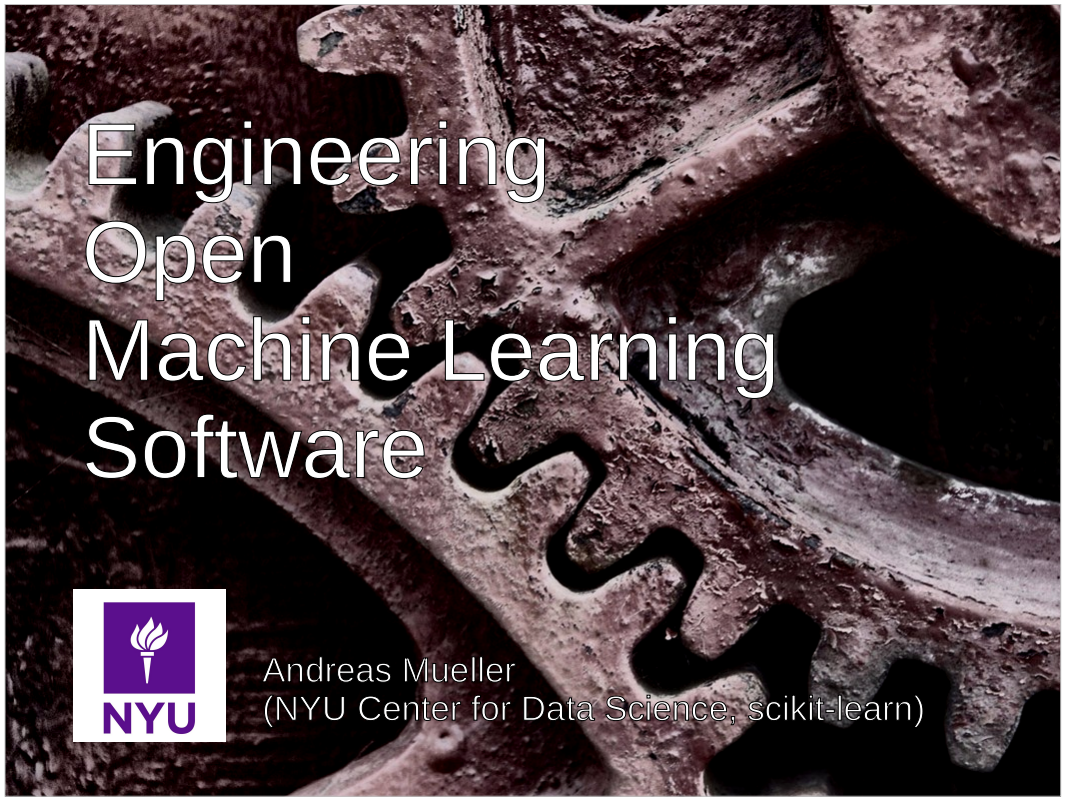
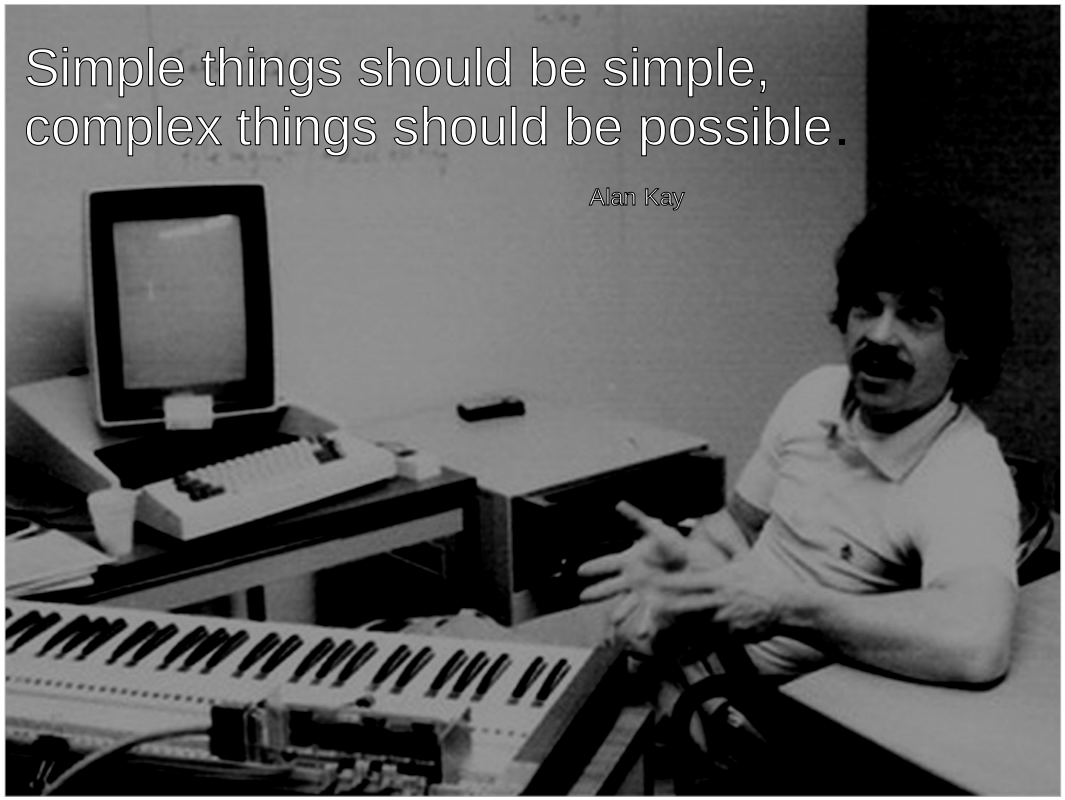


Hey everybody and welcome to my talk on engineering open machine learning software. I want to talk to you about some high-level principles that drive the development of the scikit-learn library. A lot of what I will be talking about is just a general philosophy of software engineering, but obviously with a machine learning twist. For those not familiar with scikit-learn, it’s an open source machine learning library for python, which is widely used in research and businesses.

