I used to experiment with feature engineering in the Jupyter Notebook. But on a scale of millions of users, such an approach wouldn't work.

Besides, I had to construct our training data in a way which would prevent data leakage. In other words, the features for every transaction should only be based on the previous transactions of a user when sorted chronologically. That way, we precisely emulate the production flow.

I turned to the Apache Beam framework using Google Cloud Dataflow as the runner. Beam allowed me to focus only on the logic of feature engineering for an individual user, leaving out the considerations about the parallelisation of the code and hardware infrastructure.

Choosing a machine learning algorithm

I have experimented with a wide range of machine learning algorithms: linear regression using Vowpal Wabbit, gradient boosting using Catboost and XGBoost, scikit implementations of random forest, SVM with linear and RBF kernels, one-class SVM and neural networks using Tensorflow.

In the end, we chose Catboost, an open-source library from Yandex implementing gradient boosting on decision trees, as it outperformed other algorithms on several metrics and the inference speed. This algorithm has proved itself robust on heterogeneous data, data which comes from different sources, containing both numerical features of varying nature and categorical features. Besides, it didn't require much hyperparameter tuning.

How I dealt with an imbalanced dataset

Training a machine learning model on a highly imbalanced dataset requires a particular approach. The fraud rate in the training data is around 0.03%. At first, I reduced the number of non-fraudulent transactions raising the fraud ratio to about 10%. Then, I

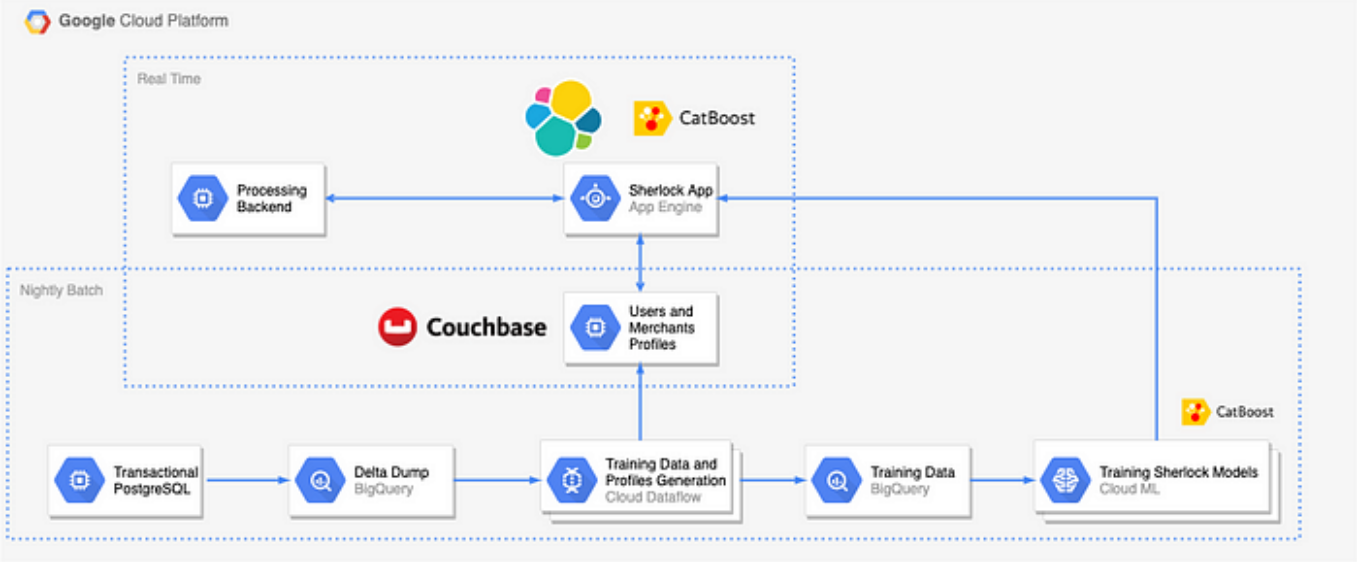
assigned weights to individual fraudulent transactions based on the amount of the transaction and several other factors.

I constructed the validation data using all transactions that happened chronologically for five weeks in the future. It's essential to keep the fraud ratio in the validation dataset the same as in production. That way, we obtain the performance results as close as possible to the ones we can expect in production during live scoring.

Deployment. Our current production flow

Finally, in production, we had to take into account that the card fraud prevention system is *a mission-critical application*. Therefore we had to implement specific measures to ensure resilience and reliability of the service, reduce the response latency down to 50 ms, and make sure that all the important events were monitored and covered by alerts.

Let's take a look into our current production flow



Production flow

It all starts with an *offline nightly job* that is orchestrated by the Google Cloud Composer, a managed service for Apache Airflow. The job dumps the delta of transactions for the day from our PostgreSQL database into BigQuery. Given that the schema of some tables might have changed, I'm modifying the BigQuery schemas on the fly and dumping the data there.

After that, an Apache Beam job running on Google Cloud Dataflow generates the training data that gets dumped back into BigQuery. At the same time, several Beam jobs create users and merchants profiles that are put into Couchbase — the in-memory NoSQL database.

The training data is then used by several machine learning jobs training Catboost models on Google Cloud AI platform. The trained models are stored in Google Cloud Storage.

What happens *in real-time*

When you make a card transaction, the Revolut's processing backend sends it to the Sherlock Flask app deployed on Google Cloud App Engine. The trained machine learning models are pre-loaded into memory.

Upon receiving a transaction via an HTTP POST request, the Sherlock app fetches the corresponding user's and merchant's profiles from Couchbase. Then, it generates a feature vector — using the same features as the ones created in the Apache Beam job that produces the training data — and makes a prediction. The prediction is then sent in a JSON response to the processing backend where a corresponding action is taken — *all within 50 ms*.

In the end

If the probability of fraud is below a certain threshold, the processing backend lets the transaction go through, and the user sees a "Payment Approved" message on a terminal or a website.

If the probability of fraud is above the threshold, the processing backend declines the transaction and sends a push notification to the app.

Performance monitoring

We use Google Cloud Stackdriver and Kibana dashboards to monitor the system performance in real-time. Our areas of interest here are:

- Functional performance, such as monitoring of merchants, number of alerts and frauds, number of true- and false-positives.
- Operational performance, such as latency (how fast the system responds), number of transactions processed per second, and more.

Stackdriver sends us email and SMS alerts in rare cases of any issues.

What's next?

We're happy with our current results — the fraud rate is already low, but we're going to continue improving the prediction quality. Essentially, we see Sherlock as the end product that can be purchased and integrated by other banks and financial institutions.

> I often talk to people from banks, and some of them have already asked if we can sell our technology to them. Operations which banks process manually for hours, take just a few minutes at Revolut. If you're going to sign up millions of users and scale your business to new markets worldwide, you cannot rely on manual work alone.

Nikolay Storonsky, CEO and CO-founder of Revolut.

Join Revolut

Today, Revolut has 9 million customers, and that number continues to grow. If you're interested in helping us take Sherlock to the next level, check out our vacancies on the Careers page.

[

Careers | Revolut

Banks are powerful. They owe us nothing. They watch us. They punish us for our mistakes. They have more money, more...

www.revolut.com

](https://www.revolut.com/careers/)

P.S. Have you ever received a push notification from Sherlock? How did it make you feel? Let me know in the comments!