CHAPTER 10

Verifying Intelligent Experiences

Intelligent experiences fail for two main reasons:

1. Because they are implemented incorrectly: the experiences have a bug, are confusing, or don't help users achieve their goals.
2. Because the intelligence is incorrect: it suggests the wrong automation, is certain of something that isn't true, or proposes the wrong ordering of information.

For example, an intelligent experience might:

- Present a list of images with faces, but fail to include the user's favorite photo from their wedding. Is the image excluded because the experience has a bug? Or is it excluded because the intelligence didn't recognize that the image had a face in it?
- Turn the lights off when the room is already sort of dark. Did the experience turn the light off because it knew the sun was coming up soon (even though it knew the room was too dark)? Or did the light turn off because the intelligence thought the room was empty?
- The user hates horror movies, but the system is recommending lots of horror movies. Is it doing this because the user's husband set a preference in some hidden menu to hide all the romantic comedy movies forever and ever no matter what? Or is it doing this because the intelligence examined the movies and thought they were adventure, not horror?

G. Hulten, *Building Intelligent Systems*, https://doi.org/10.1007/978-1-4842-3432-7_10

Verifying that an intelligent experience is functioning properly, separate from the quality of intelligence, is critical to producing effective intelligent experiences.

Getting Intended Experiences

So the intelligence might be right, the intelligence might be wrong; fine. But how did the rest of the system perform?

- Did the user get the intended experience when the intelligence was correct?
- Did the experience actually help to achieve the user's (and the system's) goals?
- Does the experience remain effective as the system and the intelligence evolves?

This section talks about ways of isolating the experience from the rest of the changes that might occur in an Intelligent System so you can have some basis for verification.

Working with Context

The context represents the state of the Intelligent System when an intelligent experience is triggered. This can include all the variables the intelligence might take into account in producing its decisions. It should also include the variables that impact how users will interpret the system's outputs and how they will react. For example:

- The context in a computer vision system might include the lighting in the environment, the camera properties, the users and their features (ethnicity, gender, clothing), and so on.
- The context in a music playing system might include the user's ratings of songs, the list of songs the user owns, the recent listening history, and so on.
- The context in a home automation system might include the number and types of sensors, the rooms they are placed in, the distance between them, and so on.

When you want to verify an intelligent experience works in a particular context, you need to create the context and try it. For example, you need to set up the computer vision system in exactly the right way to recreate the context, get the lighting just right, bring in the same users, put them in front of the camera, and have them smile exactly right, and so on.

This can be problematic. Particularly for large, open-ended problems that have many, many possible contexts—an infinite number of them.

When verifying intelligent experiences, it can help to have some tools to help produce (or capture) important contexts, explore them, and replay them in the intelligent experience so you can test them again and again as the Intelligent System evolves. Some options for working with contexts include:

- **Manually producing contexts** the same way users would, by using the system, creating an account, building up the history of interactions, browsing to the right place, and so on. This can be extremely challenging and error prone, as the contexts can be subtle; for example, the change in inflection when speaking to a computer can affect how well a speech recognition system will perform. They can be quite extensive—for example, usage history over a long period of time.
- **Crafting via an editor** or some other tool that lets contexts be created, edited, captured, and played back to the system. For example, a tool that specifies a house layout, the location of sensors, and the various readings that are on the sensors at a particular time. The level of sophistication and automation can vary, but these types of tools can be very helpful in developing baseline verifications.
- **Recorded via usage** either in a lab setting or from actual users. Examples include allowing the system to dump the context at any time, and creating tools to load the dumps and evaluate them against updated versions of the intelligent experience and intelligence. These types of approaches can help get actual usage and a great deal of variety into the verification process.

When verifying intelligence it is good to have all of these options. A typical work-flow might look like this:

- Reproducing a problem that a user reports.
- Capturing the exact context that caused the report.
- Viewing the context in some kind of tool to see what was really going on (not just the part the user was complaining about).
- Making a change to the experience.
- Executing the context again to see if the change would fix the problem.
- Communicating the problem to intelligence creators (but they may or may not be able to help).

Note that a context for verifying experiences is similar to, but different from the data needed for creating intelligence. To create intelligence you also need to capture information about the outcome. But the context alone can help verify (and improve) the experience.

Working with Intelligence

The workflow of tracking and replaying contexts is similar to tracking and fixing bugs in other complex software products. But changing intelligence can make it difficult. This is because the experience a user sees results from combining a particular context with a particular intelligence—and intelligence changes.

Because of this, it also helps to have a collection of known intelligences that can be combined with known contexts to see the results. Some options for getting intelligences to help with verification include:

- **Live intelligence** that is just what users will see. If things are going well, this will be the most accurate intelligence available, and it will be the best way to verify what users are actually seeing. But it may also change very rapidly (as intelligence evolves), requiring some work to distinguish intelligence changes from experience changes. One example of a problem this might cause is that you have an experience problem you are working to fix and then… it disappears.

You haven't changed anything yet, but maybe the intelligence changed out from under you and you can no longer reproduce the problem.