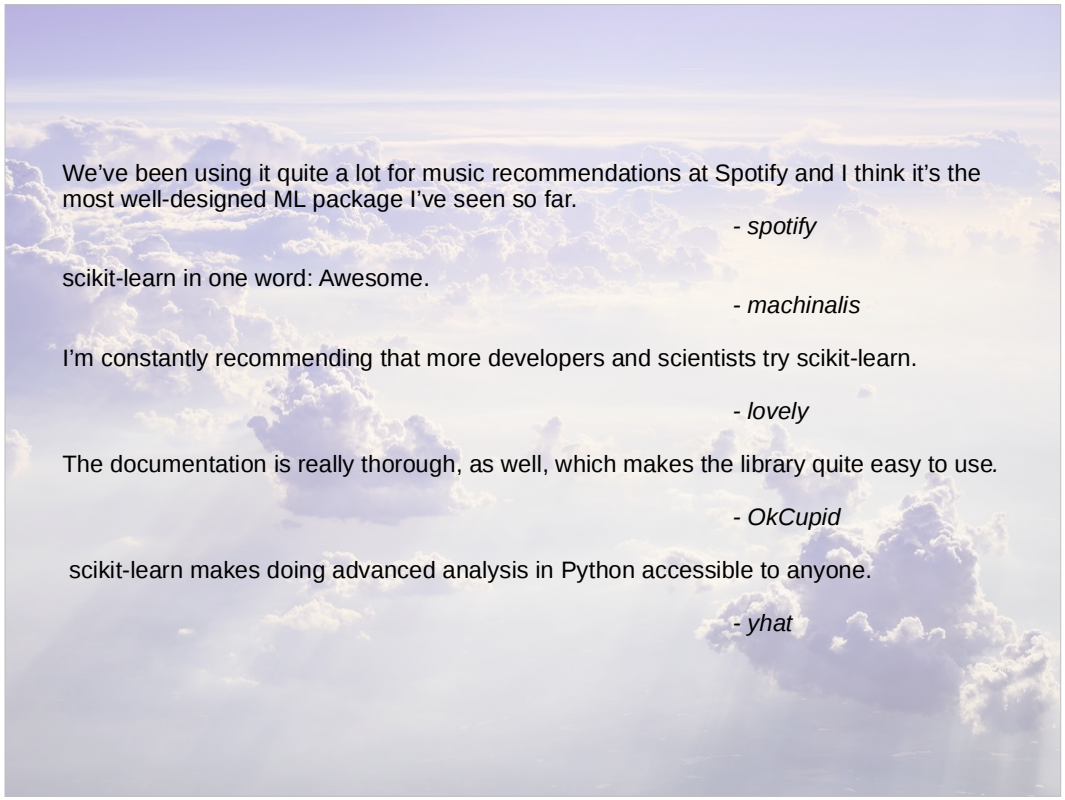
Let’s start with a mission statement. To me, and I think to many others in the project, the mission of scikit-learn is to create effective and easy to use machine learning solutions, and make them accessible to everybody. In other words, we want to commoditize and democratize machine learning. This is a goal we share with the people at google, like Josh, working on Tensorflow. We have a slightly different angle, as we are an open source project, not a company. And we have been doing this for about 6 years now. Take that tensor flow.



There is this nice quote here by alan kay, which says “simple things should be simple, complex things should be possible”. Whenever I think about a new feature or a new api, I keep that in mind. When in doubt, in scikit-learn we err on the side of keeping simple things simple, sometimes sacrificing some of the power of more complex designs. But I’ll talk about that more later.

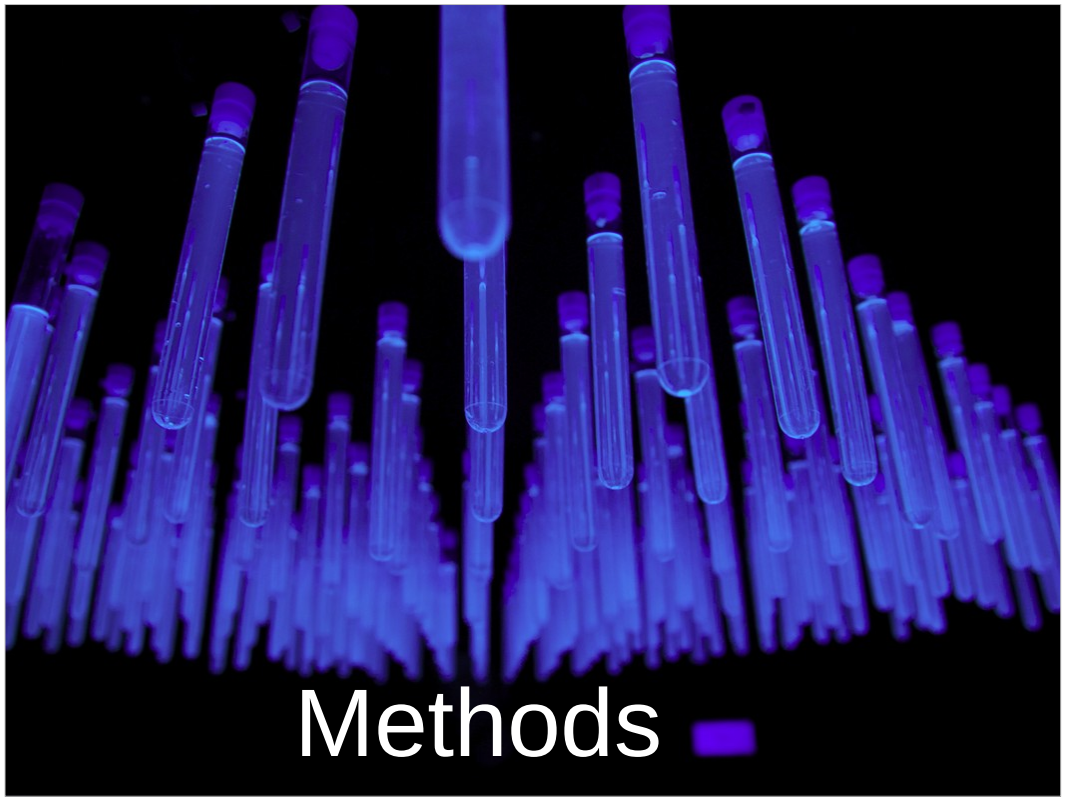
So this is our mission and philosophy. But how have we been doing so far? I think it’s fair to say we have been quite successful. At least for people that use python for data science, we are currently the default solution.

Here are some numbers, showing that we are quite popular on github, we have a lot of traffic on our website, and the paper describing scikit-learn has been widely cited. Actually the cite counter goes up by a couple every day. It really makes me wish I was on the paper.

