**Deploying Your Machine Learning Model Is Just the Beginning**

How to turn ML models into useful business actions: a primer on MLOps



Photo by [Katherine McCormack](https://unsplash.com/@kathymack?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

Like many people starting out in ML, one of the first problems that I got my hands on was the Titanic dataset. The task in this problem is to ‘predict’ whether a passenger survived the Titanic disaster or not, given features like ticket class, cabin location, gender, age, and so on. While it’s a fun problem to solve, it’s practically useless, for obvious reasons. No one actually needs a Titanic classifier. There’s no action that the model could take in the real world.

In the real world, actions matter. A model that doesn’t actually do anything is no better than not having a model at all. Yet, the subtleties involved in turning ML models into useful business actions are not typically covered in ML research.

In this post, we’ll go into:

* the spectrum of actions that your model can take in the real world,
* how to monitor your model’s operating point,
* why you can guarantee neither precision nor recall in production (and what to do about it), and
* what to consider if you have human annotators in the loop.

Let’s get started.

**What actions can your ML system take?**

Broadly speaking, there are 3 types of actions your model can take:

* *Soft actions*: sideline something for human investigation, but other than that do nothing. Let the human decide and take the final action.
* *Hard actions*: automatically do something (e.g. move an Email to spam folder, or cancel a credit card transaction).
* No action / pass action. Do nothing.

A particular ML application may require several actions in production, which can be taken depending on the ML model score. For example, in a credit card fraud detection system, you can auto-cancel orders with the highest model scores (hard action), pass orders with the lowest model scores (no action), and send everything in between for human investigation (soft action). In an email spam detection system, you can automatically move emails with the highest model scores to the spam folder, and flag emails with intermediate model scores as potential spam, but still keep them in the user’s main mailbox.

Lastly, what about ranking models, which power applications such as search, feeds, and ads? Here the action is, well, ranking: displaying the items in the best possible order.

**[Learning to rank: A primer](https://towardsdatascience.com/learning-to-rank-a-primer-40d2ff9960af" \t "_blank)**

[The algorithms that power search engines, recommender systems, social media feeds and ads](https://towardsdatascience.com/learning-to-rank-a-primer-40d2ff9960af" \t "_blank)

[towardsdatascience.com](https://towardsdatascience.com/learning-to-rank-a-primer-40d2ff9960af" \t "_blank)

**How to monitor precision and recall**

ML courses teach you the importance of precision (the ratio of positive predictions that are correct) and recall (the ratio of all positives that our model is catching). It’s a trade off, and you can sacrifice one for the other. The exact combination of precision and recall at which your system operates is called the *operating point*. But how do we actually measure the operating point in production?

Consider the problem of measuring precision first. If your system takes soft actions, then precision is relatively easy to track. Simply keep track of the human decisions that are made on the sidelined items. If your system takes hard actions, you can set up a *control group*: for a fraction of the volume, let a human decide, instead of your model. Then the precision in the control group is an estimate of the precision of your model.

Measuring recall can be a lot trickier. In order to do that, by definition, you’ll need to audit your negative predictions. This is tricky because typical use-cases may have imbalanced classes. For example, let’s say you build a fraud detection system that takes pass action on 1M orders per day, and your recall is expected to be 99.9%. That means that, on average, you’d need to audit at least 1K negatives per day just to find just one false negative, and even more for that number to be statistically meaningful.

Recall audits with simple random sampling are therefore highly impractical. A better approach is *importance sampling*, in which we sample negatives not randomly, but based on their model scores. The key idea is to sample more heavily from datapoints with high model scores because that’s where we expect most of our false negatives to be. In a [blog post](https://arun.chagantys.org/technical/2019/06/09/estimating-recall.html), Google researcher Arun Chaganty showed how he used importance sampling to reduce the cost of measuring recall by about a factor of three.

**Why you can guarantee neither precision nor recall in production (and what to do about it)**

No matter how you tune your operating point on your offline evaluation set, in practice you can’t guarantee that your system will meet a certain amount of precision or recall in production. That’s because of data drift: the data distribution in production will be always be slightly different from that in your offline test. The amount of data drift depends on the problem domain, and is especially severe in adversarial domains such as fraud and abuse.

What to do about this problem? One solution is to add a buffer: for example, if the business goal is a precision of 95%, you can try tuning your model to an operating point of 96–97% on the offline evaluation set in order to account for the expected model degradation from data drift.

