Testing AI and Machine Learning Systems in the Financial Industry

By Kerstin Kohout
AI-based tools have transformed from a vague, futuristic vision into actual products that are used pragmatically to make real-life decisions in financial services. Still, for most people, the inner workings of deep-learning systems remain a mystery.

If you don’t know what exactly is going on while the input data is fed through layer after layer of a neural network, how are you supposed to test the validity of the output? Are the days of simple tests with a clear and understandable result over?

Firstly, let’s make a clear distinction between testing applications that consume AI-based outputs and testing the actual machine learning systems.

If your application falls into the first category, there’s no need to worry – or to change your approach to testing. AI-based third-party tools don’t require any VIP treatment; they can be viewed as black boxes, just like “regular” deterministic third-party products you might use. Focus your effort on testing whether your own products behave correctly when presented with an output from the AI.

But what about the companies that create these machine learning systems? How do you go about verifying that they do what they should?
Understanding AI

If we’ve learned anything about AI and machine learning over the last decade, it’s that it’s all about data, and lots of it. This data plays a central part in your testing strategy: with that comes the need to address critical test data management challenges; such as compliance with privacy standards.

Let’s dive into how the data is used.

The most commonly used approach is to divide the data that’s available to you into three parts: a training set, a development set, and a test set. To understand how to test your AI, you’ll first need to know how these three sets play together to train a neural network.

When you develop a deep learning system, you work by feeding huge amounts of data in the form of a clearly defined input and expected output or result into a neural network. Then, you wait for the network to come up with a set of mathematical formulas that work best to calculate the correct expected output for most of the data points you provide it with.

Let’s say you’re in the process of creating an AI-based tool that detects fraudulent credit card transactions based on a variety of factors such as the amount spent and the location in relation to previous purchases made on the same account.

These transaction details, pre-processed to be computer-readable, are your input data, and each transaction has a defined output, or the expected result. That’s the training set.
Trying Out the Algorithm

Once the network has been busy optimizing for some time on your training set data, you will want to check how well it’s doing with its newly learned formulas. Your training algorithm already outputs how well it’s doing on the training examples, meaning the data you’ve been feeding it all this time. However, using this same data to evaluate the algorithm is not a good idea.

Chances are the network will detect fraud correctly in the transactions it’s seen many times, but that’s no indicator of how it will perform on different transactions like the ones it will see in production. Your fraud detection algorithm will only get one chance to assess each transaction it sees, and it needs to predict fraud reliably based on that.

The real question is: how does the algorithm perform when presented with completely new data that it hasn’t been trained on?

This new data set is called the development set, because you tweak your neural network model based on how well the trained network performs on this set. Simply put, if the network performs well on both the training set and the development set (which consists of transactions it isn’t optimized for because they were not part of the training set), that’s a good indicator that it will also do well on the transactions it will face day to day in production.

If it performs worse on the development set, your network model needs some fine-tuning, followed by some more training using the training set and, finally, an evaluation of the new, hopefully improved performance using the development set. Often you will also train several different networks and decide which one to use in your released product based on the models’ performances on the development set.
Addressing Test Data Management Challenges

One challenge that arises for financial firms is how to get suitable data.

The obvious solution in general is to simply pull a subset of the production data already available within the organization that is developing the neural network.

However, given the sensitive nature of financial data and regulations such as GDPR, this is not an option.

Whilst financial firms could mask production data so it no longer contains personal identifiable information, a quicker path to value may be to generate entirely synthetic test data.

Instead of expending effort on obfuscating credit card details, social security numbers and the like, financials can use modelling techniques to generate realistic, compliant data sets.

In both cases, it is important to ensure data validity and integrity using big data testing tools.
Choosing Dev and Test Data Sets

At this point you might ask yourself, isn’t that testing? Well, not really. Feeding the development set into your neural network can be compared to a developer trying out the new features they’ve built on their machine to see if they seem to work.

To thoroughly test a feature, though, a fresh pair of eyes – most commonly a test engineer – is required to avoid biases. Similarly, you’ll want to use a fresh, never-used data set to verify the performance of your machine learning system, as these systems become biased as well.

How does a computer become biased? As described above, during development you tweak your model based on the results it gets on the development set, so typically you will choose the model that works best with this specific data set.

For our fraud detection example, if the development set coincidentally consisted mostly of transactions performed by individuals as opposed to businesses, the network would have troubles dealing with businesses’ credit card statements, because you chose the network model that doesn’t perform best for those circumstances.

