James Stanier Follow
VP Engineering @brandwatch. Hovering somewhere around leadership, management, technology and personal development.
Jun 14 · 12 min read

How do data science projects work?

A primer for managers, stakeholders and those that are interested

No, not that kind of science. But sort of.

According to the LinkedIn 2017 U.S. Emerging Jobs Report, data science roles on the social network have grown over 650% since 2012. The same report notes that there are 9.8 times more machine learning engineers working today than there were 5 years ago.

But, given that data science is a nascent field, do we really know how to run projects in a way that allows data scientists themselves to have the space required to experiment and for management and stakeholders to be satisfied with their progress? After all, despite 43 years passing since the original publication of The Mythical Man Month, and 17 years since the Agile Manifesto was coined in a ski lodge in Utah, you could sometimes argue that we've never learned how to deliver software!

Given the inevitability that data science projects will become ever more part of the software industry as a whole, and that more managers will be held accountable for them, and that more stakeholders will be expected to follow along and give feedback, we should all understand how these projects progress.

Here's what *I've* learned.

But first, to set the scene, and because it's fascinating, let's have a very gentle introduction to deep learning, which is one of many techniques used in the field of data science. It uncovers some of the intricacies of doing data science projects.

Scaling deep learning

A number of weeks ago, I had my attention drawn to an excellent research paper via Adrian Colyer's Morning Paper newsletter. The paper is an exploration into understanding how we can improve the state of the art in deep learning, and whether we can make progress more *predictable*.

Without going into a lot of technical detail, deep learning—the new, fashionable term for building neural networks with hidden layers to solve problems—can tend to be more of an art than a science. We don't know a huge amount about exactly why they work so well for particular problems, and years of designing them hasn't yielded bulletproof design principles or patterns. Clearly, this isn't promising for the predictability of projects. After all, the industry always wants launch dates!

If you aren't familiar with neural networks, then they can be described as a mathematical model inspired by how neurons work in the human brain. Each neuron takes an input, and depending on some condition, gives an output. Neural networks are software representations of chains of neurons. Deep learning—specifically the "deep" part—means that there multiple layers of neurons. More specifically, deep learning means that there are "hidden" layers of neurons: ones that exist in the chain between the input and output layer. These deep learning networks have been used successfully for making computers do "intelligent" tasks such as voice recognition and image recognition.

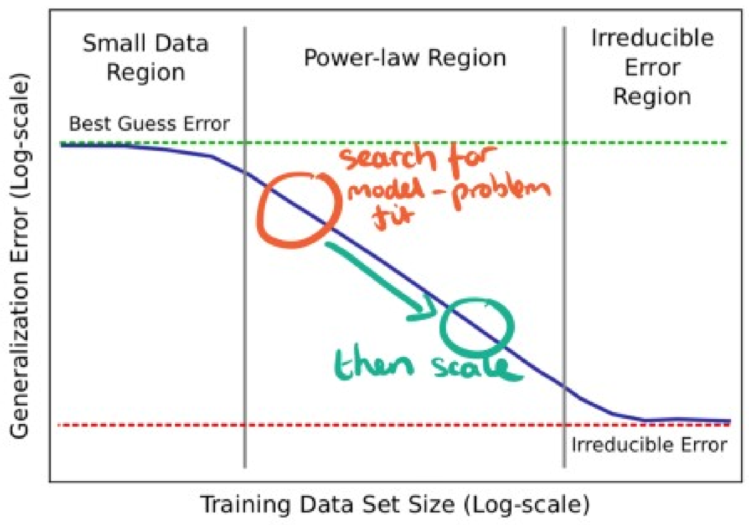
Building neural networks in software isn't like regular programming, where a human writes out the specific instructions of what happens when a user clicks a button, or assembles exact queries to a database to retrieve data. Instead, you specify how the network looks: the number of neurons and the number of layers and how they all connect to one another, which data inputs make them activate, their individual activation functions and the algorithm used for training them. That's called defining the architecture of the network. Some networks have had millions of neurons and billions of connections between them.

Once you have an architecture designed, creating a deep learning classifier requires *training*. For example, if you wanted to use a neural network to determine if pictures had a cat in them, you would expose the network to lots—often many thousands—of pictures of cats. Special algorithms tune the neurons to understand the commonalities in the cat images, meaning that when exposed to a picture of a new cat, it can classify it as such.

Creating one of these classifiers roughly unfolds in these stages:

1. Defining the architecture of the network.
2. Evaluating it with a small about of data to see whether it works.
3. Iteratively training and adjusting the network based on the amount of error seen.
4. Stopping training when the network performs well enough, or when it isn't improving any more.

The paper contained the following diagram to show how increasing the amount of training data (x-axis) reduces the amount of error in a given deep learning model (y-axis).



