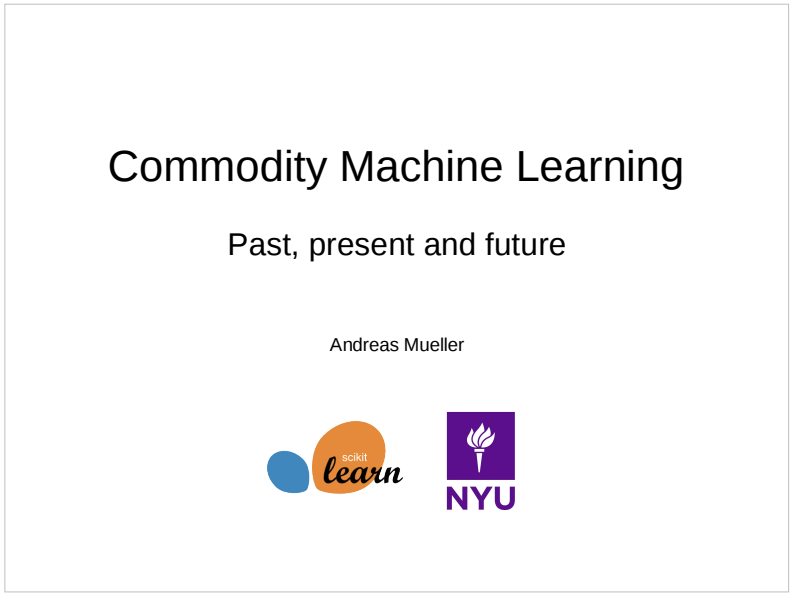


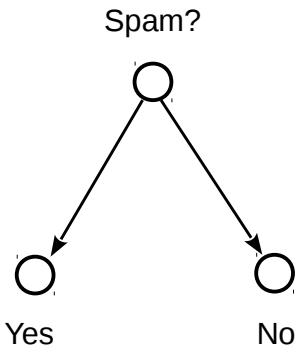
Hey everybody, welcome to my talk on commodity machine learning, past present and future.
And thank you to the organizers for inviting me to this great conference.
I already met a lot of new people and had some great discussions.

I really enjoyed the other two keynotes as well, Van and BG are amazing speakers. My talk will be quite a bit more technical than theirs, I hope you'll like it anyhow.

What is machine learning?

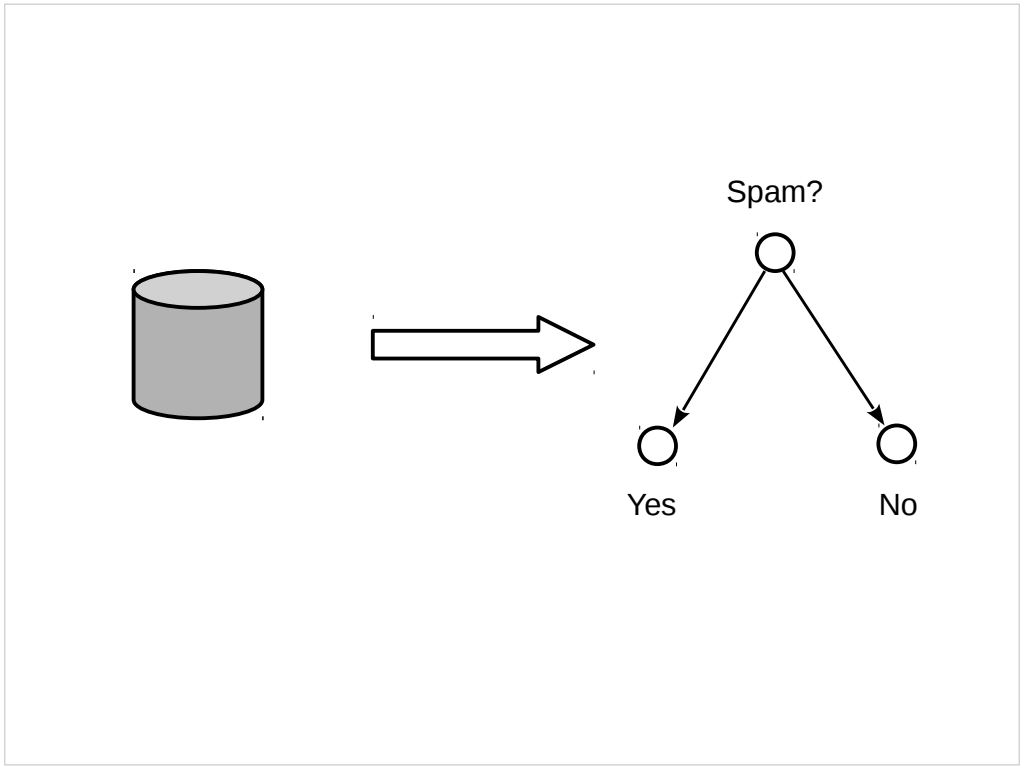
Let’s start with “what is machine learning?” Many of you are probably familiar, given the recent hype. But this is PyCon and not PyData, so I want to give a little bit of background and my personal perspective.

