Chapter 13. Feedback Loops

Now that we have a smooth pipeline for putting a machine learning model into production, we don't want to run it only once. Models shouldn't be static once they are deployed. New data is collected, the data distribution changes (described in Chapter 4), models drift (discussed in Chapter 7), and most importantly, we would like our pipelines to continuously improve.

Adding feedback of some kind into the machine pipeline changes it into a life cycle, as shown in Figure 13-1. The predictions from the model lead to the collection of new data, which continuously improves the model.

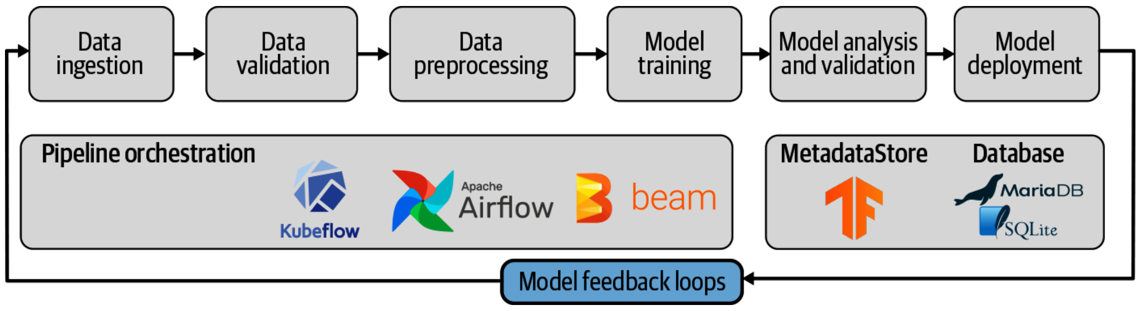

Figure 13-1. Model feedback as part of ML pipelines

Without fresh data, the predictive power of a model may decrease as inputs change over time. The deployment of the ML model may in fact alter the training data that comes in because user experiences change; for example, in a video recommendation system, better recommendations from a model lead to different viewing choices from the user. Feedback loops can help us collect new data to refresh our models. They are particularly useful for models that are personalized, such as recommender systems or predictive text.

At this point, it is extremely important to have the rest of the pipeline set up robustly. Feeding in new data should cause the pipeline to fail only if the influx of new data causes the data statistics to fall outside the limits set in data validation, or if it causes the model statistics to move outside the boundaries set in model analysis. This can then trigger events such as model retraining, new feature engineering, and so on. If one of these triggers occurs, the new model should receive a new version number.

In addition to the collection of new training data, feedback loops can also provide information on the real-world use of the model. This can include the number of active users, the times of day when they interact with it, and many other pieces of data. This type of data is extremely useful for demonstrating the value of the model to business stakeholders.

FEEDBACK LOOPS CAN BE DANGEROUS

Feedback loops can also have negative consequences and should be approached with caution. If you are feeding the model's predictions back into new training data with no human input, the model will be learning from its mistakes as well as its correct predictions. Feedback loops can also amplify any biases or inequalities that are present in the original data. Careful model analysis can help you spot some of these situations.

Explicit and Implicit Feedback

We can divide our feedback into two main types: implicit and explicit.[1] *Implicit* feedback is where people's actions in their normal usage of a product give the model feedback—for example, by buying something suggested by a recommender system or by watching a suggested movie. User privacy needs careful consideration with implicit feedback because it's tempting to just track every action that a user takes. *Explicit* feedback is where a user gives some direct input on a prediction—for example, giving a thumbs-up or thumbs-down to a recommendation or correcting a prediction.

The Data Flywheel

In some situations, you may have all the data you need to create a new product powered by machine learning. But in other cases, you may need to collect more. This happens particularly often when dealing with supervised learning problems. Supervised learning is more mature than unsupervised learning and generally provides more robust results, so the majority of models deployed in production systems are supervised models. Frequently, the situation arises that you have large amounts of unlabelled data but insufficient labelled data. However, the growth of *transfer learning*, as we used in our example project, is starting to remove the need for vast amounts of labelled data for some machine learning problems.

