But what is this “machine learning engineer” actually doing?

medium.com/@tomaszdudek/but-what-is-this-machine-learning-engineer-actually-doing-18464d5c699

May 27, 2018

Top highlight

Inspired by recent talk at O’reilly Data Show Podcast “What machine learning engineers need to know” I have decided to shed some light at this traction-gaining subject. While I am not that much of an expert, I think that my year and a half of experience in this exact position might be sufficient to write a short summary of what is this all about.

Machine learning, artificial intellience and similar buzzwords are currently on the top of VC interests. Startups are establishing left and right, promising to the world astounding *smart* future. The better ones get acquired by Big Five to furtherly increase their revenue. With the growth of this IT branch also comes the rising market demand for developers that can actually deliver promised artificial intelligence applications.

So, the natural choice is to hire a team of data scientists to develop your idea, right?

Well, not exactly.

Data scientists usually come from different background than software engineers. They do not necessarily make great programmers. In fact, they never intended to be them — for a data scientist coding is merely just a means to solve the current puzzle. And nothing more. Unlike software developers they do not treat the code as a form of art. Of course, their wisdom is invaluable but the range of skills required for being a successful data scientist is already broad (especially when the field frequently evolves with the new discoveries, making a significant portion of hard-earned knowledge obsolete on a daily basis). Too broad. **You can not expect from a person that is highly specialized in computer vision or prescriptive analysis to be also bread-and-butter programmer, productionizing the models and putting them in the heavy-scalable cloud environment.** While also maintaining a high quality, reusable code. Using functional programming. Or reactive. Or both.

On the other hand software engineers are quite reserved when it comes to machine learning. The whole concept is rather weird from their perspective, especially when the majority of so called models their data science team creates are short, hacky scripts with strange method calls and unreadable code in unfamiliar language. Where are all the design patterns? Where

is the clean code? Where is logging or monitoring? Why is the code not reusable? Shouldn't the code solving such a complex problem be more than two hundred lines long? It is a very ugly script that only one person can understand! Is it even programming anymore?

The merger

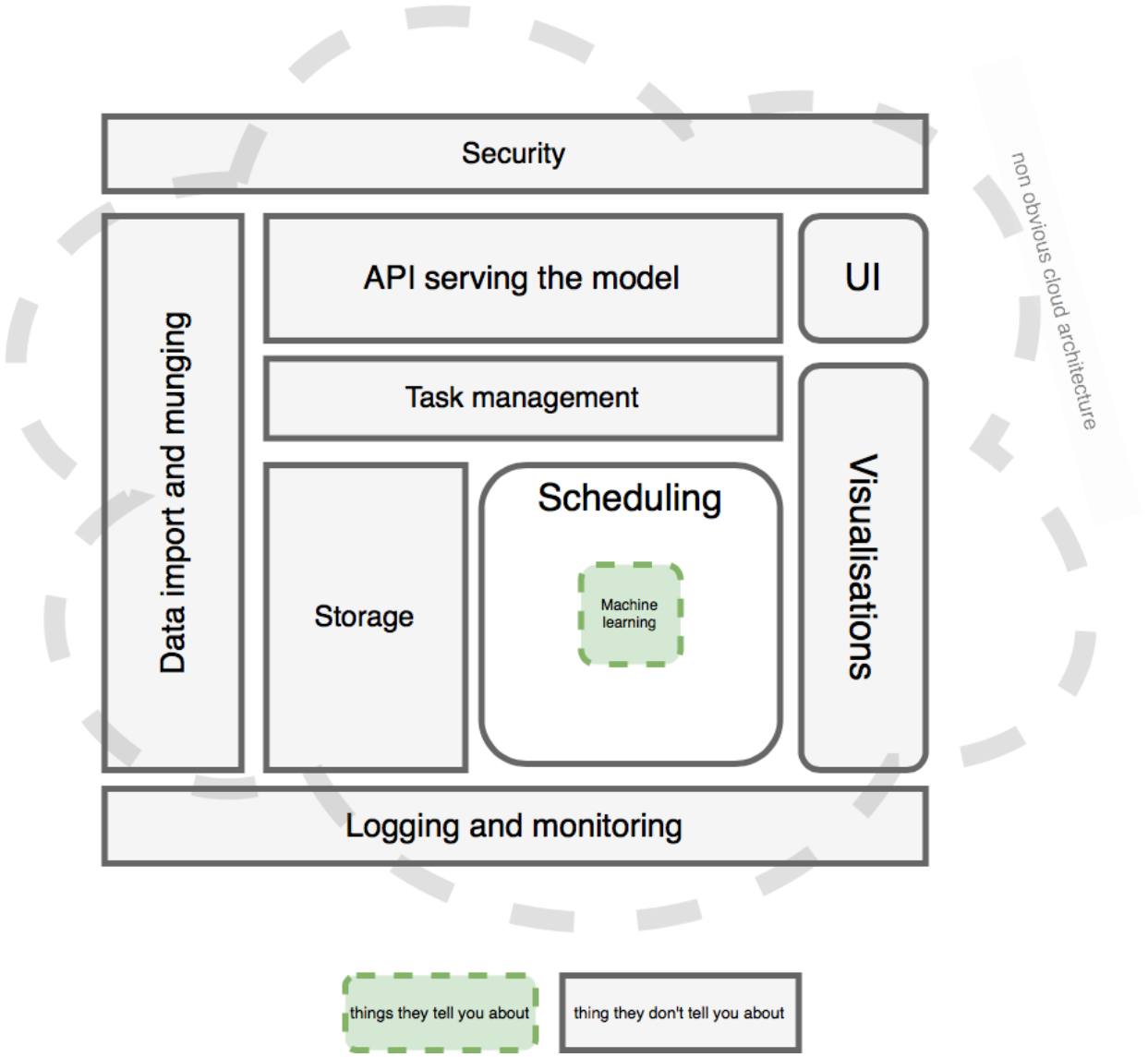
With this conflict arising, a need was born. A need for a person that would reunite two warring parties. One being fluent just enough in both fields to get the product up and running. Somebody taking data scientists' code and making it more effective and scalable. Introducing to them various programming rules and good practices. Abstracting away parts of code that might be used in future. Joining the results from potentially unrelated tasks to enhance the models performance even more. Explaining the reasons behind architectural ideas to the devops team. Sparing software developers from learning concepts way beyond their scopes of interests.