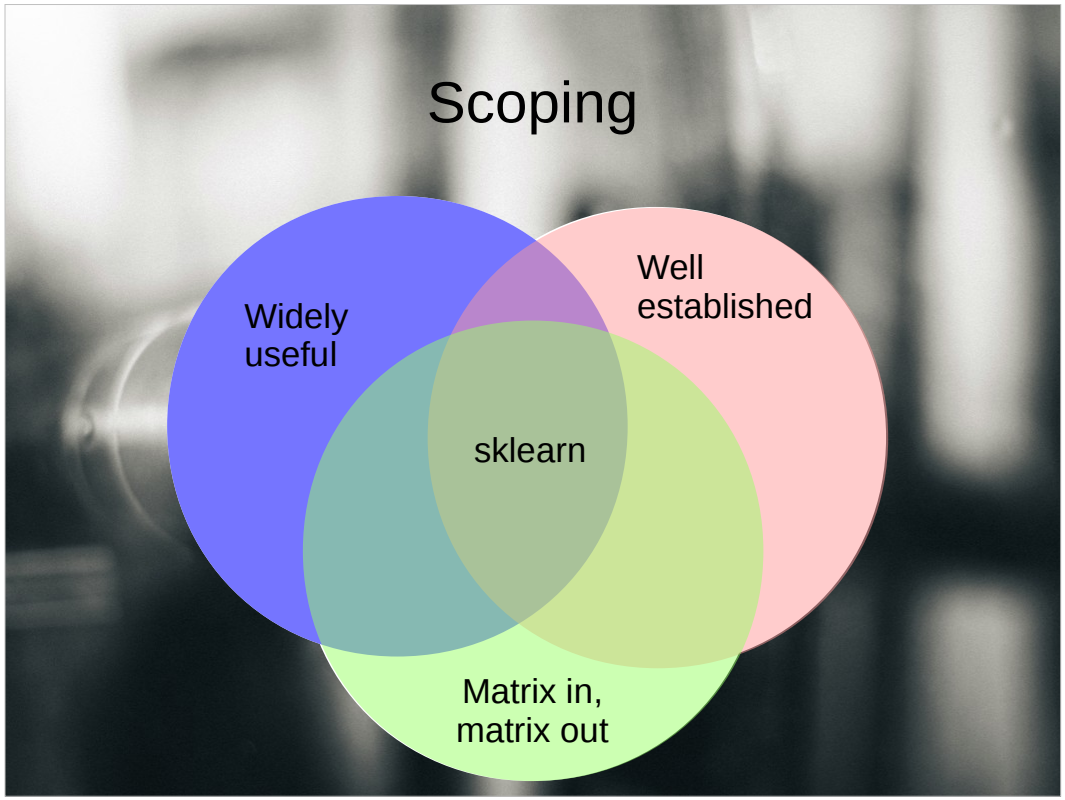


And people say good things about us. People find that scikit-learn is simple to use, and that that it is well documented. Both are high up on our list of goals to make scikit-learn approachable and usable.

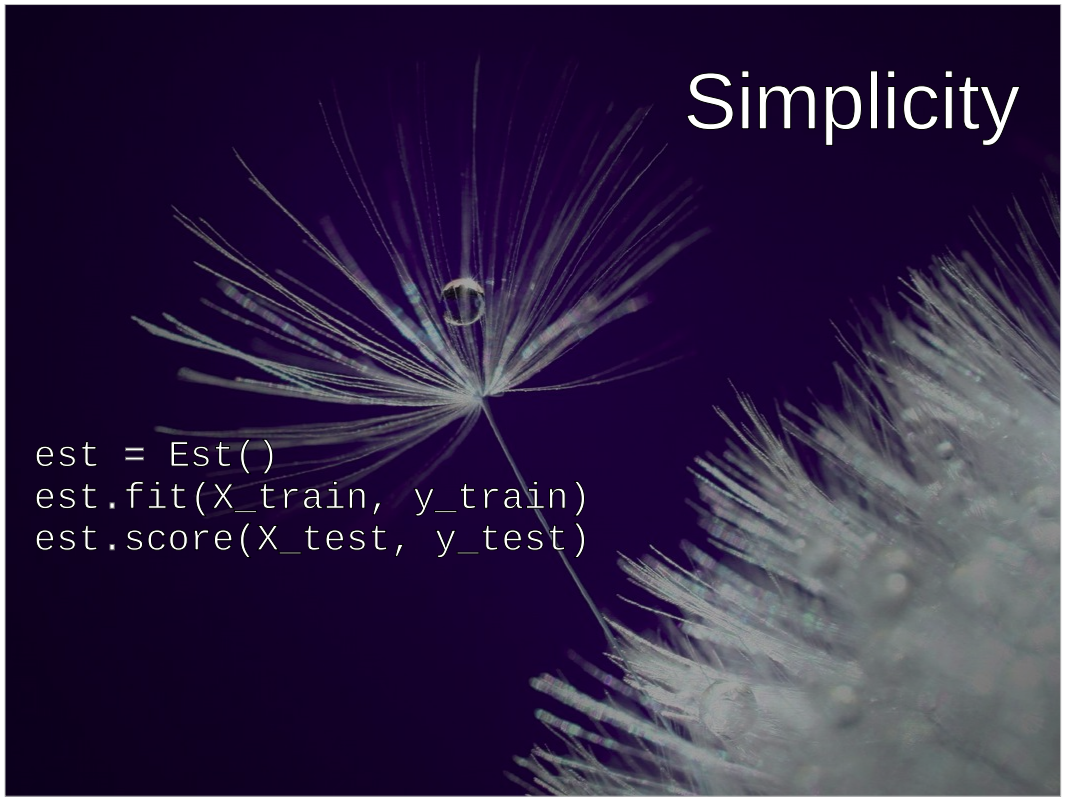
Scikit-learn is mostly used in offline data analysis and in scientific experimentation, but some people also use it in their life production systems – though it only seems to scale to spotify scale, not google scale.