Deep learning with little data is rarely better than random. But after finding a suitable architecture, throwing more data at it makes it improve, to a point.

The *small data region* involves prototyping approaches on small amounts of data, where often results can be no better than random. Once a design yields some results, more data is gathered and used to train the model to see whether it truly fits the problem; this is represented by the orange circle annotation. If it is improving, then as much data is fed into the network as possible while it improves (the *power-law region*). We continue until it cannot get any better (the *irreducible error region*).

If this all sounds a bit strange, then consider the similarities with doing a start-up: typically the founders keep iterating on MVPs and pivoting until the product-market fit is found and customers start signing up, and then decide to stick, invest and scale it into a larger business.

I hope that gives some idea of how deep learning projects work. As it turns out, there are many similarities in the steps taken in most data science projects. As a manager or a stakeholder, you'll need to understand when projects are moving through these phases to have a greater appreciation for the work going on.

Managing data science projects

If you're used to running or observing software teams who aren't doing data science, then you'll notice that the above sequence of steps sounds fairly peculiar and unpredictable when compared writing your typical CRUD applications, and you'd be right! If your business expects the same kind of predictability and time commitments for data science that they do for their regular software projects, then there may be some uncomfortable conversations required: they work *very* differently.

The approach to training deep learning networks above can be generalized to represent data science projects more broadly, regardless of the exact technique, whether it be NLP, regression techniques or statistics.
Data science projects iterates through multiple phases, many of which can result in failure, and many which require financial decisions for the business such as whether to invest in more training data or more compute power.

If you want data science success, then it's *incredibly important that managers and stakeholders understand how projects typically work*. This ensures there is a mutual understanding and appreciation of risks and progress, and gives data scientists the trust and space they need to experiment.

Common iterative areas are as follows:

1. **Defining the problem:** Typically there is a *business problem* to solve that needs formulating into a *data science problem*. Is the problem actually solvable? What data are available and what features do we need to model? How do we know that we have succeeded? Which trade-offs, notably in precision and recall are acceptable to the user?
2. **Finding a model:** Assuming we think we can solve the problem, what sort of techniques may be likely to work? How can we prototype them and get some initial results that let us be confident enough to proceed?
3. **Training the model:** How and where do we get more data from?
4. **Application to real data:** Now that our model is trained and is giving acceptable results, how do we apply it to real data to prove that it works?
5. **Production:** Now that we have a successful model, how does it get moved into production and by whom?
6. **Maintenance:** Will this model degrade over time and, if so, how much? Does it need retraining at regular intervals in the future?

Steps 1–4 may result in failure and a decision to go back a number of steps, or even to not continue with the project. These steps are *scientific processes*. In steps 5–6, we should have a model that we would be confident delivering, whether that is to a customer, or building it into software. This makes them *engineering processes* requiring a different set of skills, and often staff.

1. Defining the problem

Contrary to belief, this can be the riskiest, most difficult and time consuming part of the entire process. Business problems aren't often easily translatable into a scientific problem that can be immediately worked on. Stakeholders will be wanting to deliver a feature to users, or they may be wanting to gain an insight into some data. But the team will have to start with first principles and build the problem from the ground up.

Firstly, is the problem actually solvable? It may be the case that it isn't —i.e. it's intractable by definition, or requires data that don't exist. It may be the case that a model could be built, but the insight is so subjective that many people will think that it is incorrect. If the data exist, is it clear which parts of those data are required for modelling, and what the features are?

Most importantly, how will the team know when they have succeeded? Is there a clear right answer that will allow them to prove that the model works? Or will it require lots of user testing, and if so, with who? Additionally, what trade-offs are acceptable? Does it need to work most of the time, or *all* of the time?

2. Finding a model

Given that there is a clear definition of a problem, such as whether it's possible to predict whether users are about to leave your website, or whether you can classify images of credit cards, the project begins with finding a model. These stages are typically run in a time box, depending on the size and difficulty of the problem. Therefore the first question that the business has to answer is *how long are we willing to spend seeing if this might be possible?*

Secondly, in order to find a model, you'll need data. Specifically, two types of data: training data and test data. You may already have the data you need. In the first example project above, it may be a case of taking a sample of user logs from your website. However, you might not have any data at all: in the case of the second example project, where are you going to get images of credit cards from?

So the next questions to ask are *can we get any data to test with?* This is then followed by *can we annotate the data easily?* Going back to pictures of cats, if a dataset doesn't already exist, humans are going to have to look at those pictures and say whether they are a cat or not. If so, you will probably want to crowdsource the annotation using platforms like Mechanical Turk or Figure Eight. This costs time and money and requires a clear definition of the questions to be asked. Sometimes it is simple such as "is this a cat?" Sometimes is highly subjective, such as "does this text convey disappointment?"