- **Fixed intelligence** that is the same every time it is executed. For example, some sort of checkpoint on the live system's intelligence (maybe the first day of alpha testing or the first day of every month). This intelligence is similar to the live intelligence, but provides a stable baseline to use for working with experience problems over an extended period of time.
- **Simple intelligence** that is somewhat interpretable. For example, instead of using a model produced by machine learning, use a small set of heuristic rules. This can be useful when debugging problems—by having just enough intelligence to cause interesting problems, but not so much that the intelligence errors overwhelm the experience problems.
- **Dumb intelligence** that always gives the same answer no matter what context is provided. For example, no matter what bands the user likes, the intelligence recommends Dire Straits. A dumb intelligence can help rule out obvious experience problems.

When verifying intelligence it helps to have all of these options available.

Bringing it Together

Common types of errors to look for include:

- The context being incorrectly interpreted by the intelligence, resulting in unexpected output.
- The intelligence being misinterpreted by the experience, resulting in systematically incorrect experiences.
- The rendering of the user experience being incorrect for some combinations of context and intelligence, resulting in hard-to-interpret, unappealing, or illegible results.

Achieving Goals

The experience plays a very important role in achieving a system's objectives.

By presenting intelligence, deciding how and how often users see it, deciding how it is colored, where it is displayed, how often it happens, and what words or icons are used to describe it, an experience can make a poor-quality intelligence into a useful tool; an experience can also render a high-quality intelligence useless.

Consider a system designed to protect users from scams on the Internet. If the user visits a page that the intelligence thinks is a scam web page, the experience gives them a warning of some sort.

If the intelligence were perfect, the warning could be very forceful (perhaps blocking access to the page and giving the user no recourse) but intelligence is never perfect, so the warning needs to balance the risk of the user getting scammed with the risk of the user being scared away from a site they actually wanted to go to.

So how do we verify the effectiveness of any particular experience?

This can be accomplished by isolating the job of the intelligence from the job of the experience.

The job of the intelligence is to flag scam sites. It might be right; it might be wrong.

The job of the experience is to:

1. Successfully keep users from being scammed when intelligence correctly flags a scam.
2. Minimize the cost to the user when the intelligence incorrectly flags a non-scam as a scam.

And so these are the properties that need to be measured to verify that the experience is doing its job:

- What percent of users who correctly received a warning got scammed anyway?
- What percent of users who incorrectly received a warning got turned away from a legitimate site?

More generally, for every possible outcome of the intelligence—all the ways it could have been right and all the ways it could have been wrong—consider what the role of the experience is in that setting—and measure it.

And note: these types of things are very difficult to measure in a development setting—accurately measuring them often requires real users making real decisions.

Continual Verification

Intelligence changes over time. User behavior changes over time, too. These changes can cause experiences to become less effective over the life of a system. Or the changes could result in opportunities to make the experience more forceful and get more benefit.

Some changes that can affect the experience are:

1. The quality of the intelligence could become better or worse.
2. The quality of the intelligence might remain the same, but the types of mistakes it makes might change.
3. Users might become accustomed to (or fatigued by) the experience and start to ignore it.
4. New users might adopt the system and have different behaviors than the original users.

In an Intelligent System there are many sources of change. And most changes provide opportunity (at least if you're an optimist, you'll consider them opportunities) to rebalance the parts of the system to achieve better results—as long as you know the change happened and how the change affected the system.

Adapting the experience isn't always the right answer—but it might be.

Summary

Verifying that an intelligent experience is working can be complex, because mistakes can be caused by the experience, by the intelligence, by having the user in some context you didn't anticipate, or because something has changed.

When there are a lot of mistakes they can become overwhelming, and it is useful to have some ways to simplify the problem and isolate what is going on. This can also help prevent lots of finger-pointing: "This is an intelligence mistake!" "No, it is an experience problem." "No, it is an intelligence problem." … And so on.

It is good to have a way to capture, inspect, and manipulate contexts. This can be done through:

- Manually producing them.
- Having tools to inspect and modify them.
- Having ways to record them from real users and situations.

It is also good to have ways to play contexts back against various versions of the intelligence and see what the experience would produce. Common intelligences used in verification include:

- The live intelligence that users are getting.
- A checkpoint of the live intelligence that can be used over an extended period without changing.
- A simple intelligence that is easy to interpret (maybe some hand-crafted rules).
- A dumb intelligence that always says the same thing.

It is also important to verify that the intelligent experience is helping users achieve their objectives. This is most commonly done by looking at live interactions when the intelligence was right and ensuring that users are ending up with good outcomes (and not being confused or misled by the experience).

And Intelligent Systems are always changing. The effectiveness of the intelligent experience needs to be constantly reverified. Change also creates opportunities to rebalance and provide more user value—if you are tracking it and have the tools to identify the effects of the change.

For Thought…

After reading this chapter, you should be able to:

- Isolate experience issues (from intelligence ones) and verify that the experience is working as intended on its own.
- Build an effective test plan for an intelligent experience.
- Understanding the importance of reverifying as the intelligence and user-base change.

You should be able to answer questions like these:
Consider an Intelligent System you use regularly.

- Sketch a plan to verify that its experience is functioning correctly.
- Design a simple intelligence to help verify this. Describe three contexts you would manually create to execute this simple intelligence as part of verification. Why did you pick those three?
- How could you monitor the system to detect that the experience is losing effectiveness?