That need has been met with emerge of machine learning engineer role.

What is always missing from all the articles, tutorials and books concerning the ML is the production environment. It literally does not exist. Data is loaded from CSVs, models are created in Jupyter, ROC curves are drawn and voilà — your machine learning product is up and running. Time for another round of seed funding!

Hold on.

In reality **the majority of your code is not tied to machine learning. In fact, the code regarding it usually takes just a few percents of your entire codebase**! Your pretrained black box gives only the tiny JSON answer — there are thousands of lines of code required to act on that prediction. Or maybe all you get is a generated database table with insights. Again, an entire system needs to be built on top of it to make it useful! You have to get the data, transform and munge it, automate your jobs, present the insights somewhere to the end user. No matter how small the problem is, the amount of work to be done around the machine learning itself is tremendous , even if you bootstrap your project with technologies such as Apache Airflow or NiFi.



that was not on Coursera, was it?

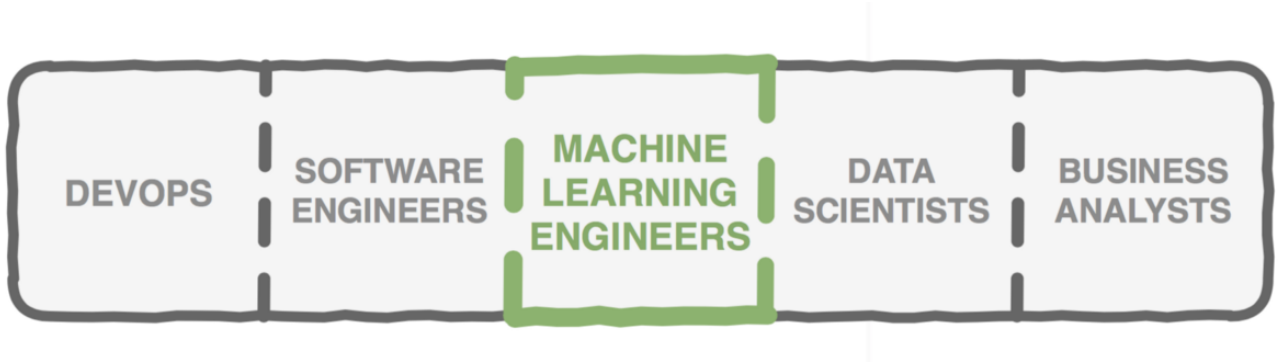
Yet, somebody has to glue all the “data science” and “software” parts together. Take the trained model and make it work on quality production environment. Schedule batch jobs recalculating insight tables. Serve model in real time and monitor its performance in the wild. And this is the exact area in which machine learning engineer shines.

When creating software, developers are naturally looking for all the possible outcomes in every part of application. **What you get from a data scientist is just a happy path that leads to model creation for the particular data at the particular moment in time**. Unless it is one-time specific analysis, the model will live for a long time after it gets productionized. And as the time flies, the bugs and all the edge cases are popping up(many of them were not even possible when the code was written). Suddenly a new unknown value shows up in one of the columns and the entire model start to perform way worse.

As a machine learning engineer you prepare your applications for such events. You provide the logging and monitoring pipelines not only around machine learning tasks but also inside them. You try to preserve all the information so it is possible to answer a very important questions: What is the cause of bad model's performance? Since when does it happen?

