Agile Machine Learning with R

A workflow

Edwin Thoen

Contents

Chapter 1

My Workflow Evolution

Not even too long ago, when I was starting my career as a data scientist, I did not really have a workflow. Freshly graduated from an applied statistics master I entered the arena of Dutch business, employed by a small consulting firm. Neither the company I was with, nor the clients I was working for, nor myself had an understanding of what it meant to implement a statistical model or a machine learning method in the real world. Everybody was of course interested in this "Big Data" thing, so it did not take long before we I could start at clients, often without a clear idea what I was going to do. When we finally came to something that looked like a project, I plunged into it. Eager to deliver results quickly I loaded data extracts into R and started to apply all kinds of different models and algorithms on it. Findings ended up in the code comments of the R scripts, scripts that often grew to hundreds or even thousands of lines. To still have somekind of an overview I numbered the scripts serially, but this was about all the system I had. Soon I found myself amidst dozens of scripts and data exports of intermediate results that were no longer reproducible. The R session I was running *ad infinitum* was sometimes mistakenly closed, or it crashed (which was bound to happen as the memory used grew). I spent hours or even days to recreate the results when this happened. Deadlines were a nightmare, everything I had done so far had to be loaded, joined and compared at the last moment. More often than not, the model results appeared to be different from the indications in the notes I took earlier, with no idea if I was mistaken earlier, I was using the wrong data now, or some other source of error I was not aware of was introduced. Looking back at these times, I had no clue about the importance of a good workflow for doing larger data science projects. I was saved several times when plug was pulled from the project, due to other reasons. If I was expected to bring the results to production then, it would certainly been a painful demasqué.

I learned a great deal since these day, both from other people's insights and from my own experiences. Writing an R package that was shipped to CRAN

enforced me to understand the basics of software engineering. Not being able to reproduce crucial results forced me to start thinking about end-to-end research and model building, controlling all the steps along the way. Last year, for the first time, I joined a Scrum team (frontend, backend, ux designer, product owner, second data scientis) to create a machine learning model that we brought to production using the Agile principles. It was an inspiring experience from which I learned a great deal. My colleagues patiently explained the principles of Agile software development and together we discovered what did and did not work for machine learning.

1.1 What this Text is About

All these experiences culminated in the workflow that we are adhering to at work now and that I think is worthwhile sharing. It is heavily based on the principles of Agile software production, hence the title. We have explored which of the concepts from Agile did and did not work for data science and we got hands-on experience in working from these principles in an R project that actually got to production. This text is split into two parts. In the first we will look into the Agile philosophy and some of the methodologies that are closely related to it (chapters 2 and 3). Both will be related to the machine learning context, seeing what we can get from the philosophy (chapter 4) and what an Agile machine learning workflow might look like (chapter 5). The second part is hands on. We will explore how we can leverage the possibilities in the R software system to implement Agile machine learning.

1.2 Writing out Loud

Machine learning projects can greatly differ from each other. There are so many variables that make each project unique, there are many situations I have not experienced and there are necessarily many aspects of machine learning I am not aware of. In this writing I am relating my own experiences to the theory and best practises of Agile software development, to come up with a general workflow. This means that if I were to write this text in isolation I would be "overfitting" the workflow on about the dozen machine learning projects I have done. That is why I need your help. I hope many of you will read the first drafts of this book very critically and relate the content to their own experiences. If you find that parts are incomplete or plainly incorrect, please file an issue. Also, anyone who succesfully completes machine learning projects must have developed an effective workflow for themselves, even when it is not grounded in a widespread theory such as Agile. I am very interested in the best practised you have developed, even when they don't fit directly in the framework. File an issue with what you would like to add, if we can't fit it in the text we can always add it as an appendix or a discussion. This text is meant to be a living thing

with the objective of documenting a workflow that yields optimal reproducibility, highly reproducible results and quality code. The more people share their best practises, the closer we get to this objective. Please follow along on this journey and get involved! Finally, I am not a native of English so fixed typos and style improvements are greatly appreciated.