Automatic Decision Making



The goal of machine learning is to automate decision making processes. Instead of having a human make a decision, we want a program to make a decision, for example, “is this email spam” or “does this person have cancer”, or, one of my favorites “is there a face in this image”.

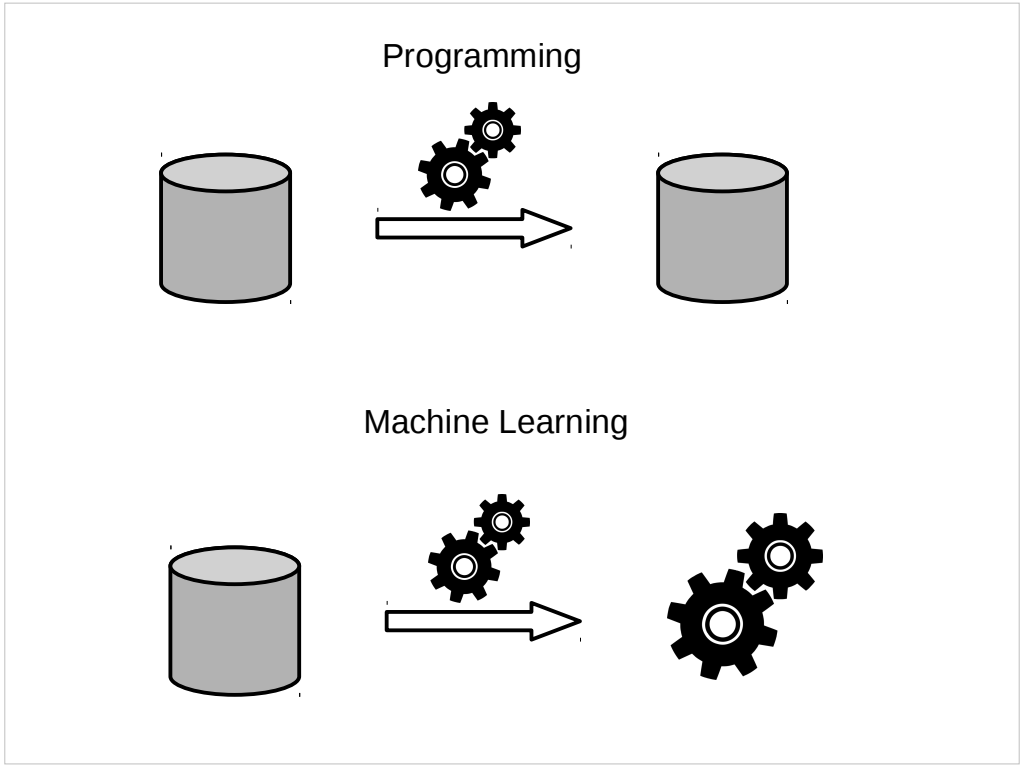
For some simple decisions, we, the programmers can come up with some rules that allow the program to make decisions. We could say “if the email comes from a known bad host, or contains “You have won” or three dollar signs, it’s spam”. That would get rid of at least some amount of bad emails.



This is where machine learning comes in. Instead of programming the rules ourselves, we let an algorithm decide how to make a decision. So we let the algorithm come up with it's own rules.

The only thing we need to make this work is data, more specifically, we need to show the algorithm examples of possible inputs and the decisions that should be made for these inputs.