It is just another API

Because **you do not treat ML as magic,** you are aware of all other typical programming dangers that may arise when a machine learning job is executed. Database might refuse connection. GroupBy may blow up for large dataset. Memory or disk can be full. Combination of parameters specified by user might be illegal for certain algorithm. External service could respond with Timeout Exception instead of credentials. Column may not exist anymore. While nobody blinks an eye when such events take place in a safe lab environment on a daily basis, it is your responsibility to ensure they won't happen when the end product is actually delivered.



machine learning project roles

Your data science team is always full of ideas. You have to make sure that no technology is limiting them. As good and customizable as the current ML frameworks are, sooner or later your teammates will have an intriguing use case that is not achievable with any of them. Well, not with standard APIs. But when you dig into their internals, tweak them a little and mix in another library or two, you make it possible. **You abuse the frameworks and use them to their full potential.** That requires both extensive programming and machine learning knowledge, something that is quite unique to your role in the team.

And even when framework provides all you need programming wise, there still might be issues with the lack of computation power. Large neural networks take large amount of time to train. This precious time could be reduced by an order of magnitude if you used GPU frameworks running on powerful machines. You are the one to scout the possibilities, see the pros and cons of various cloud options and choose the most suited one.

You may also be responsible for picking other tools and ecosystems, always taking into consideration the whole the project lifecycle(not just the reckless research part) — e.g. Azure ML Workbench or IBM Watson might be great tools for bootstrapping the project and

conducting research but not necessarily meet all the requirements of your final version of the product when it comes to custom scheduling and monitoring.

You must stay up to date with the state of art technologies and constantly look for the places in which the overall product performance could be improved. Be it a battle-tested programming language, new technology in the cloud, smart scheduling or monitoring system — **by seeing your product on the bigger picture and knowing it well from both engineering, business and science sides, you are often the only person that has the opportunity to spot the potential area of improvement**.

This frequently means taking the working code and rewriting it entirely in another technology and language. Thankfully, as soon as you "get the grip" of what this fuzz is actually about and what steps are always taken in the process of learning and productionizing the models, you realize that most of these APIs do not differ at all. When you juggle between various frameworks, the vast majority of the whole process stays the same. You bring all the best software craftsmanship practices and quickly begin to build an abstraction over many repetitive tasks that data science team fails to automate and software development team is afraid to look at. A strong bridge between two worlds. A solid, robust foundation for a working software.

Untold cons

You can freely commune with all the hottest technologies in the field. Keras, pyTorch, TensorFlow, H2O, scikit-learn, Apache Spark — pick a name, you will probably use it. Apache Kafka! Pulsar! AWS! Every conference you attend speaks out loudly about your technology stack, as if it was The Chosen One. People look at you jealously, knowing that you are the guy using *all the coolest things*.

What is always conveniently omitted is the fact that those *cool things* also happen to be *not widely used* things. And when the technology is new all you are left with is poor documentation and a bunch of blog posts. What you see on the conferences and tech talks are just the happy green paths(similar to those Jupyter notebooks you get from your DS team). You know that it is not how software works. Many times, after hours of debugging Apache Spark internals, I questioned my will to pursue my programming career in machine learning. Wouldn't I be happier without all this? Was web development really *that* boring?

You are expected to know a lot of concepts, both in software development and data science. Most importantly, people want you to gain new knowledge very quickly. I learn a lot by taking somebody else's snippets, changing and breaking them and seeing what happens. Well, what if there are no snippets? **What if the stacktrace you get is pretty meaningless and googling the exception name leads only to the code at GitHub throwing it?**

not even a quarter of all the buzzwords available is shown

The learning and understanding curve is quite steep in some areas, especially when it comes to implementing ideas written in whitepapers. As cool(and sometimes exotic) as these tend to be, their form is always pretty *scientific* and just getting to understand them takes you a longer while. Then comes the coding part where you are totally on your own. Even though your application compiles fine and does not throw Runtime Exceptions all over the place it is **often unclear how to ensure that your implementation actually works properly**. And when it doesn't, you ponder whether your code has a bug, data is skewed or maybe the whole idea is just not applicable in your use case.

It is already challenging to keep up with the data science alone. When you throw in classical software development and cloud technologies as well, your brain might quickly become overwhelmed. You must organize your learning resources well and have a constant stream of

news flowing in. And also accept the fact that being the jack of all trades has its cons, because you will probably never have a very deep knowledge in any of your areas. The imposter syndrome kicks in very hard. And often.

But for some this is truly a dream job. Merging two so similar yet so different worlds.

A full stack developer of the data science.

Summing up

A person called machine learning engineer:

- asserts that all production tasks are working properly in terms of actual execution and scheduling
- abuses machine learning libraries to their extremes, often adding new functionalities
- ensures that data science code is maintainable, scalable and debuggable
- automates and abstracts away different repeatable routines that are present in most machine learning tasks
- brings the best software development practices to the data science team and helps them speed up their work
- chooses best operational architecture together with devops team
- looks constantly for performance improvement and decides which ML technologies will be used in production environment

This also happens to be my first blog post ever. I must say I really enjoyed writing and meanwhile had over a dozen ideas for more content. I will keep digging deeper into how a software developer can transition into full blown machine learning engineer. There are way too many pitfalls on the way. And too many resources to learn from!

Lastly, I promised my friend we would both start blogging around 2018.

Well, here it is, Joanna! It is your turn now.