It’s also important for ML teams to set the right expectations with business stakeholders. For example, I’ve seen a case the ML team’s contract with the business stakeholders was to guarantee X% recall on known (historic) data. This is a great contract because it doesn’t try to make guarantees about something that the ML team doesn’t have complete control over: the actual recall in production depends on the amount of data drift, and is therefore not predictable.

**How the dynamics change with humans in the loop**

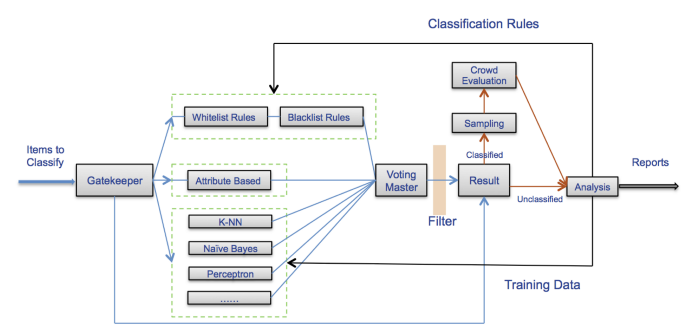
The operational aspects of your ML application change dramatically once you introduce sideline actions with human investigators in the loop.

First, some terminology:

* the *backlog* is the amount of items that you have sidelined for human investigation.
* The *queue rate* is ratio of all orders that you’re sidelining.
* The *capacity* measures how much you can at most queue in a given time period. For example, if you hire 100 human annotators working 8 hours per day, and a single annotation takes 5 minutes, then you can queue around 10K items per day, and that’s your system’s capacity.

The fundamental law of backlog management is this: if your queue rate is higher than your capacity, your backlog increases. If your queue rate is lower than your capacity, your backlog decreases. Ideally, you’ll want a stable backlog where as much is coming in as going out, just like a bathtub where as much water is flowing in as draining at any time.

In practice however, the queue rate can be unpredictable because of data drift. For example, in fraud detection a new fraud attack may cause your queue rate to spike. Similarly, in product classification there can be sudden spikes in products from a new vendor. For this reason, you’ll want a certain amount of *elasticity* in your labeling workforce, i.e. being able to quickly increase or decrease the size of the workforce as needed.



Walmart’s product classification pipeline leverages both human experts and crowd workers. ([source](https://pages.cs.wisc.edu/~anhai/papers/chimera-vldb14.pdf))

An interesting solution to this problem is presented in a paper from [Walmart Labs](https://pages.cs.wisc.edu/~anhai/papers/chimera-vldb14.pdf) (Sun et al 2014): their system uses a combination of both expert human analysts and crowd workers for product classification. While the crowd workers are less accurate than the human experts, they’re also an extremely elastic workforce. This elasticity is especially helpful in dealing with “sudden burst of hundreds of thousands of items that come in”. In addition, human experts do regular audits of the crowd-generated labels to ensure their quality.

**Conclusion: deploying your model is just the beginning**

I hope to have made at least one thing clear: after deploying your ML model, the work is far from done. I’d argue that the work is really just getting started. To summarize:

* an ML model is only useful if it takes actions. Those can be hard actions (e.g. canceling a credit card transaction) or soft actions (e.g. flagging it for manual inspection).
* it is critical to monitor both precision and recall in order to track the health of the overall system. Recall audits are expensive for problems with high class imbalance, but can be made cheaper with importance sampling.
* in practice, you can guarantee neither precision nor recall in production because of data drift. A good contract between ML teams and stakeholders may therefore be to guarantee a certain amount of recall/precision on known data.
* when human labelers are in the loop, then, in addition to precision and recall, you’ll need to keep track of the queue rate as well. If the queue rate spikes, you’ll need additional human labelers to keep up with the extra volume.

Lastly, despite the [focus in ML research on model performance](https://medium.com/towards-data-science/is-my-model-really-better-560e729f81d2), it should be clear that not all improvements to an ML production system need to be model improvements. For example, suppose you can find a way to cut investigation time in half by building a better UI for your human labelers. Such an improvement would allow us to either save on labeling cost, or improve overall recall or precision by taking advantage of the new capacity. Either way, the system would have improved dramatically without any improvements to the ML model.