1.3 Intended Audience

The title of this text has four components: *Agile*, *machine learning*, *R*, and *workflow*. When you are interested in all four, you are obviously at the right place. This text is not for you if you hope to learn about different algorithms and statistical techniques to do machine learning, more knowledgeable people have written many books and articles on those topics. The workflow I present is completely separate from the algorithms you choose, as it focuses on code organisation and delivery. When you use python rather than R, you will still find this text valuable. The first part especially, which focuses on workflow only and is tool agnostic. Finally, this text is intended for everybody building a data science product in R. Whether a Shiny app or a complex statistical model, this text should be valuable for you as well. The iterative nature of Agile will make your process more effective and your stakeholders more involved. I contemplated calling it *Agile Data Science…* instead of *Agile Machine Learning…* to broaden the scope of the text, but frankly I have only done simple, internal-use Shiny dashboards and not too complex statistical models. Therefore, I will stick to what I know well and only give examples of machine learning. I hope you will have little trouble translating the concepts to you own situation if it is other than machine learning. You are most welcome to do suggestions if you think the text will benefit from expanding to other examples as well.

Chapter 2

Agile in a Nutshell

2.1 The Origins of Agile

Agile software development is not a specific methodogy, a process, or a prescription how to do your job. Rather it is a set of believes, a philosophy, that should guideline a team developing software in making the best possible decisions. Agile was not created out of thin air, of course, it was very much a reaction of the then ubiquitous approach called *Waterfall*. In *Waterfall* large software projects are divided into specific stages, each stage should be completed before the next stage can start. Subsequentially these are *problem analysis*, *designing the project*, *writing the software*, *testing it* and finally *implementation and maintenance*. Here you can find a clear and neutral introducion into the *Waterfall* approach. What stands out at *Waterfall* is delivering complete and faultless software. In order to do so, the centre of gravity of a Waterfall project is the documentation, in which every requirement and aspect of the software is written down meticulously. The underlying conviction is that the software is written faster and is of higher quality when all aspects of it is decided upon upfront.

Projects done with the Waterfall method have a major pitfall, however. They can take a long period of time to be completed. The combination of sequentiality and completeness might cause projects to last many years before they get delivered. Each time an error or incompleteness is found in one of the lower stages, the project moves back to the previous stage to fix it. Because of the long duration of the entire process of *Waterfall* it occured often that the end result was no longer a good fit for the market. This that once the product is finally completed, it is no longer a good fit for the market. The problem analysis might be done years ago and the problem has changed in the meantime. *Agile* has a radical different appoach to software design. Rather than doing an elaborate research and

2.2 The Manifesto

During the nineties several reactions to the cumbersome Waterfall came to being. A number of influential thinkers in the world of software development started thinking what an alternative to the malfunctioning practises could be. Alternative processes were suggested, such as scrum and Xtreme programming. Eventually, in 2001 a group of seventeen came together in Utah and drew up the *Manifesto for Agile Software Development*. Their just 68-word-long statement is:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

You might think "well that is rather vague", yes it is on purpose. It is not a process you should follow or a methodology that prescribes how you should approach software development. Rather, they are core values that guide the development team with the many choices it makes along the way. At every crossroads the option that is most in line with these values should be selected.

2.3 The Twelve Principles

The Manifesto was accompanied by a set of twelve principles that flow from the values. They are more applicable than the four values and are thereby the principle guidelines when making choices. They are (numbering added by me):

1) Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4) Business people and developers must work together daily throughout the project.

5) Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7) Working software is the primary measure of progress.

8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9) Continuous attention to technical excellence and good design enhances agility.

10) Simplicity–the art of maximizing the amount of work not done–is essential.

11) The best architectures, requirements, and designs emerge from self-organizing teams.

