Solution

Provided by Michael Boerman on 2023-11-20 (GitHub repo)

Given...

- Bank A is integrating a new-to-them, vendor-provided fraud model
- the model is supervised and produces a score, not a binary classification
- the model relies on fingerprint, geolocation, and PII data
- the vendor is reputable, has existing banking clients, and has proven to reduce fraud by up to 30%
- the vendor has model documentation and can support requests for help
- the model is in compliance with banking regulations and industry best practices

I Assume...

- I assume Bank A has an existing fraud detection and mitigation infrastructure with other existing models such that this is fitting into an existing suite of models and infrastructure (*this assumption narrows the scope of the project and I think is more realistic to this job*)
- I assume Bank A has historical, labelled data for fraud, though not necessarily with fingerprint, geolocation, and PII data.
- I assume Bank A's historic data are diverse, represent the entire population dataset, and large enough to train the model.

What would be your framework and steps to ensure this model is integrated into fraud strategies in a compliant manner with signoff from relevant stakeholders? What tools would you use?

1. Stakeholder Sign-Off

Before the fun begins, I would talk with the stakeholders to answer the following questions:

1. The vendor claims the model can reduce fraud losses by 30%. Given that we already have fraud models in place (*see assumptions*), we might not reduce our losses by that number. What percent reduction would we consider a success? If we reduce losses by, say, 5%, is this project still worth it?
2. How much time and effort are you, the stakeholders, willing to spend in order to implement this model if the reduction in losses is only half the vendor's claim? What about 10%? 5%? 1%?
3. What metrics do you, the stakeholders, need to consider the project a success? Just implementing it, or seeing a particular loss reduction within a particular time frame? What will make this project a failure?
4. Would anything be a deal-breaker to you, the stakeholders, that would prevent us from implementing this model? For example, if the model sends our bank's data back to the vendor for their future training.
5. Are you, the stakeholders, able to talk through each item on your list to categorize as a requirement vs. a preference?

I'd also take this time to review documents from previous models implemented. I'd look for the criteria used to measure success any any concerns from stakeholders addressed.

2. Understanding the Model

The next step I would take is to read the vendor's documentation for a general understanding of both the model's methodology and it's implementation.

Understanding the methodology will help me think through how the model fits into our existing model suite. We aren't replacing an existing model, but if this new model's scope and methodology overlap significantly with another existing model, I'd need to consider their interaction effect.

Understanding how the model is to be implemented will help me prepare the infrastructure to handle the necessary inputs. For example inputs, can it read right from a cloud-hosted database? If so, does it require a particular dialect of SQL like T-SQL or MySQL -- and if so, do our other models use that dialect, too? For example outputs, can it write to a cloud-hosted database? Can we inspect the results before any actions are taken?

Together, these are analogous to a surgeon transplanting an organ. They need not only to ensure the organ will work for the recipient by checking its blood type against the patient's blood type, but also that the organ's tubes and vessels are of similar size and will align with the patient's body. The organ is only valuable if it's able to be integrated into the recipient, regardless of how well it performed in the donor!

The goal at this stage is not 100% understanding, but a pareto-principle approach to understand 80% of the model with 20% of the effort that would be required to understand 100% of it.

3. Compliance Check

Personally, I don't take the claims from company trying to sell me something as truth, and I'm certainly not going to stake my name on it. Therefore, I would dig into the code to better understand it in light of banking regulation standards (ie, the SR 11-7).

I'd also be focusing on data privacy in light of our company's protocols. I wouldn't plug-and-play a model unless I understood how it uses our raw data and even the hyperparameters. Does the software communicate any of this back to the vendor? I'd be asking this directly to the vendor for two reasons: the code may hide it, and having an answer in writing from the vendor as historic evidence.

If data is transmitted, I'd check it against our company's data privacy standards and communicate this to the stakeholders.

4. Gather Data

At this point I'd begin to look for the necessary inputs (fingerprint, geolocation, and PII) for the model. I'd look for the source of this data, see where else it's used in the bank and in Fraud MRM, and inspect it to understand outliers.

The data must be:

- labelled, so that I can check performance

The data would be nice to be:

- historic to our bank, not generic or vendor-provided

The data does not need to undergo feature engineering, since the independent variable is pre-defined (fingerprint, geolocation, and PII). Likewise, there is no need to select an algorithm.

I'd compile it into a central location, most likely SQL view compiling data from more than one table.

5. Pre-implementation Prototype and Evaulation

Next, I would create a prototype using the model. This will help me get ready to build the full infrastructure and also evaluate it's performance. Just as in *Step 2 - Understand the Model*, my goal here is an 80/20 prototype.

I'd then follow the process to split the data into training and test sets. At this stage, I would not introduce a 3rd set for validation because tuning the hyperparameters for a prototype is overkill. If only a small amount of data is available at this stage, I'd use k-fold cross-validation.

If the model's output is a binary classification, I'd look at precision, recall, and the F1 score. More likely, though, the model's output is a regression with a continuum showing the probability of a transaction being fraudulent. In this case, I'd look at metrics like MSE or MAE, taking into consideration the metrics used to evaluate our other models.

I'd compare these results against any of our existing models. If this model and any other use the same input data, I'd compare the results 1:1 for each transaction. Seeing if the results are similar for any given transaction (ie, probability of it beign fraudulent is within a ~5% difference) or not will help me understand if this model adds value to our suite. Subsequently, it might show that other models are superseded by this new model.