Assuming the business is happy with the time and money investment, then this phase will run until a suitable model is found, or until the time runs out and we admit failure. This phase of the project will typically have staff running small experiments, and spending a few hundred dollars on data. The end of the time box is a great opportunity for a demo in front of stakeholders, regardless of whether there has been a success or not: there are always learnings to share.

3. Training the model

If the first phase has been a success, it's time to train the model. Like the first phase, this centers around *time* and *money*. In broad terms, the more training data that are available, the better the model will get. You will need to discuss, again, where to get the data, how to get them annotated, and how much you are willing to spend on that task. It will typically be much more than in the first phase—maybe even thousands of dollars if you need human annotation to be done.

Additionally, depending on the scale of the problem, training the model may require more compute power than is available on local machines. For non-trivial training tasks you will want to utilize fast machines, often from a cloud provider. Even using spot instances can be pricey depending on the task, so upfront estimates are essential to avoid an expensive surprise.

Assuming the budget is acceptable, you'll also want to have a conversation about when we'll know the model is performing acceptably. Is there a precision and recall goal we are aiming for? Or are we going to commit a certain amount of time and money and then reassess? Given the variables above, this phase will, hopefully, produce a model that works well against test data. Demonstrating it against real data in the problem space is a great way to conclude this phase.

4. Application of the model to real data

Precision is the accuracy of your model on the data that it has classified. Recall is the amount of data that it has correctly classified out of all possible correct classifications. (Interested parties might find it fun to read a thorough definition.) There will always be errors in any model, but tuning it to balance precision and recall can greatly improve the model's effectiveness: high recall typically lowers precision, which means your users may see more errors. Is that acceptable? It could be for classifiers diagnosing illness where you'd rather be safe than sorry, but it could be extremely irritating for users of text analysis software.

At this point, you'll have a model that seems to work well at the task at hand. Before giving it the green light towards production, you'll want to test it on real data, as precision and recall figures alone cannot be trusted to determine whether the model is of acceptable quality.

A good approach is to have the team produce multiple versions of the same model with different parameter tunings representing different precision/recall balances. These can then be applied to real data and shown to your stakeholders for their feedback. This can help them understand the implications and make an informed choice as to which models and tunings acceptably solve their problem.

5. Production

If your team has made it to this phase, then it's looking very likely that you'll be getting that feature delivered. But they're not done yet. Unless you're extremely lucky, your data scientists will not be experts at production engineering, so it's at this point that they'll partner up with other engineers to move the project forward.

Key considerations here include how to technically integrate the classifier into the production system, how to store the models, and the speed of classification which determines how the code around the models will need to be architected: perhaps it needs wrapping in an API because many parts of the system will need to use it. Maybe due to data volumes and speed of classification it will require tens or hundreds of instances.

Work at this point becomes easier to estimate. It fits more naturally into how your feature teams deliver their work. You can start giving more concrete deadlines for the feature becoming available, and assuming all goes well, you'll be able to ship it.

Awesome. But is the project done? Not exactly…

6. Maintenance

Like regular software, models that you build will also need maintenance with time. Depending on the type of data you are processing, especially if it is topical data such as social media feeds, then inputs will change: consider how widespread the use of emojis are today compared with five years ago. Consider the names of popular video games now compared to last year. Models won't know about these if they are abandoned once shipped.

If the input data does evolve and change rapidly over time, then the team will need to revisit it in the future to analyse it against new data. Documentation on how the team built, chose, and trained the model is essential as inevitably everyone will have forgotten in the future when it's time to check how it is performing.

Whereas maintenance of the production system is the duty the production engineers, the maintenance of the models is the duty of the data scientists. You'll need to find an ongoing balance of creation of new models and maintenance of the ones that you already have in production.

Cycle complete

And that's it: the rough life cycle of a data science project. In many ways, they are harder to manage than traditional software projects.

Getting something into production involves much more chance of failure, many more unknowns, financial implications for data and compute, and cross-collaboration between disciplines. And given that people understand less about how this sort of work is done, and the hype in the industry about the promise of AI to solve all of our worldly problems, managing expectations is even more challenging. Not to mention the ethical implications of data science as a whole, but that's for another article…

. . .

Thank you to my colleagues Alastair Lockie, Dan Chalmers, Hamish Morgan, Óskar Holm and Paul Siegel who gave invaluable input as I put together this article.

. . .

Hey, I'm James. 👋

I hope you enjoyed this article. If you did, why not check out some of my other articles, such as ***why trust can't be bought*** *and whether* ***optimism can ever be bad****.*

If you like what you read, follow me on Twitter @jstanier.