TEST AUTOMATION FOR MACHINE LEARNING: AN EXPERIENCE REPORT

ANGIE JONES
NORTH CAROLINA, USA

MACHINE LEARNING IS A METHOD OF equipping computers with artificial intelligence so that they are able to utilize information that they have been exposed to in order to further develop themselves without the need for additional programming. Not long ago, I had the unique opportunity to work on an exciting project that utilized machine learning as one of its most attractive features. The product was an advertising application that used analytics to determine the best ad that a business could provide to a specific customer in real-time. Much like the advertisements we see when surfing the web are tailored to us based on our browsing habits, this software allowed businesses to show specific products to visitors of their websites, users of their mobile apps, or customers calling into their call centers based on characteristics of that person and their current context. This type of algorithm isn't entirely new, however, what gave us the advantage over competitors was the coupling of real-time decision making along with a learning algorithm that promised to make a business's advertising better over time.

I was tasked with verifying that the learning algorithm was working as intended, and chose test automation as a tool to assist me in this effort. While exciting, this was also quite the challenge. Here's a look into my journey.

Learning how it learns

As a tester, and more specifically an automation engineer, I am accustomed to working on scenarios where the expected result is exact. For example, when working with a typical e-commerce application, if I add a product to my empty cart that costs $29.99 and I modify the quantity in the cart to add two more of this product, my expected subtotal is $89.97. Exact. Testable. Automatable.

> “with machine learning, there is no exactness. There’s a range of valid possibilities that could occur based on what the machine has learned”

I quickly discovered that with machine learning, there is no exactness. There’s a range of valid possibilities that could occur based on what the machine has learned and how it will utilize that information in its future decisions. So the first thing I needed to do was learn how the algorithm learns so that I could determine the range of correctness. This required quite a bit of deep diving under the surface of the application into the code and talking extensively with developers to understand what triggers this machine to learn, how it learns, and how it uses what it has learned to make decisions.

The code was quite complex and I’m not ashamed to admit that the algorithms were over my head. However, I pressed on until I felt I had a fundamental understanding of how it worked and therefore was able to come up with scenarios to test it.

Training it

Before testing any learning, I had to give the system something to learn. Without any learned patterns, the system was designed to randomly recommend an ad to be provided to a customer. I created a set of five ads that the system could choose from.

To trigger learning, I needed to simulate a substantial number of customer transactions to occur within a customized window of time. After this period of time elapsed, the system would then build a model based on ads that led to conversions during that time. This essentially would represent what the machine has learned.

I started with a very simplistic (and stereotypical) scenario where recommended ads were requested for 2,000 unique customers who identified as either male or female. I randomly generated the gender of each customer, so there was always between a 48-50% split of one customer type versus the other.

My goal was for the vast majority of women to buy the dress after being provided with the dress ad, and the vast majority of men to buy the trousers after being provided with the trouser one. Since there were five ads, the chance of a woman being shown the dress ad or a man being shown the trousers ad was 20%. One thing I learned about the system during my initial investigation of the learning algorithm was that too much of something was not considered a good thing. Meaning, if 100% of women who were shown the dress ad ended up buying the dress, then by design, the system would treat it as an abnormality. So, to make it more realistic, I skewed the conversions a bit so that is was more like a 65% conversion rate. Also, to ensure models were created for each ad, I had any customer buy the product for whatever ad they were shown 2% of the time.

“one thing I learned about the system during my initial investigation... was that too much of something was not considered a good thing”

This essentially ended up with a pool where roughly 1,000 of the customers were women, of which approximately 200 of them were provided with an ad for a dress. Of that, about 130 of the women actually bought the dress. Another 20 or so of the 1,000 women made random purchases based on the ad they were shown which could include a few dresses. A similar pool was generated for men and trousers.

Within my automation code, I kept the results from the training exercise in a data structure that looked something like this after a given run:

Ad	Women who saw	Women who purchased	Men who saw	Men who purchased
Dress	**202**	**133**	204	3
Trousers	205	3	**196**	**128**
Scarf	199	2	203	3
Gloves	194	1	198	1
Belt	197	1	202	2

Testing it

Given the training I’d simulated, it was clear to me that the system should have learned that women are likely to buy dresses and men are likely to buy trousers. Having learned this, I also expected the system to now offer women dresses more often and men trousers more often.

> “I’m no statistician or data analyst so I resorted to using POCS - plain ole common sense”

However, what does “more” mean exactly? No one could really give me an exact answer or even a range for this. I’m no statistician or data analyst so I resorted to using POCS - plain ole common sense. If 20% of women were shown the ad for the dress when there was no learning in place, then after the system has learned, I expect more than 20% of the women to be shown the dress ad and I expect the four other ads to be shown less than the dress ad and less than they were shown before the learning. Likewise, I expected a similar result for men but with the trouser ad.