In the case where you have a lot of unlabelled data and need to collect more labels, the *data flywheel* concept is especially useful. This data flywheel allows you to grow your training dataset by setting up an initial model using preexisting data from a product, hand-labelled data, or public data. By collecting feedback on the initial model from users, you can label the data, which improves the model predictions and thus attracts more users to the product, who label more data, and so on, as illustrated in Figure 13-2.

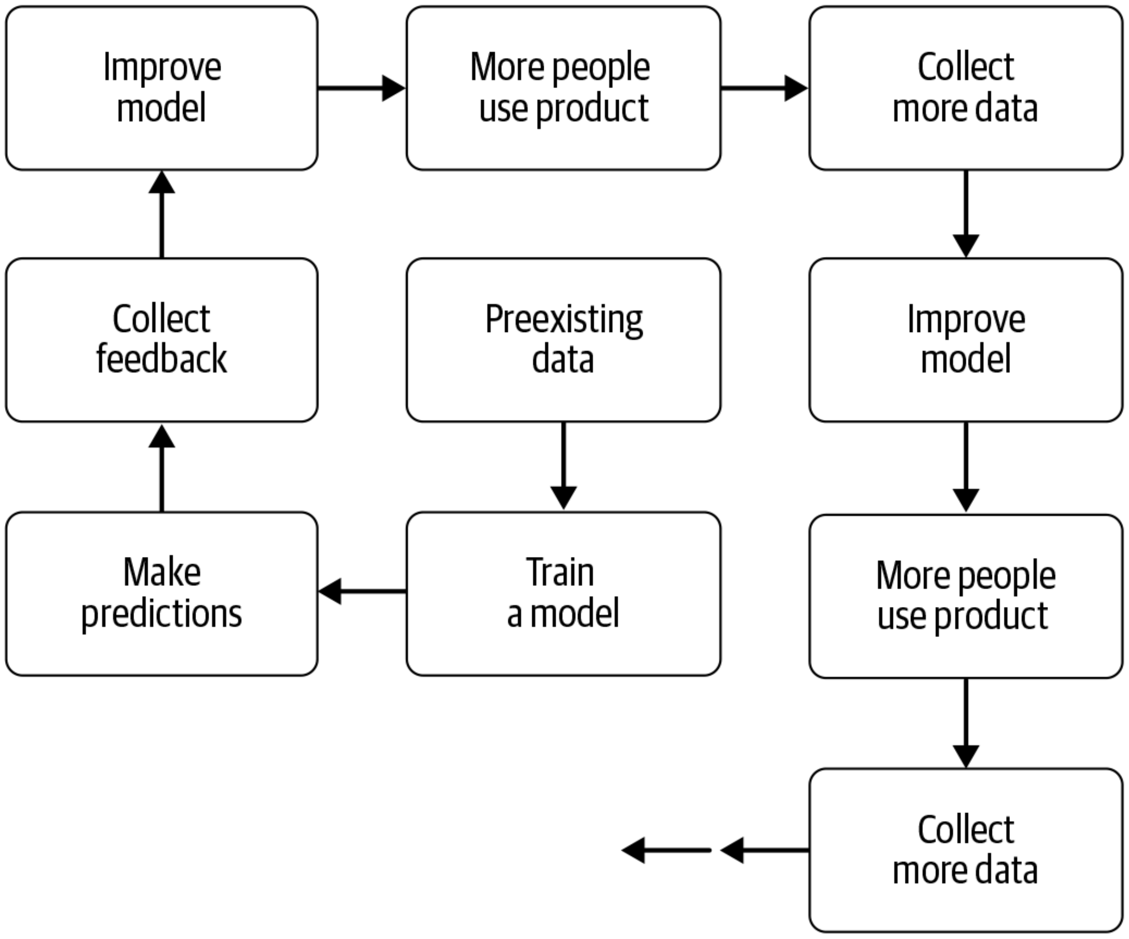

Figure 13-2. The data flywheel

Feedback Loops in the Real World

Some of the most familiar examples of feedback loops in machine learning systems occur when a model's predictions are exposed to a customer. This is particularly common in recommender systems, where a model predicts the top *k* most relevant choices for a specific user. It's often difficult to collect training data for recommender systems in advance of launching a product, so these systems are often heavily dependent on feedback from their users.

Netflix's movie recommendation system is a classic example of a feedback loop. The user gets movie recommendations and then provides feedback by rating the predictions. As the user rates more movies, they receive recommendations that are more closely tailored to their tastes.

Originally, when the main business of Netflix was shipping DVDs in the mail, it used a one to five star rating system to collect DVD ratings, which signaled that the customer had actually watched the DVD. In this situation, Netflix was only able to collect explicit feedback. But when its business changed to streaming movies online, the company was also able to collect the implicit feedback of whether a user watched the movies that were recommended to them and whether the user watched the whole movie. Netflix then switched from the one to five star rating system to a simpler thumbs-up or thumbs-down system, which allowed it to collect more feedback because the system required less time from users. In addition, the finer-grained ratings may not have been so actionable: how should a model respond if a movie is rated three stars? A three-star review doesn't signal that the prediction is correct or incorrect, whereas a thumbs-up or thumbs-down gives a clear signal to the model.[2]

Another example of a feedback loop—in this case a negative one—is Microsoft's infamous Twitter bot TAY. This hit the news in 2016 when, within 16 hours of its launch, it was taken offline because of its offensive and sometimes racist tweets. Before it was taken offline, it had tweeted over 96,000 times. It was retrained automatically based on replies to its tweets, which were deliberately provocative. The feedback loop in this situation was that the system took the replies to its initial tweets and incorporated them into its training data. This was probably intended to make the bot sound more human, but the outcome was that it picked up on the worst replies and became extremely offensive.