Of course, you should try to use well-balanced training and development sets, but you won’t really know if you managed to do that without using a completely new data set to test the final algorithm. The network’s performance on the test set is the most reliable indicator of how it will perform out in the real world.

For that reason, it’s important to choose a test set that resembles the data your AI will receive in production as closely as possible.

For the fraud detection algorithm, that means it should include transactions made in different areas of the world, both by businesses and individuals, and the amounts spent should vary. These transactions must be labelled as correctly as possible as fraudulent or not fraudulent.

Now, for the test, you simply let the algorithm assess all the test examples and compare the algorithm’s output to the expected output. If the percentage of correctly assessed transactions is satisfying, the test is successful.
Defining Requirements

Those of you who are experienced testers will certainly ask, what does “satisfying” mean in terms of those results? In traditional testing, the answer is usually quite clear: The output should be correct for all test cases.

However, this will hardly be possible when it comes to machine learning algorithms, especially for complex problems such as fraud detection. So, to come up with a concrete number, the best place to start is to look at how qualified humans perform at that exact task.

For our fraud detection example, you’ll want to assess the performance of a team of trained fraud analysts and use that as your goal. If your AI detects fraud as well or better than that, we can consider the test results satisfactory.
Risk-based Testing in the World of AI

Up until now, we’ve been talking about the percentage of correctly assessed credit card transactions as the metric to look at in the test results. In other words, you’d evaluate your deep-learning algorithm based on how many authentic transactions were flagged as fraudulent and how many fraudulent transactions as authentic. However, these two things are not the same in the real world.

On top of angering the affected customer, a wrongfully blocked transaction could have a severe impact on the customer’s business and may even cause financial damage for them. However, the losses incurred due to fraudulent purchases can be much higher depending on the amount spent.

For that reason, you will need to decide which weight to place on false positives and false negatives. Like risk-based testing of non-AI tools, the decision on whether to release your product in its current state even though some test cases might fail depends on the risk associated with the failing test case.

 Blocking a low value consumer purchase is a low risk; allowing a fraudulent charge running into the millions to go through is a considerably higher risk.

If your application falls into the first category, there’s no need to worry – or to change your approach to testing. AI-based third-party tools don’t require any VIP treatment; they can be viewed as black boxes, just like “regular” deterministic third-party products you might use. Focus your effort on testing whether your own products behave correctly when presented with an output from the AI.

But what about the companies that create these machine learning systems? How do you go about verifying that they do what they should?
Ruling Out Data Biases

Another important part of testing a complex deep-learning financial software system is bias testing.

Because neural networks base their decisions strictly on the data they are trained on, they run a risk of mimicking biases we see when humans make decisions, since these biases are often reflected in data sets that were collected.

Let’s go back to our fraud detection example.

Say our data set was collected by combining random authentic transactions with transactions reported as fraudulent by card owners. A lot of people might not pay close attention to low value charges and be more likely to miss fraudulent transactions that are under a certain threshold.

Since the network’s training is based on data collected from actual purchases, this bias will likely be transferred to your algorithm. The network will learn that low value purchases are generally authentic and therefore fail to detect fraud in these circumstances.

To rule out biases in neural networks, you’ll need to carefully analyze the test results – especially the failures – and try to find patterns. For example, you could compare the algorithm’s success rate for transactions in relation to their values. If there is a strong correlation between these two factors, the algorithm might have become biased during training.

If there is any reason to suspect a bias, you’ll need to perform additional exploratory tests with tailored data sets to confirm or disprove your suspicion.
The Right Tools

These complexities might lead you to conclude that you’ll need highly specialized tooling to test your deep-learning system. However, rest assured that most of the hard work is taken over by the AI developers.

Weight calculations, data processing, and result evaluation are already woven into the neural network during the development process, as they are required right from the beginning. Once the neural network is built, you can pass any data set into it and it will output the result, along with the overall accuracy of said result. All there’s left to do is to switch your development set with your test set and look at your network’s performance. No new tools are required for that.
It’s All Still Testing

Testing AI systems is not that different from testing deterministic tools.

While there are big differences in the details, it’s still the same process: define your requirements; assess the risk associated with failure for each test case; create and curate appropriate test data; run your tests; and evaluate whether the weighted, aggregated results are at or above a defined level. Then add some exploratory testing into the mix to find bugs in the form of biased results.

Testing AI in complex financial systems is not magic: it’s just testing.
About the Author

Kerstin Kohout is a passionate software engineer currently working on machine learning projects at Tricentis.