For the spam example, we need to collect a whole lot of emails, and have a human label them as spam or not spam. Then we can provide these emails to the algorithm, together with the information if they were



Another way of thinking about machine learning is as a generalization of programming:

Usually, a program gets as input some data, and produces some data as output.

A machine learning program gets as input data, and as output it produces a new program – the program that knows how to make decisions.

However, keep in mind, machine-learning is not a cure-all. And it’s somewhat complex, so if you can actually come up with a good rule yourself, just use that, and move on.

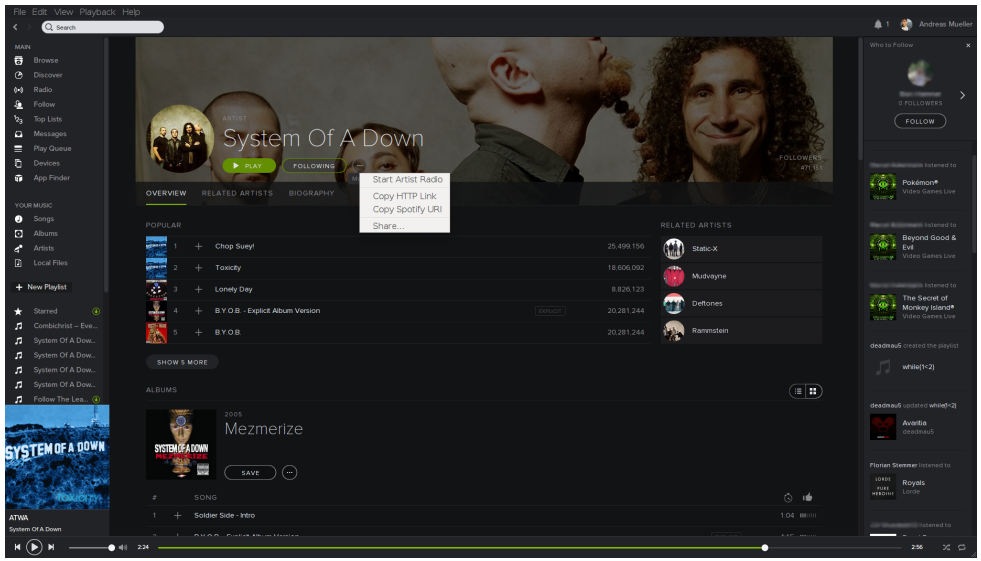
Machine learning is EVERYWHERE

Now you might think (well, at least if you haven't benn on the internet for the last couple years) "wow that sounds interesting but is it actually useful?".

And yes, it is. These days, machine learning is every where. I'm pretty sure half of the tabs you have open in your browser right now show you the results of some machine learning algorithm. Let me give you some examples.

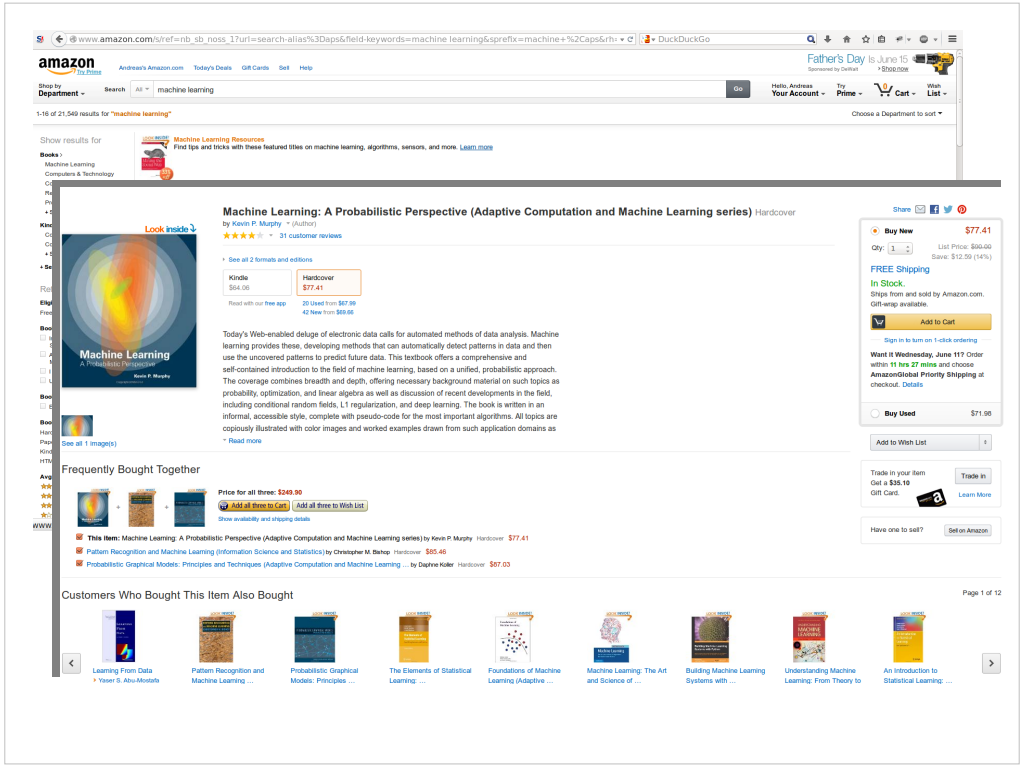
So this social network here might look familiar to you. There are probably many more algorithms at work here than what I can see, but some are fairly obvious. They are showing me advertisements, and they use machine learning to know what ads to show. They also estimate how much money each add should cost.