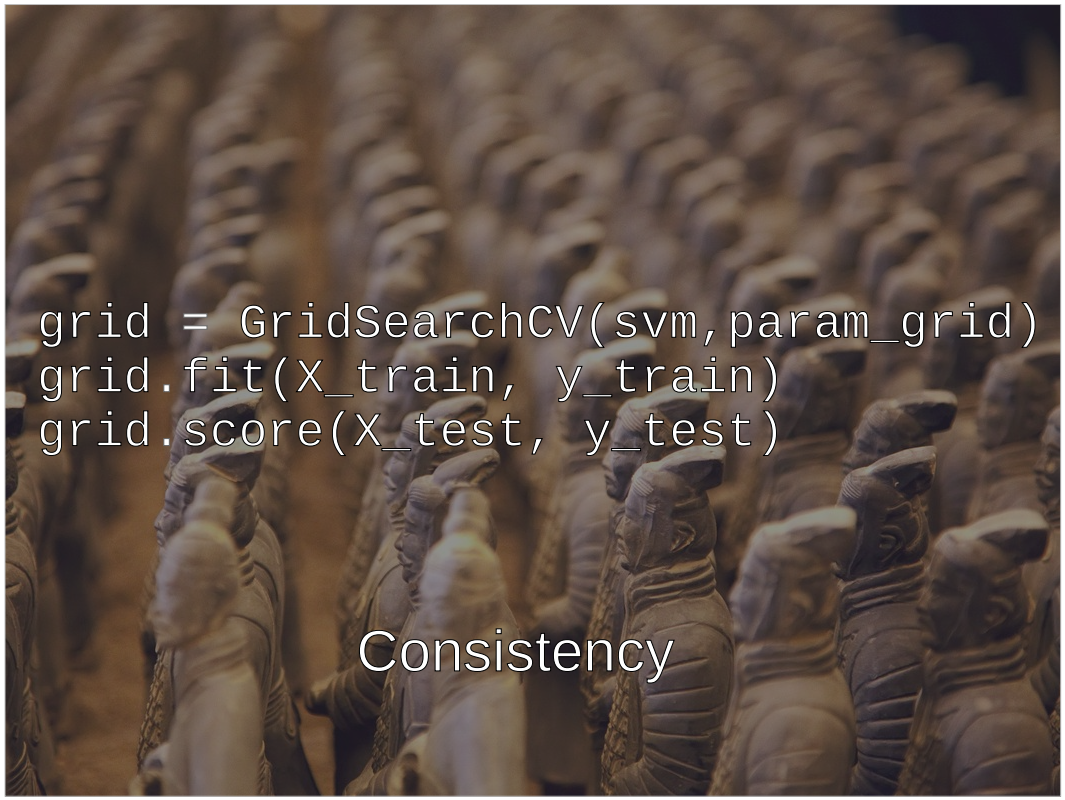
So how did we achieve these goals, in particular as an open source project? Open source development is quite different from working in a company, because you can’t really tell anyone what to do. Still we managed to end up with a pretty good package. By the way, I take no credit for any of these principles, most of the methods for making the package great were well established when I joined. I do think I helped implement them, though.



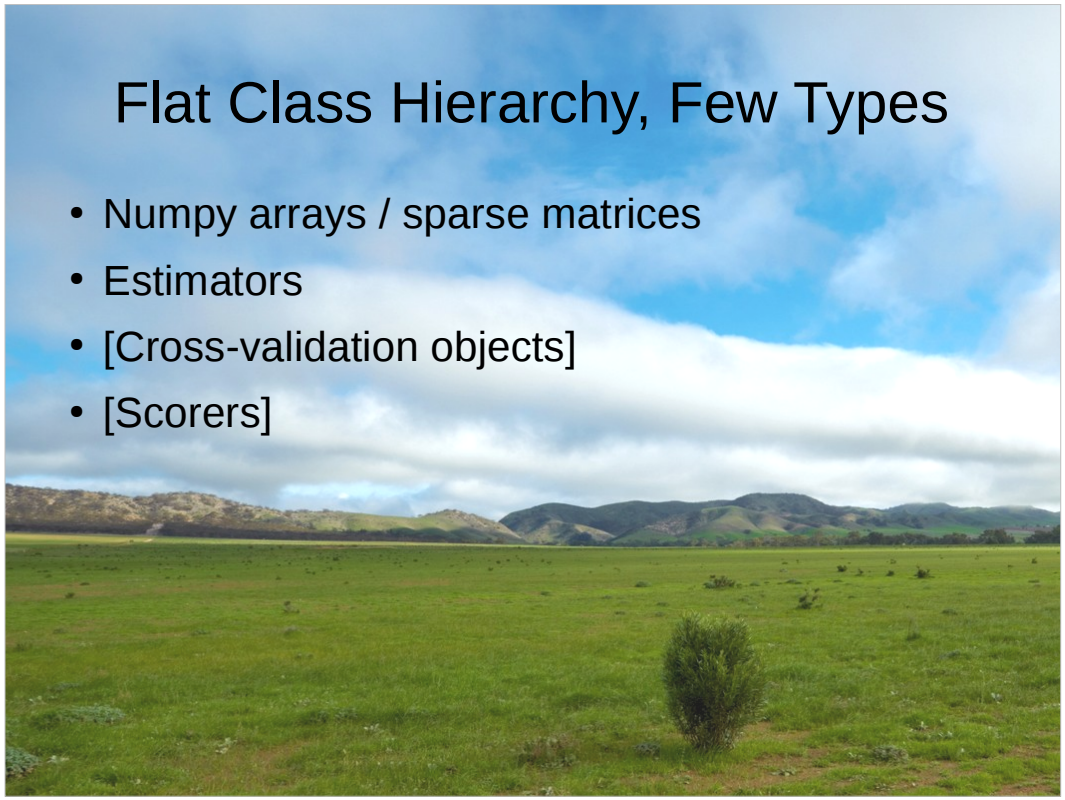
One of the most important tools we used is very rigorous scoping. We only include methods that are well established and widely useful. We don’t have methods that are only good in biology, or that are only good for financial data. We aim to have a very broad spectrum. We also limit ourselves to well-established methods. One of the reasons we don’t want to implement the bleeding edge is that the field of Machine learning is very fast moving, and what looked promising half a year ago might no longer be any good. Basically everything in scikit-learn stood



To make scikit-learn usable, we really focus on simplicity. You can build really powerful machine learning experiments and pipelines with scikit-learn with just a couple of lines of code. And doing a standard supervised machine learning task boils down to just these three lines: instantiate a model, fit the model on the training data, evaluate it on the test data. We really want to keep simple things simple.