12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Chapter 3

Agile Methods

In this chapters we are exploring the methods that promote working according to the Agile principles, at least Scrum and Kanban.

Chapter 4

Agile Machine Learning

4.1 Machine Learning's Waterfall

As discussed in the previous chapter, Agile is a response to Waterfall methodology that was widely adopted in the eighties and nineties. Many projects following this methodology failed because the long duration of these projects. The world had moved on while the process-heavy steps were completed and either the plug was pulled before the product was finished or the finished product had limited value because it was a misfit to the changed world. In machine learning, as far as I am aware of, there are no such formal methodologies that are followed by many practioners. However, there are ample testimonies of projects that never reached production and I think this is partially due to suboptimal workflow. Just like Waterfall, machine learning projects can take many months or even years before the results are productionised or the plug is pulled. The data scientist might want to optimise many asprects of the project to give the best predictions possible, before sharing the results with stakeholders. The code might be poorly organised, leading to a lot of time lost merging differenct scripts. Or there might be unclarities on what to predict in the first place, due to lack of communication between stakeholders, business people and data scientist. Whatever the reason, adhering to the principles of Agile can get you more productive and efficient. Here we take the time to interpret the twelve principles in the machine learning context.

4.2 The Twelve Principle in the Machine Learning Context

1) Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Just as Waterfall prescribes a complete and fault-free product delivered at once, data scientists might be inclined to only release a machine learning model to production once they are confident its predictions are spot on. This principle is a revolutionary break from Waterfall, you should not wait with releasing software until its perfect, instead get it out in the open when it is just good enough. A common term used for this is the MVP (*Minimal Viable Product*). After the MVP is released it is closely monitored how users are interacting with it and where the biggest room for improvement is. The biggest possible improvement is then tackled first and a new version is released. This cycle of release, monitor, improve, release is repeated many times, such that the product gets better and better. There is no clear definition of done, instead there is debate if the software can be further improved and if the required investments are worth the effort.

The machine learning equivalent to this would be a *Minimal Viable Model*, a model that is just good enough to put into action. This might be scary and counterintuitive to the high standards you have for yourself, but it is preferable over long optimisation before releasing for at least the following reasons:

- *It will keep stakeholders excited.* Managers and users of the model who commisioned the machine learning project are impatient to see results. As the projects drags on without any output they are likely to loose interest and confidence the project will end well. Eventually they might pull the plug or put it on hold before anything reached production. If they can interact with the results soon, even if it is imperfect will give a totally different dynamic.
- *You will fail fast.* There is a wide array of reasons a machine learning project might fail, such as; the problem appears not be translateable into a model in the first place, the data is not of the quality needed, or the relationship between the features and target. The sooner you implement the model the sooner lurking problems suface.
- *You will get feedback sooner.* This is the main reason Agile wants to implement quickly and then iterate. Lets say you build a churn model which the sales department uses for customer retention. As soon as they start acting on your MVM they find out that the interval in which you predict is too short, many customers already canceled their subscription. Instead of further optimising this model, you focus on predicting a longer time ahead.

What a MVM looks like is project-dependent of course, but in many cases it would probably make sense to define it a regular statistical measure. The machine learning model might be replacing a business rule that has been in place for many years, the MVM is then ready as soon as the model outperforms the business rule. Another way to build an MVM is by only releasing the model for a subset of your target audience. This might be a certain geographical area or users of a certain age. A model only implemented for a part of the population also makes a great MVM.

2) Welcome changing requirements, even late in development. Agile processes

harness change for the customer's competitive advantage.

This principle comes natural to machine learning, since the outcome of a project is at least partially dependent on the relationships discovered in data. The Waterfall approach in which every step of the project is planned would be appear hideous to even the biggest lover of process. Keep in mind that flexibility should not only be exercised towards assumptions of your data or the models and algorithms you use. Requirements can also be in the framing of the business problem or the way the model predictions are exposed. Whatever it is, don't be lazy and be prepared to steer in a different direction as soon as the situation requires.