They also use machine learning to know what to show in my timeline, and in which order. If you upload pictures, they'll find all the faces, and they might also label the faces with the names of your friends.



Another machine learning application I’m constantly using is spotify.

Here, you can see an artists page, and there are links to “similar artists” - generated by machine learning. And an “artist radio” - also generated by machine learning. They have recently started a pretty big marketing campaign for their “discover weekly” feature, which are tracks they recommend for you once a week. Again this is driven by, you guessed it, machine learning.



Here's another website you might have open. I actually know a bit more about this one because I worked for them. But many of the machine learning applications are fairly obvious. This is a product page for a pretty good machine learning book.

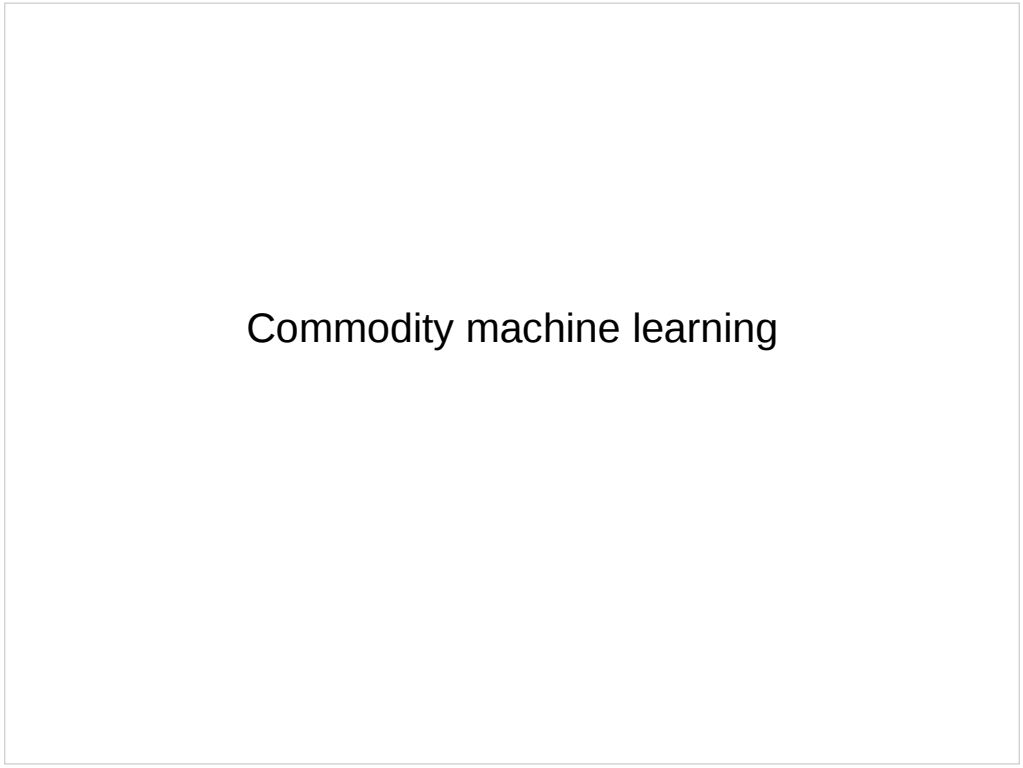
At the bottom, you can see recommendations of what other books I might want to consider buying with it. And other items people bought that bought this item. Both are driven by machine learning. Less obvious is machine learning that picks a seller to show in the buy box to the top right. There are many people selling this book, but one was selected just for you

Machine learning is increasingly used in sciences of all kinds, from astronomy for finding distant planets, to finding the higgs boson or predicting elections.