If all looks good from this step, I'd touch base with the stakeholders to inform them of the initial results.

6. Deployment

Now comes the fun: deploying the model into production. This takes that prototype proof-of-concept into the big leagues, falling into the realm of ML Ops / Engineering.

In brief, the steps would be:

1. Gather all data and package into a robust SQL view/table that won't break in the future
2. Re-train the model, this time using 3 sets (adding a validation set) in order to fine tune the hyperparameters
3. Package the model and its dependencies into a deployable artifact
4. Containerize & Orchestrate, which will allow it to run on any server at any time
5. Create a Git repository to store the code, allowing version control and issue management
6. Add any unit, regression, and integration tests into the code
7. Integrate the artifact into the existing CI/CD pipeline(s)

7. Monitoring

The other ML Ops / Engineering facet is continuous monitoring. I'd like to monitor:

- overall accuracy of predictions, assuming we eventually receive feedback on if a marked-fraud event was or was not actually fraudulent.
- MSE or MAE
- percent categorized above the threshold to be marked as fraudulent, as compared with our other fraud models
- which transactions each model mark above the threshold that the other models do not
- inference latency to determine how long the model takes to perform its regression
- throughput percentage to determine if the model can handle all transactions fed into it without delay
- resource utilization to determine the computation costs of the model (ie, "are we getting our money's worth out of this model?")
- data drift to determine if the incoming transactions represent different trends than the training data I used, indicating the need to retrain the model
- concept drift to determine if the relationship between the independent variables (fingerprint, geolocation, and PII) and the score differ from the relationship established when initially trained
- accuracy by segment to determine if any segments (by demographic, statement balance, seasonality) differ from the average
- details on errors to determine any correlation among them, indicating the model needs to be retrained or those transactions need to be excluded from this model
- metrics between model versions to determine MSE, MAE, etc. between different deployed versions of the model whenever changes are made

8. Stakeholder Sign-Off

I would then revist the charter established with the stakeholders at the beginning of the project. I'd take with me initial model results from Step 7, but ultimately, to any business partners, translate them into net losses reduces due to this model. After all, the high-level goal of a Fraud model team is to reduce losses for both the bank and their customers. If a new model doesn't contribute to this mission, then it's hard to justify the time, energy, and money spent on it.

I'd also circle back on data privacy regarding telemetry of raw data or hyperparameters sent back to the vendor.

9. Documentation and Training

My approach to documentation is iterative. When one waits until the end of a project to document, I find it never gets done! Nonetheless, writing my answer as a linear timeline facilitates placing documentation at the end.

I typically use the Diataxis framework for documentation, knowing that a one-size fits all document (or anything, really) rarely helps anyone. Instead, I break documentation into 4 types:

1. Tutorials (for exploration and learning)
2. How-To Guides (for detailed tasks)
3. Explanation (for nuanced prose of decisions made or methodologies used)
4. Reference (for quick, searchable, articles to refresh one's memory or find additional resources)

I am also a proponent of asynchronous training with the option of synchronous follow-ups. For example, in a how-to guide, I will screen record myself following the procedure, broken down into ~5 minute segments for each major step in the process. I will embed the video directly into the how-to guide for anyone to watch any time. If or when someone has follow-up questions, we can call and walk through it. This method reduces implicit "tribal knowledge" and also mitigates the risk of a small number of people knowing something that is not able to be accessed anywhere other than talking with them directly -- heaven forbid they leave the team!

Tools

Above all, I'd default to using tools that the Fraud MRM team at SoFi:

- already have access to
- already have been used in Fraud MRM
- already are familiar to people on the Fraud MRM team

This mitigates technical debt from tools that only 1 person knows how to use or aren't compatible or tested with other tools in use elsewhere.

In the case in which no tools fit these 3 criteria and any software could be used, I'd pick my tools as follows.

1. For all programming, I would use Python, preferably in an IDE like VS Code. I would use virtual environments to ensure the model can be deployed by anyone on any server.

2. For data input, I would consider using a SQL database, preferably cloud-hosted like Snowflake.

3. For the CI/CD pipeline, if one didn't already exist, I'd consider using Jenkins, GitHub Actions, GitLab CI/CD, Azure DevOps, or AWS CodePipeline.

4. For monitoring, if no tools were already in use, I'd consider using Fiddler or AWS SageMaker.

5. For stakeholder engagement, I'd consider using Tableau connected to the SQL table in which results are stored.

How would your recommendation change if this score was critical to address an ongoing large scale fraud attack?

If there is an urgent fraud attack and we need to stop the leak *now*, I'd expedite the process by doing the following:

1. Reduce stakeholder sign-off to
    - receive buy-in to implement the model regardless of cost
    - receive buy-in to accept the risk of extra false-positives by casting a wider net

Given the model still meets regulatory policies.

2. Consider using the vendor's pre-tuned model using their sample data

    - this would cut the time to deploy the model and likely produce ~80% as good results

3. Implement the prototype and refine parameters on a rolling basis

    - instead of prototype, train, tune, and deploy, I'd engage in dynamic model adjustment by training and tuning after deployment

4. Heighten performance monitoring

    - keep open communication with stakeholders regarding the number of transactions caught before/after this model was implemented
    - keep open keep open communication with stakeholders regarding the amount of false positives to align with business needs
    - keep a sharp eye on drift in input data to determine the direction of the attack