WHAT COULD GO WRONG?

It's important to think about what might go wrong with a feedback loop, as well as the best-case scenario. What is the worst thing that your users might do? How do you protect against bad actors who may want to disrupt your system in an organized or automated way?

A third example of real-world feedback loops comes from Stripe, the online payment company.[3] Stripe built a binary classifier to predict fraud on credit card transactions, and its system would block transactions if the model predicted that they were likely fraudulent. The company obtained a training set from past transaction data and trained a model on it, which produced good results on the training set. However, it was impossible to know the precision and recall of the production system because if the model predicted that the transaction was fraudulent, it was blocked. We can't be sure whether it was in fact fraudulent because it never happened.

A larger problem arose when the model was retrained on new data: its accuracy decreased. In this case, the feedback loop caused all the original types of fraudulent transactions to be blocked, so they were unavailable for new training data. The new model was being trained on the residual fraudulent transactions that hadn't been caught. Stripe's solution was to relax the rules and allow a small number of charges to go through, even if the model predicted that they would be fraudulent. This allowed it to evaluate the model and provide new, relevant training data.

CONSEQUENCES OF FEEDBACK LOOPS

Feedback loops will often have some consequences that weren't apparent during their design. It's essential to keep monitoring the system after it has been deployed to check that the feedback loop is leading to positive change rather than a negative spiral. We suggest using the techniques in Chapter 7 to keep a close eye on the system.

In the preceding example from Stripe, the feedback loop caused the model's accuracy to decrease. However, an increase in accuracy can also be an undesirable effect. YouTube's recommendation system is designed to increase the amount of time that people spend watching videos. The feedback from the users means that the model accurately predicts what they will watch next. And it's been incredibly successful: people watch over one billion hours of video on YouTube every day. However, there are concerns that this system leads people toward watching videos with increasingly extreme content. When systems become very large, it's extremely hard to anticipate all the consequences of the feedback loop. So proceed with caution and ensure there are safeguards for your users.