It's also used in many engineering disciplines, in particular in chip and circuit design to estimate efficiency and power consumption without actually building a chip. There's of course also self-driving cars, and maybe soon drones delivering packages.

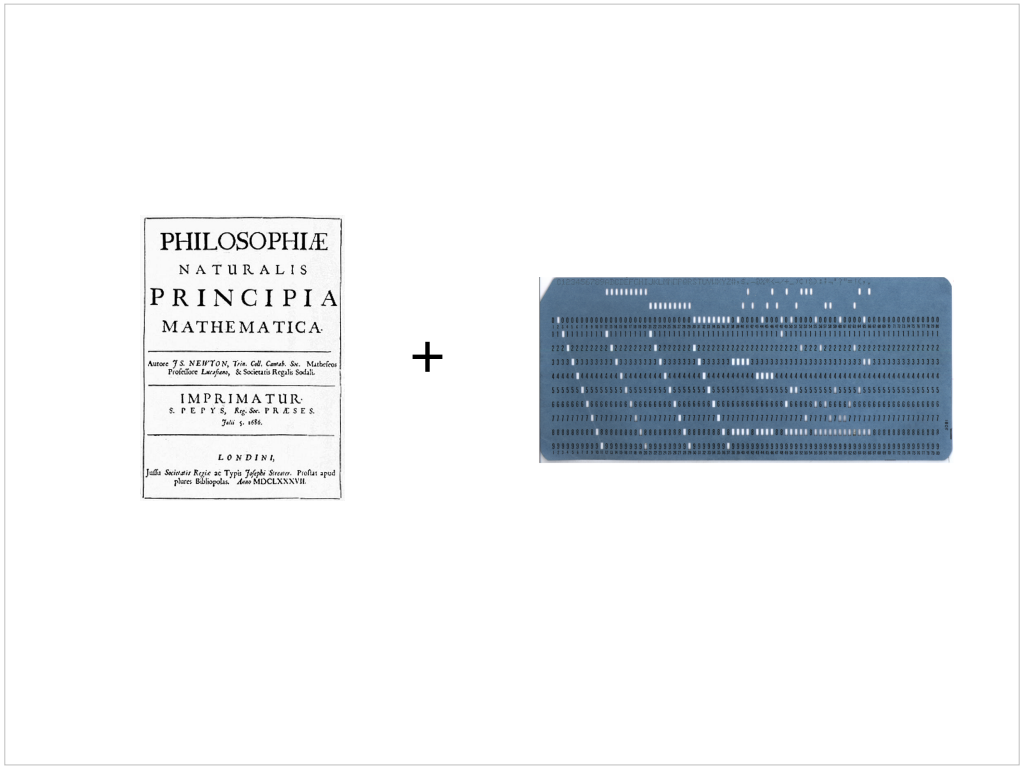
In medicine, machine learning is used to diagnose disease based on lab results and medical imaging, but also to estimate the effect of new drugs.

And these are just some of the many important



So now that we talked about what machine learning is, and where it’s used, I want to talk about what I mean by commodity machine learning. To me, machine learning is an important tool in data processing, and should be accessible to anyone. It shouldn’t be restricted to large companies with giant R&D departments, or to sophisticated research labs, it should be accessible to all of you. But there are some hurdles for a relative sophisticated technology like machine learning to become a commodity. What I want to talk about here is the path that machine learning tools and software took, and where it is

So let’s start with the dark ages of machine learning software, say the early 2000’s