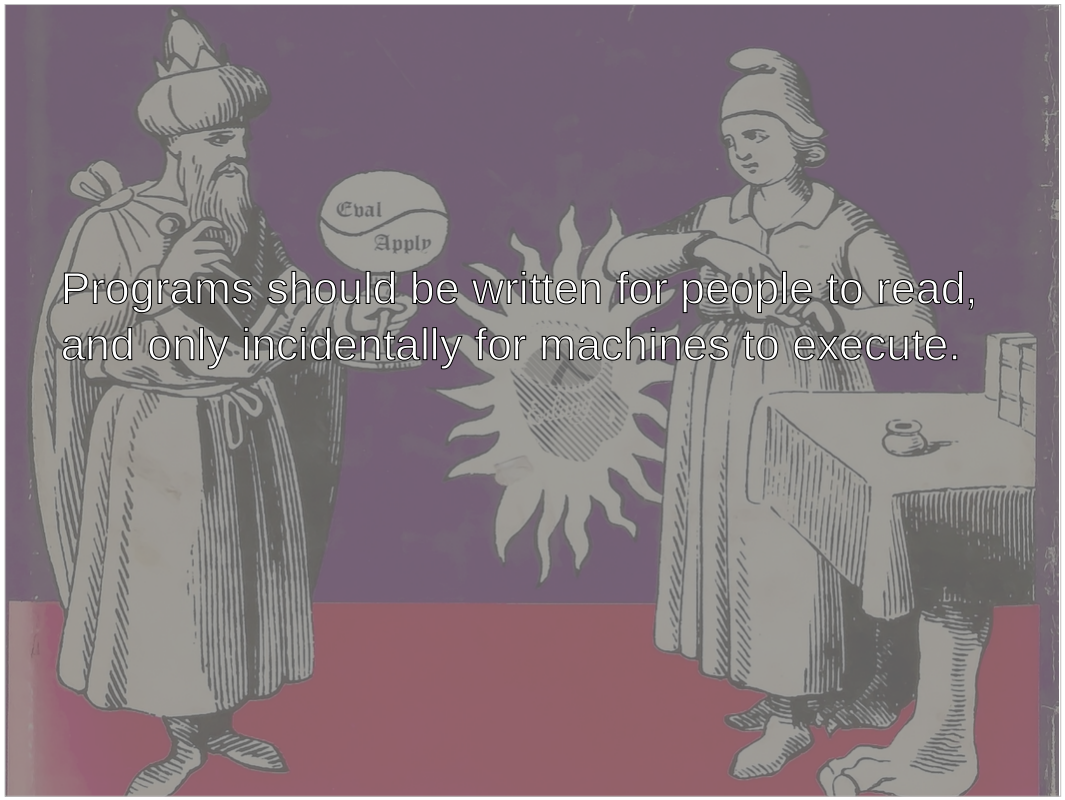
And not only is the scikit-learn api designed to be simple, it’s also designed to be consistent, and only have a few basic building blocks. The grid-search for example acts as a model (which we call an estimator), and has exactly the same interface as any other model. So adjusting the parameters of a model looks like just another model fitting.



This simplicity is not only in the methods, interface, it is also in the types. There are very few types of objects in scikit-learn. Data is always numpy arrays, and basically every building block in scikit-learn is an “Estimator”. And that’s it. That’s more or less the only two types of objects you have to deal with in scikit-learn.

There is no dataset object, for example. There are some limitations in restricting ourself in this way, but having few types and a flat class hierarchy not only makes the project easier to use, it also makes it

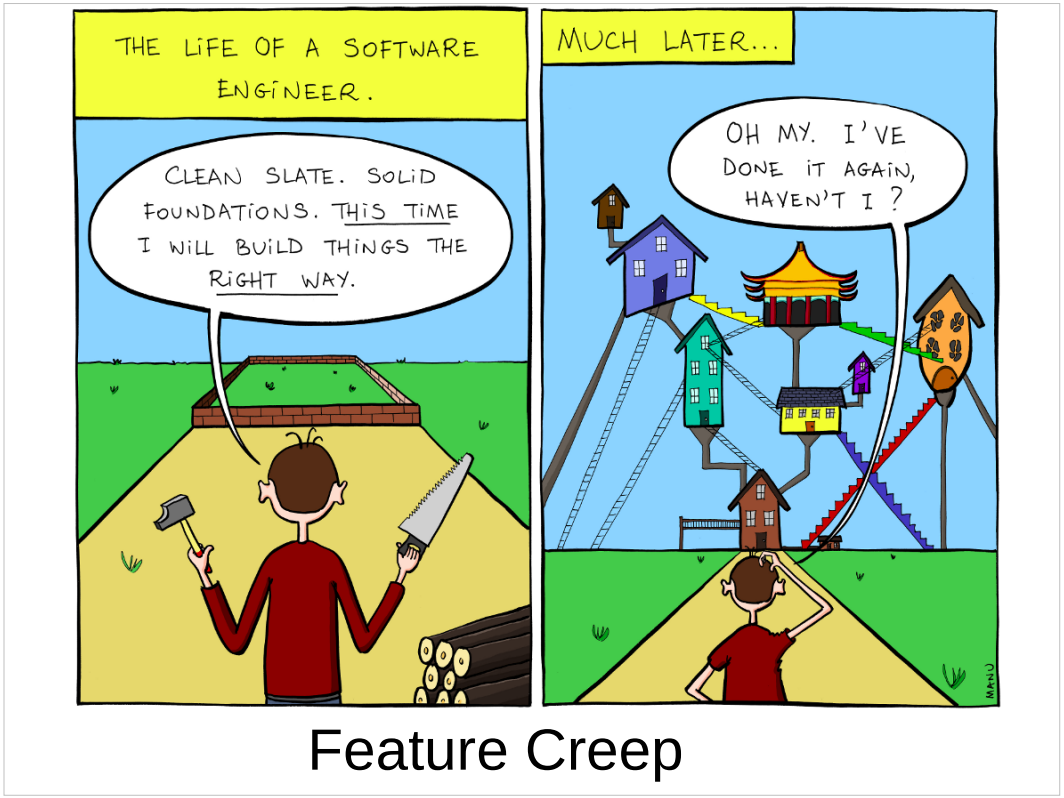
The small number of types brings us to the next point, which is particularly important for an open source project: maintainability. The project was built with maintainability in mind from the start. I’d say maintainability is more important in open source, because at some point, you might want to find another idiot to waste their weekends on maintaining this code voluntarily. You can’t hire a new engineer and say “here deal with this mess of legacy code”. You wouldn’t have any contributors. Maintainability helps us create a thriving ecosystem of contributors



Maintainable code is easy to read and understand for someone new to the code base. You should keep in mind this classic quote from the classic wizzard book.