The same scenario ran again but this time with 2,000 new unique customers. I captured the ads shown and purchases made into a new data structure and compared it with the before-learning data.

Ad	Women who saw before	Women who saw after	Men who saw before	Men who saw after
Dress	**202**	**890**	204	874
Trousers	205	66	**196**	**59**
Scarf	199	18	203	24
Gloves	194	14	198	19
Belt	197	15	202	21

Do you see what’s wrong here? The system indeed learned that the dress was popular but it began showing the dress ad to most customers, regardless of gender.

Of course, I began wondering what *I* did wrong...being a novice to machine learning and all. Did I not train the system properly? I cleared all models that the system had stored and re-ran the test. This time I noticed that after the system learned, it began offering the trousers ad to most customers, regardless of gender! Huh?

After investigating the data collected for each run, I noticed that whichever ad led to the most conversions during that training period would be the product that the system learned was most favorable. The second most popular one would be shown more than the random 20% of the time, but less than the most popular ad. It became clear to me that the customer demographic was not being considered in these decisions and that this was clearly a bug.

Doubting my findings

Writing up the bug report was similar to writing this article. Because there's no exact expected result, I needed to include detailed information about how I trained the system, why my expectations were what they were, and how I verified them.

There were only two developers assigned to work on the learning module. One of which was a brilliant developer and understood this machine learning stuff way more than anyone else in the company - let's call him Arty, and the other a junior developer looking to learn from him.

Arty received my bug report but was not convinced that there was a problem with the system. You see, Arty had set up his own scenario, which was different from mine, and you guessed it...it worked on his machine!

Arty closed my bug report.

Being that I was new to this and Arty was the in-house expert, I second-guessed myself yet again and went back to my automation code to study my setup. I ran the scenario multiple times to generate more models and further enforce the learning that women buy dresses and men buy trousers. That didn't seem to change the outcome. After a day of racking my brain, I was convinced that my initial test was correct and regardless of what Arty said, the way the system was learning was not as intended.

"being that I was new to this and Arty was the in-house expert, I second-guessed myself"

I reopened the bug report.

Proving my findings

Arty was convinced that the issue I reported was not a bug and I was convinced that it was. Consequently, this high-priority bug sat there for several weeks. Eventually, it became time to release the software and of course we had to come to some resolution before doing so. The product owner called a meeting with Arty and myself to hear both arguments.

As a tester, I believe my primary role is to provide information about the product, and while I don't fight to get bugs fixed, I do believe a part of my role is also to advocate for the customer. So, while I had the data from my findings, I presented my observations to the product owner

with customer impact instead. I explained that 65% of men bought trousers, yet our system is much more likely to present a man with an ad for a dress, even when there's only a 1% chance that this will lead to a conversion. I highlighted that this issue will cause businesses to miss out on potential sales, it goes against what we've marketed our product to be, and I don't think our customers will find this acceptable.

Arty presented his beliefs and went into a lot of theory about machine learning. After the lesson, he then changed the scenario on his machine to resemble mine and proceeded to run it during the meeting to prove that it, in fact, works.

It didn't.

Arty finally conceded that we indeed had a bug on our hands. He also kicked himself for ever doubting me in the first place, which I'll admit, felt pretty good.

What I learned

As technology advances, so must our test approaches. In this case, hard rules such as "the exact results must be known in order to test or automate" had to be revisited and challenged.

The need for automation became much more apparent as thousands of customer transactions needed to be simulated and analyzed. Yet, coding the scenario was only one small part of actually testing the feature. There was still a need for thoughtful and critical testing, and this was definitely a case where automation was not just about validating that certain features were correct, but was utilized as a tool to support that testing.

I also learned that no matter how smart machines become, having blind faith in them is not wise; plain ole common sense can still trump an algorithm. While we should embrace new technology, it's imperative that we, as testers, not allow others to convince us that any software has advanced beyond the point of needing testing. We must continue to question, interrogate, and advocate for the human customers that we serve.

Angie Jones is a Consulting Automation Engineer who advises several scrum teams on automation strategies and has developed automation frameworks for countless software products. As a Master Inventor, she is known for her innovative and out-of-the-box thinking style which has resulted in more than 20 patented inventions in the US and China. Angie shares her wealth of knowledge internationally by speaking and teaching at software conferences, serving as an Adjunct College Professor of Computer Programming, and leading tech workshops for young girls through TechGirlz and Black Girls Code.