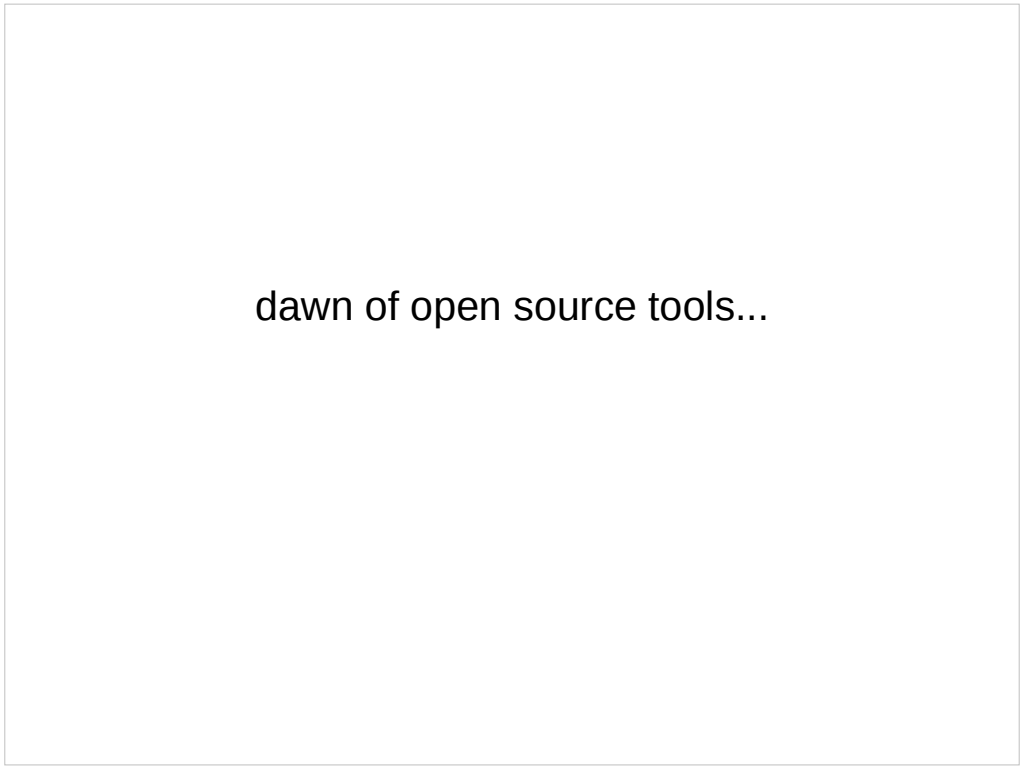
I got into machine learning in 2009, at the start of my PhD. Luckily, by this point, already a lot of machine learning tools were available.

But if you go back just a little bit further, the amount of code that people are willing to share was small, and the amount of documentation was basically zero. So if you wanted to apply an algorithm, you usually had to do it from scratch. You took the paper, you implemented the algorithm, often in C or C++, or matlab if the algorithm was fast enough or you didn't have a lot of data.

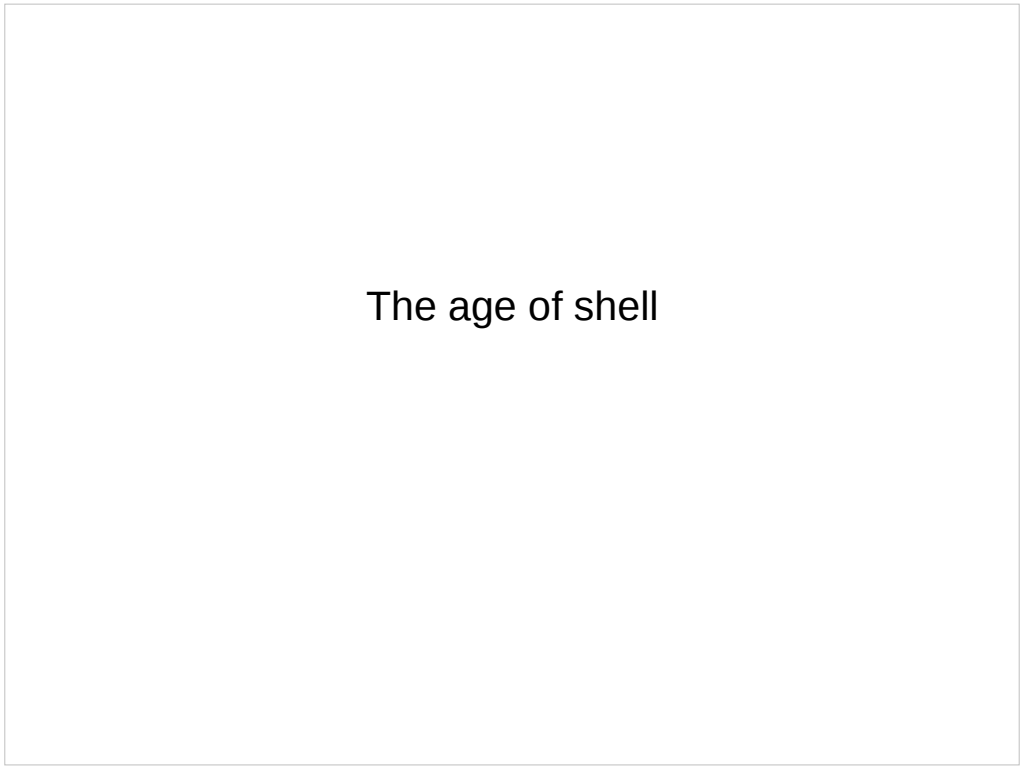
You maybe checked if you can reproduce the results in