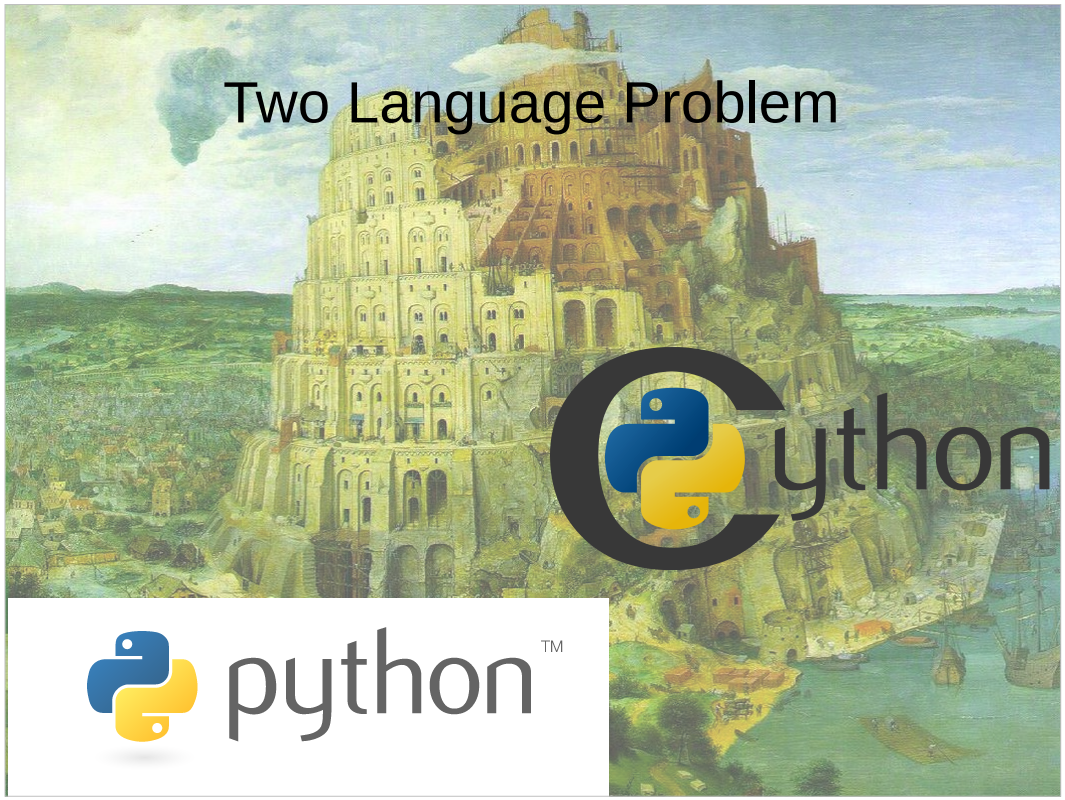
One of the easiest way to keep a project maintainable is to avoid code. I really don’t like code, and I like to delete as much of it as I can. Some people really love writing long and complex code. But some day some poor soul has to read that and understand it. Every time you add a new feature, you add more complexity, and more moving parts to a project. Stop writing so much code, and start deleting more code. For every feature you add, ask yourself: is this worth the added burden for my future self, maintaining this code to the end of time?

So now that I talked about things that worked well for us, here are some things that we haven’t really figured out, or that keep coming up.

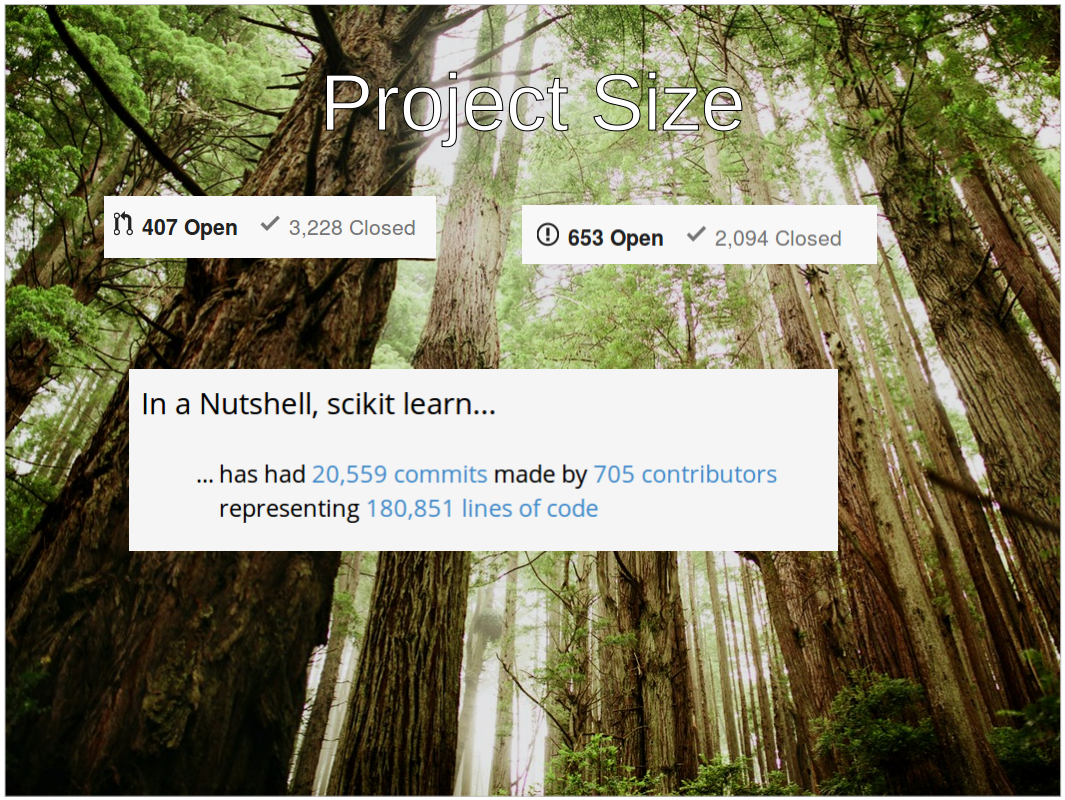


Feature Creep

I'm not sure if you know this comic, it's from bokersworld. If you don't know bonkersworld, definitely check it out. So, you know, whenever you start a new project, you think to yourself: This time I'll do it right. And then at some point you end up with some frankenstein monster. The comic says "much later", but in my experience, it usually doesn't take that long. Even though we have pretty strict scoping, feature creep is a problem, and we constantly fight with the tradeoff of adding features and keeping complexity of the project reasonable.