3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Whereas the first principle is about the philosophy of early deployment and iteration, this one is about the frequency of deploying updates. The Scrum framework is really strict in the amount of time that can be spent until the next release. The team commits itself to making certain changes to the product in typically a two-week period. At the end of this period the improvements, small as they might be, are deployed. The Scrum mindset is not totally applicable for machine learning, as we will explore in the next chapter. It is typically not feasible to commit to a time interval for model improvent because we simply cannot commit to it. We are dependent on the relationships in the data and we don't know beforehand if the next road we enter is a dead-end or not. However, it is good to keep in mind that every improvement to the model should be deployed as soon as its ready. This creates momentum and excitement by customers, stakeholders, your teammates and yourself.

4) Business people and developers must work together daily throughout the project.

Machine learning cannot be done in isolation by a data scientist. Navigating through the data cannot be done without knowing about the underlying business process, often additional information is needed from business colleagues. Stakeholder management, keeping them informed about the progress and presenting them with important choices. Also, the customer might be business colleagues who should act upon the predictions. Not involving the business is a recipe for disaster for every machine learning process. Within the Scrum methodology the role of Product Owner is crucial for the allignment of the team with the business. Having such a representative is also very welcome for a machine learning project, he or she is than the translater between the modelling and the business. Keeping this person informed at all times is essential for decision making.

5) Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

This principle is the antithesis of Waterfall in optima forma. Instead of meticuously describing how the job should be done, just set the goals of the projects

and leave it up to the team how these goals should be attainted. Machine learning practioneers typically already enjoy this type of freedom for the sheer reason that stakeholders often don't really understand how the predictions are done. It can happen that business people get overly involved in the process, they can have a strong opinion on which targets should be used or how the target should be defined. Take their advice at heart but trust your instincts. If you feel a different approach will yield better results than rely on your expertise. You know about overfitting, multicollinearity, non-convergin algorithms and many other topics the business cannot grasp. Take the time to explain why you think a different approach is better (in lay men terms of course) and thank them for their input.

6) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

A machine learning project is rarely done end-to-end by one single person. Data might be made available by a dba, a backender might expose the model results within the website, the frontender builds the interface for interacting with the results, etc. If possible working with these people directly will speed up decision making and improve allignment. Communication by email or chat programs are often slow and lack the interaction. Make an effort to be in the same room with your direct colleagues, for at least a part of the project time.

7) Working software is the primary measure of progress.

As long it is not part of the modelling pipeline you have not reached any results yet. Only when the update to the predictions is fully implemented and the predictions are ready to be consumed by the business, there has been true improvement. All too often the reported improvement in accuracy in research scripts does not hold when it is implemented in the full model pipeline. Sometimes it has been done on just a subset of the data that was conveniently available. Or the new feature was tested in isolation and there is not yet a sense of multicollinearity. There is only one true measure of how well we are currently doing, and it is the pipeline. This implies that as long as there is not end-to-end pipeline in place, we cannot tell how well we are doing.

8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

The deadline way of doing a machine learning project; the stakeholders and the developers meet, they set to have a first version of the model ready at a set moment in the future. The sponsors forget about the project until right before the deadline, busy with meetings and memos. The devdeloper goes to work, having ample time before the deadline there are many things that can be explored. The result is an array of research scripts and intermediate results. Suddenly, as the deadline comes near, all this separate research has to come together. Pulling an all nigther the team is able to deliver a result, which is presented to the sponsors. The project is then continued, a new deadline is set, and the cycle starts over.

Don't - do - deadlines. They are a recipe for hastily created, nonreproducible results. They promote a workflow of taking it easy at first, stressing out when the dealine comes near and exhaustion after it. Instead set small goals that are attainable in a short timespan, update the model if its results are favorable and set a new small goal. This will result in better quality code, a better grip on the model results and a happier team. Moreover, it will result in a model that is constantly updated, which excites sponsors and users.