As these examples show, feedback loops can be positive and help us obtain more training data that we can use to improve a model and even build a business. However, they can also lead to serious problems. If you have carefully chosen the metrics for your model that ensure your feedback loop will be a positive one, the next step is to learn how to collect feedback, which we will discuss in the next section.

Design Patterns for Collecting Feedback

In this section, we'll discuss some common ways of collecting feedback. Your choice of method will depend on a few things:

- The business problem you're trying to solve
- The type and design of the app or product
- The type of machine learning model: classification, recommender system, etc.

If you're planning to collect feedback from the users of your product, it's very important to inform the user what's happening so that they can consent to providing feedback. This can also help you collect more feedback: if the user is invested in improving the system, they are more likely to provide feedback.

We will break down the different options for collecting feedback in the following sections:

- "Users Take Some Action as a Result of the Prediction"
- "Users Rate the Quality of the Prediction"
- "Users Correct the Prediction"
- "Crowdsourcing the Annotations"
- "Expert Annotations"
- "Producing Feedback Automatically"

While your choice of design pattern will be driven to some extent by the problem that your machine learning pipeline is trying to solve, your choice will affect how you track the feedback and also how you incorporate it back into your machine learning pipeline.

Users Take Some Action as a Result of the Prediction

In this method, our model's predictions are shown directly to a user, who takes some online action as a result. We record this action, and this record provides some new training data to the model.

An example of this would be any kind of product recommendation system, such as the one used by Amazon to recommend a next purchase to their users. The user is shown a set of products that the model has predicted will be of interest. If the user clicks on one of these products or goes on to buy the product, the recommendation was a good one. However, there is no information on whether the other products that the user didn't click on were good recommendations. This is implicit feedback: it does not provide exactly the data that we need to train the model (this would be ranking every single prediction). Instead, the feedback needs to be aggregated over many different users to provide new training data.

Users Rate the Quality of the Prediction

With this technique, a model's prediction is shown to the user, who gives some kind of signal to show that they like or dislike the prediction. This is an example of explicit feedback, where some extra action must be taken by the user to provide new data. The feedback could be a star rating or a simple binary thumbs-up or thumbs-down. This is a good fit for recommender systems and is especially useful for personalization. Care must be taken that the feedback is actionable: a rating of three stars out of five (such as the preceding Netflix example) does not give much information about whether a model's predictions are useful or accurate.

One limitation of this method is that the feedback is indirect—in the recommender system situation, users say what are poor predictions but do not tell you what the correct prediction should be. Another limitation of this system is that there are a number of ways that the feedback can be interpreted. What a user "likes" may not necessarily be something that they want to see more of. For example, in a movie recommendation system, a user may give a thumbs-up to show that they want to see more movies of the genre, by the same director, or starring the same actors. All these nuances are lost when it's only possible to give binary feedback.

Users Correct the Prediction

This method is an example of explicit feedback, and it works as follows:

- Predictions from a lower-accuracy model are shown to the user.
- The user accepts the prediction if it is correct or updates it if it is incorrect.
- The predictions (now verified by a user) can be used as new training data.

This works best in cases where the user is highly invested in the outcome. A good example of this would be a banking app through which a user can deposit checks. An image recognition model automatically fills in the check amount. If the amount is correct, the user confirms it; if it is incorrect, the user inputs the correct value. In this case, it's in the user's interests to enter the correct amount so that the money is deposited into their account. The app becomes more accurate over time as more training data is created by its users. If your feedback loop can use this method, this can be an excellent way to collect a lot of high-quality, new data quickly.