Another problem that is somewhat ongoing is the two language problem. Python is a nice high level language and very easy to use. It's general purpose and great for data science. But it's slooow. Which means we have to use Cython, which basically boils down to C, to implement most algorithms. That is frustrating because it means we need to work with both python tooling and C tooling, for debugging and profiling. It also means that the barrier to contribute to the actual algorithms is quite high for people that only program in python, because all of a sudden they



Another struggle for us, that I guess is also particular to open source, is our fight with the project size. We have a lot of people contributing and reporting bugs, which is great. But we have very few people that have a good overview of the project, and most of these are busy writing grants all day to fund the people actually implementing stuff. That means we have hundreds of contributors, around 10 or 20 people that review code, and at the top maybe 5 very busy people that try to keep a lid on things. And the problem is that you can't really just add people to the

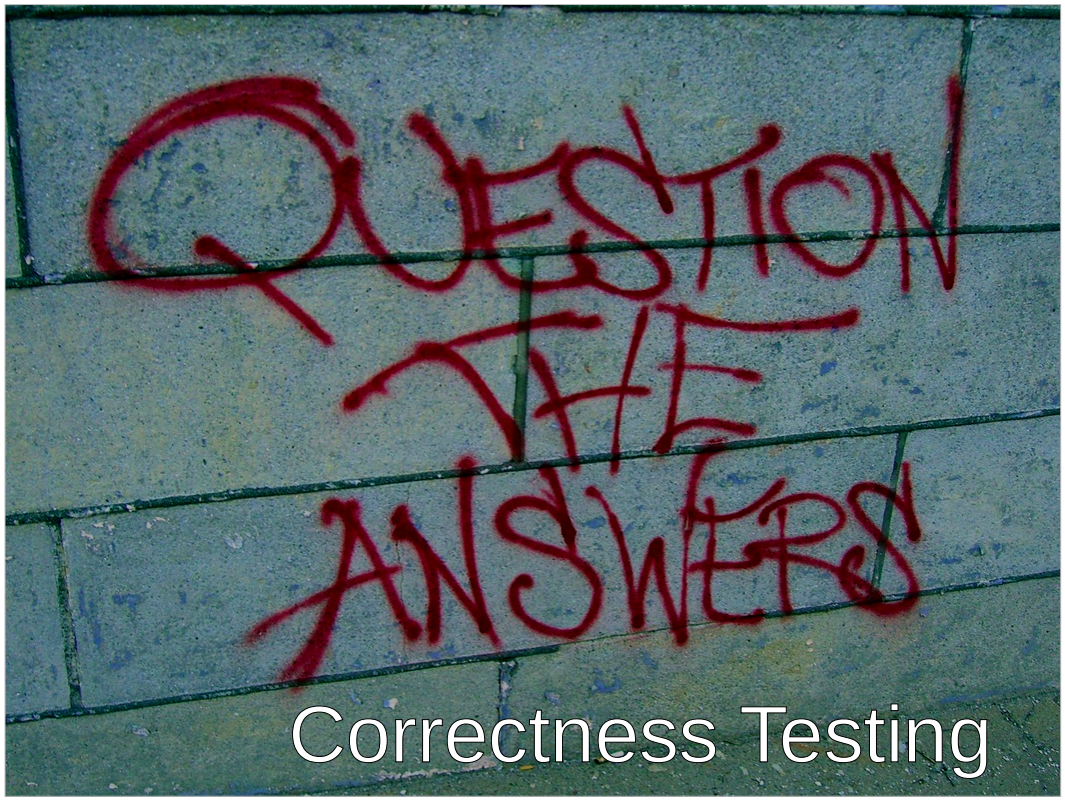
Before I give a glimpse into what we are currently working on and thinking about, I want to pose some open questions, and I’d love your input on those.

A real issue for data science in python right now is data structures. There is numpy arrays, which basically are C arrays. They are efficient, but they are homogeneous. Usually they are all floats. That's not great for heterogeneous data, that might have a mix of categorical and continuous data.

Then there is pandas dataframes, which allow heterogeneous data, but they don't have a C api. So we can't write fast algorithms on them.

Then there is a whole bunch of others, most prominently maybe the Sframe from turi. It's great

Another thing that I’m struggling with is benchmarking. We want to have good defaults and efficient implementations. To measure these, we need benchmarks. But there are very few realistic datasets out there. The UCI collection doesn’t really reflect the day-to-day of most data scientists. But the day-to-day data of data scientists is locked into some company, and we have no access to it. So we keep using scientific datasets, which might be a really bad benchmark for a large part of our user base.