9) Continuous attention to technical excellence and good design enhances agility.

Machine learners can have much to learn from software engineers as it comes to standards and rigor. In machine learning much of the code that is used to produce the predictions is not shipped as part of the product. Cleaning of the train data, splitting in train and validation sets, running algorithms that produce the models, doing research on relationships in the data and many more steps are for the machine learning practioneer's eyes only. It is tempting to cut corners when you are the sole user of your own code. Why go to the trouble of writing unit tests and documentation for your functions, as soon it does not do what it is supposed to do you are right there to fix it.

At the moment of writing your code it is very obvious what is supposed to do and as you run the code against the data you are then working with it is straightforward to see if the program indeed does what it supposed to do. However, three months from know you completely forgot the reason you wrote that part and you have no clue why it failed against the refreshed data. You never work alone on a project, even if you are the only person working on it. Always consider future you as a separate person who you respect very much and you want to help to do its job as good as possible. The result of many parts of code of poor quality is that they don't click to make the bigger, complex system that large machnine learning projects are. Trying to create this system to produce final predictions is then a bit like ... Each time you fix one part another part comes tumbling down. To produce reproducible, reliable predictions it is essential you can completely trust the code you wrote.

10) Simplicity–the art of maximizing the amount of work not done–is essential.

A machine learning project's goal is often straightforward, predict y as best you can such that some business goals can be achieved. Other than software development there is not much to choose in which features should and should not be included in the final product (features as in characteristics, not as in predictors). The options how to arrive at predicting y, however, are abundant. The biggest challenge is often "what should I explore next?". Should we explore another database in which we might find new predictors or should we try a different model on the current predictors which involves some additional preprocessing of the data?

We can roughly estimate what the amount of work would be to explore both options, it is, however, very hard to predict what the amount of value is the

new part will add. A good rule of thumb is that when in doubt choose the option with the least unknown components. Choose an algorithm you know well over one you have never used in practise. Only tap into a new data source if you are convinced that the options on the current data base are exhausted. Machine learning is a field with rapid developments, it is often tempting to seize the opportunity to dive into a new technique or algorithm. Be critical before doing so, is there really no way to obtain similar results with something already familiar to you?

11) The best architectures, requirements, and designs emerge from self-organizing teams.

This is another principle that is a clear antidote to Waterfall. Instead meticulously plan every aspect of the project upfront, let the developers come up with the most important project designs as they go. It is impossible to foresee all the aspects of the software project before implementing it, so trying to come up with before writing code is a guarantee for going back and forth between the planning and implementation stages. Due to the iterative nature of building predictive models and the insecurity we have on the relationships in the data, this principle seems quite natural in the machine learning context.

12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

In the Scrum methodology a retrospective is done after two week sprint. The team discusses what went well in the past sprint and what deserves attention. Every development process has its inefficiencies, wether they are unclear communication or not havring the right priorities. Having to reflect on the process forces you to look critically at all aspects of the project. Inefficiencies can quickly become project features when they exist for a while, the sooner they are tackled the better.

Even when you are not in a team following an official methodology such as Scrum or Kanban, you do best in planning regular reflection meetings. Even when you are the only data scientist or even the only development of the team, you should also your technical issues here. Maybe you are wanting to refactor a certain part of the project for a while but are unsure if it is worth the time. Even though your business colleagues don't understand the technical aspect of the problem, they can still challenge you on the pros and cons of both sides.

We have reflected on the twelve priniciples with machine learning in mind. Some principles appear to be not too interesting for us, but many can be a great guide in delivering results quicker and with more joy and confidence. In the last chapter we briefly discussed the methodologies Scrum and Kanban. Now it is time to see what an Agile methodology for machine learning migh look like. We are going to explor this in the next chapter.

Chapter 5

A Methodology for Agile Machine Learning