Care must be taken to only use this method in cases where the objectives of the machine learning system and the user are strongly aligned. If the user accepts incorrect responses because there is no reason for them to put in effort to change it, the training data becomes full of errors and the model does not become more accurate with time. And if there is some gain to the user if they provide incorrect results, this will bias the new training data.

Crowdsourcing the Annotations

This method is particularly useful if you have a large amount of unlabelled data and it's not possible to collect labels from users through the normal usage of a product. Many problems in the NLP and computer vision domains fall into this category: it's easy to collect a large corpus of images, but the data isn't labelled for the specific use case of your machine learning model. For example, if you want to train an image classification model that classifies cellphone images as either documents or non-documents, you might have your users take many photos but not supply your labels.

In this case, a large pool of unlabelled data is usually collected, which is then passed to a crowdsourcing platform, such as AWS Mechanical Turk or Figure Eight. Human annotators are then paid (usually a small amount) to label the data. This is most suitable for tasks that do not require special training.

With this method, it's necessary to control for varying quality of labelling, and the annotation tool is usually set up so that multiple people label the same data example. The Google PAIR guide gives some excellent, detailed suggestions for designing annotation tools, but the key thing to consider is that the incentives of the annotators need to be aligned with the model outcomes. The main advantage of this method is that it's possible to be extremely specific about the new data that's created so it can exactly fit the needs of a complex model.

However, there are a number of drawbacks to this approach—for example, it may not be suitable for private or sensitive data. Also, be careful to ensure that there is a diverse pool of raters that reflect the users of your product and society as a whole. There can be a high cost to this approach too, which may not scale to a large number of users.

Expert Annotations

Expert annotations are set up similar to crowdsourcing, but with carefully chosen annotators. This could be you (the person building the pipeline), using an annotation tool such as Prodigy for text data. Or it may be a domain expert—for example, if you are training an image classifier on medical images. This method is especially suitable for the following situations:

- The data requires some specialist knowledge to annotate.
- The data is private or sensitive in some way.
- Only a small number of labels are required (e.g., transfer learning or semi-supervised learning).
- Mistakes in annotations have high, real-world consequences for people.

This method allows the collection of high-quality feedback, but it is expensive, manual, and doesn't scale well.

Producing Feedback Automatically

In some machine learning pipelines, no human is required for feedback collection. The model makes a prediction, and some future event happens that tells us whether the model was correct or not. In this case, new training data is collected automatically by the system. While this does not involve any separate infrastructure to collect the feedback, it still requires care: unexpected things can happen because the presence of the predictions can perturb the system. The preceding example from Stripe illustrates this well: the model influences its own future training data.[4]

How to Track Feedback Loops

Once you've decided which type of feedback loop best fits your business and your type of model, it's time to incorporate it into your machine learning pipeline. This is where model validation, as we discussed in Chapter 7, becomes absolutely essential: new data will propagate through the system, and, as it does so, it must not cause the system performance to decline against the metrics you are tracking.

The key concept here is that every prediction should receive a tracking ID, as shown in Figure 13-3. This can be implemented with some kind of prediction register in which each prediction is stored along with a tracking ID. The prediction and the ID are passed to the application, and then the prediction is shown to the user. If the user gives feedback, the process continues.