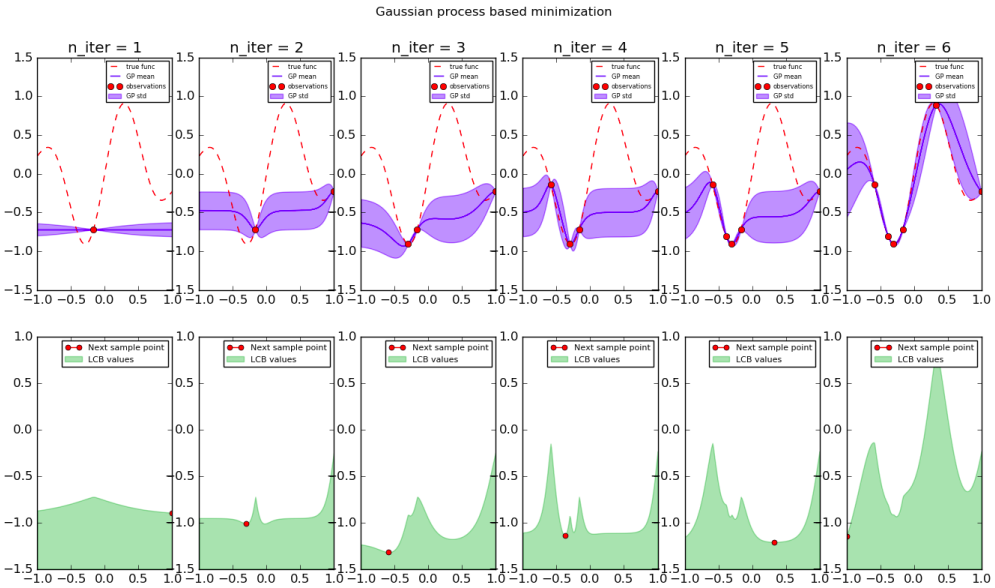


The last, but certainly not least, of my open questions is correctness testing. This is a really tough one. How do you know that your algorithm works correctly. Or just mostly correctly? If I ask people that implement machine learning for a living, the usual answer I get is that they test against scikit-learn….

That means that people reproduced our results, but maybe it just means they reproduced our bugs. And all software has bugs. So how can we protect against incorrect algorithms? Some algorithms have reference implementations, but they are often in R or

So, finally, I want to give you a bit of a taste what we've been up to recently, and what you can expect to see in the near future.

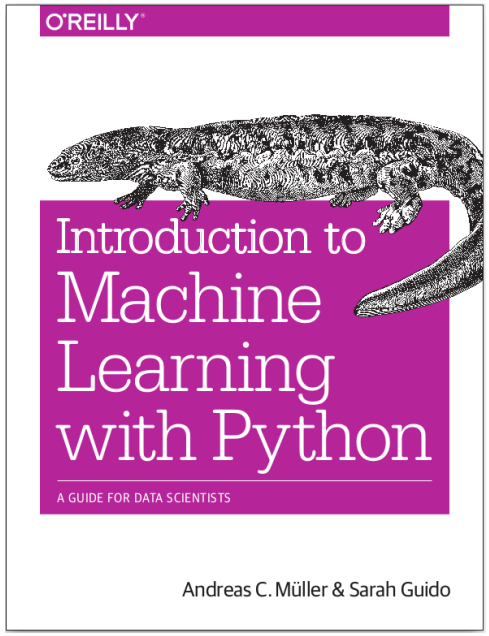
Bayesian parameter optimization



One of the things I'm thinking about a lot these days is automation of machine learning, also known as AutoML. Currently, you still need to pick the right algorithm, and you still need to pick the right parameter ranges for your parameter search.

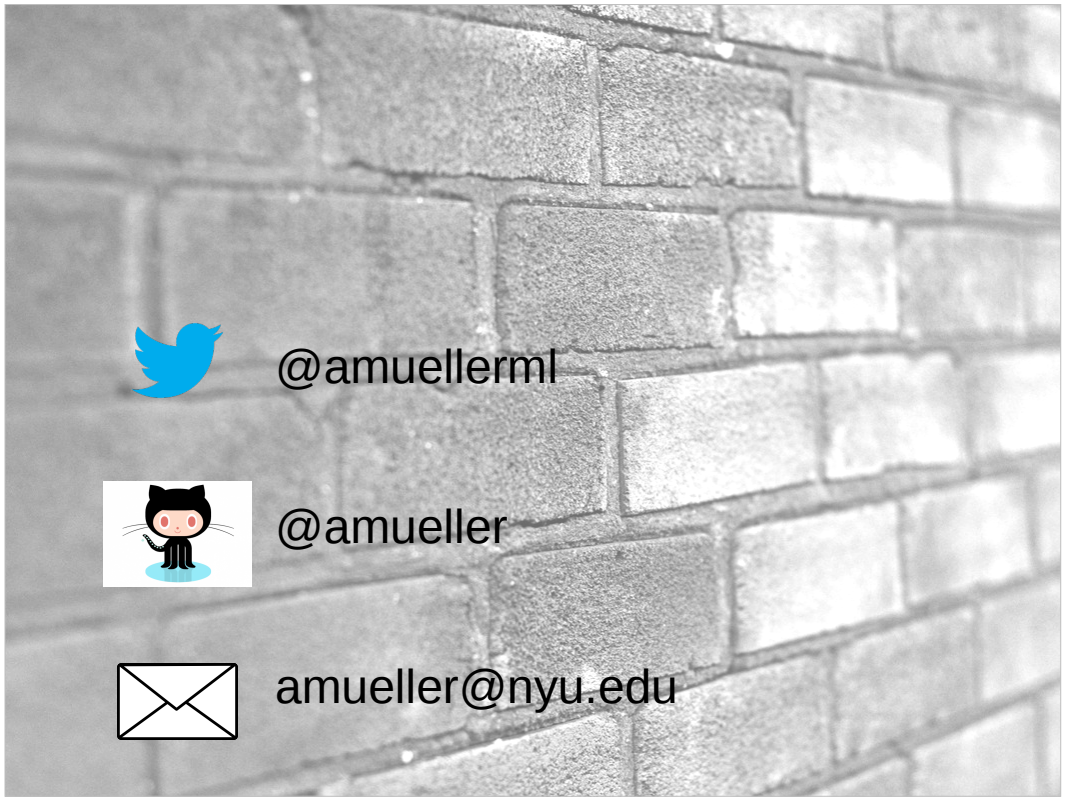
We can at least get rid of the parameter search by using bayesian optimization for hyper parameters. This has been going around the research community for a while, but there hasn't really been a wide adoption. And with wetlab being bought by twitter, now might be a good time for us to add an

Another future direction to improve usability, which is somewhat connected to the question of data structures, is working with feature names. Often your input features have some semantic names, like age, gender, time spend online, adds clicked on and so on. Currently, scikit-learn ignores these, which makes it a bit tricky to analyze complex pipelines that include feature engineering and feature selection. But we are working on improving the handling of feature names, so that you can easily inspect what your model is doing with your original features, no



Buy my book. It's about scikit-learn. It has no math, because there are great books that have all the machine learning math. This one has mostly code.

Actually, you might not have to buy it as Dev Fest has organized a give-away later today. So stay tuned for that.



That’s it, thank you for attention.