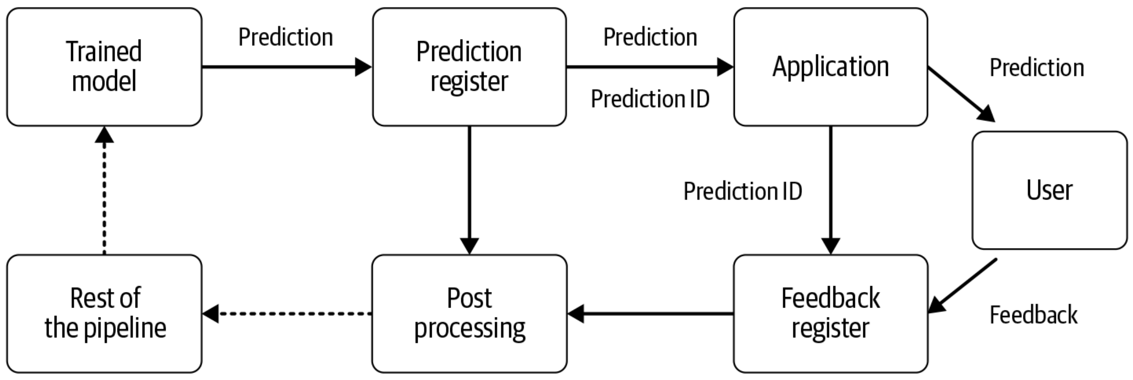

Figure 13-3. Tracking feedback

When feedback is collected, it is stored in a feedback register along with that prediction's tracking ID. A data processing step joins the feedback with the original prediction. This allows you to track the feedback through the data and model validation steps so that you know which feedback is powering the new model version.

Tracking Explicit Feedback

If the system is collecting explicit feedback, as described previously, there are two possibilities for how to track it:

Binary feedback

In most situations, only the feedback that tells you that a prediction is correct can give you new training data with an associated tracking ID. For example, in a multiclass classification system, user feedback only tells you whether the predicted class is correct or not. If the predicted class is marked as incorrect, you don't know which of the other classes it should be. If the predicted class is correct, the pairing of the data plus the prediction form a new training example. A binary classification problem is the only situation where you can use the feedback that a prediction is incorrect. In this case, this feedback tells us that the example belongs to the negative class.

Reclassification or correction

When a user gives the model a correct answer, the pairing of the input data plus the new classification form a new training example and should receive a tracking ID.

Tracking Implicit Feedback

Implicit feedback generates binary feedback. If a recommendation system suggests a product and the user clicks on that product, the pairing of the product and the user data form a new training example and receive a tracking ID. However, if the user does not click on a product, this doesn't mean that the recommendation was bad. In this situation, it may be necessary to wait for many pieces of binary feedback for each product that is recommended before retraining the model.

Summary

Feedback loops turn a machine learning pipeline into a cycle and help it grow and improve itself. It's essential to incorporate new data into the machine learning pipeline to prevent the model from getting stale and having its accuracy drop. Make sure to choose the feedback method that is most aligned with your type of model and its success metrics.

Feedback loops need careful monitoring. Once you start collecting new data, it's very easy to violate one of the most fundamental assumptions of many machine learning algorithms: that your training and validation data are drawn from the same distribution. Ideally, both your training and validation data will be representative of the real world that you model, but in practice, this is never the case. So as you collect new data, it's important to generate new validation datasets as well as training datasets.

Feedback loops require you to work closely with the designers, developers, and UX experts involved in the product. They need to build the systems that will capture the data and improve the model. It's important that you work with them to connect the feedback to improvements users will see and to set expectations for when the feedback will change the product. This effort will help keep users invested in giving feedback.

One note of caution is that feedback loops can reinforce any harmful bias or unfairness in the initial model. Never forget that there can be a person on the end of this process! Consider offering users a method to give feedback that a model has caused harm to someone so that it's easy for them to flag situations that should be fixed immediately. This will need far more details than a one to five star rating.

Once your feedback loop is set up and you are able to track the model's predictions and responses to predictions, you have all the pieces of the pipeline.

[1] For more details, see Google's PAIR manual.

[2] Feedback should be easy to collect and give actionable results.

[3] See Michael Manapat's talk, "Counterfactual Evaluation of Machine Learning Models," (Presentation, PyData Seattle 2015), *https://oreil.ly/rGCHo*.

[4] More on this can be found in D. Sculley et al.'s "Hidden Technical Debt in Machine Learning Systems," in *Advances in Neural Information Processing Systems 28* (NIPS, 2015), *https://oreil.ly/eUyZM*.

Support Sign Out