There are some obvious problems with this approach:

- First, it's very time-consuming and tricky to implement complex algorithms from scratch in a language like C or C++. One thing I often stumbled over is that there is no widely accepted common language for arrays in C++, so often your have to rewrite many standard operations yourself.
- Then, papers often leave out important details, or the tricks that are needed to make an algorithm work. The implementation is often different from the mathematical guarantees, and that's not always mentioned in the paper.

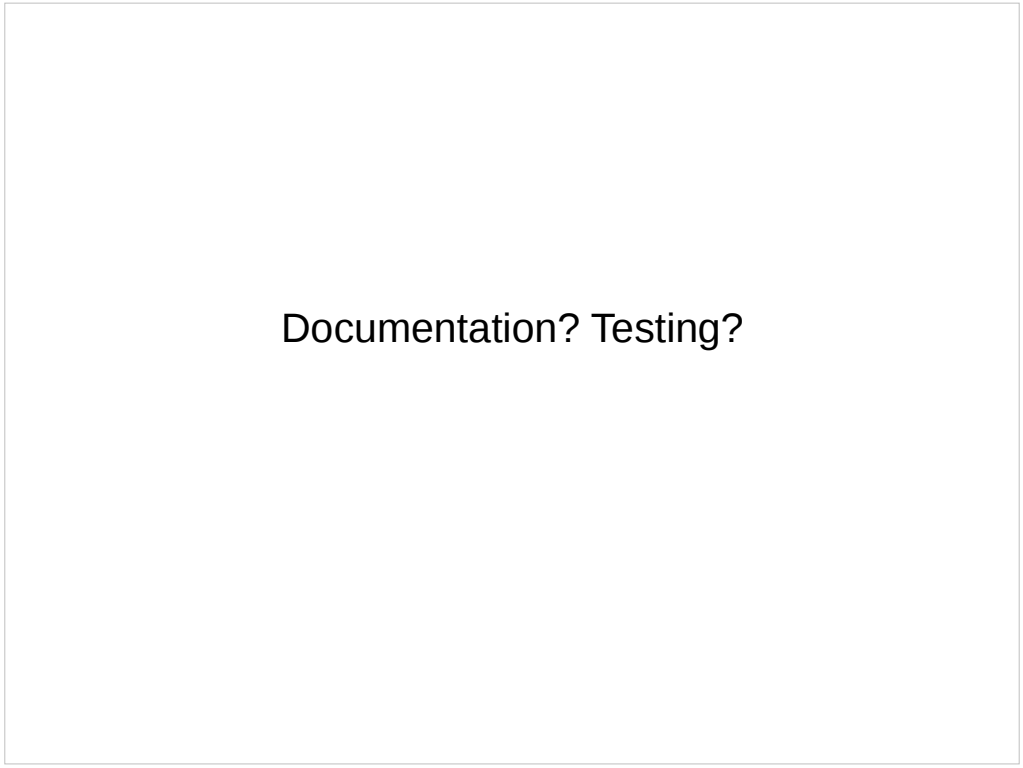


Luckily, as some methods got more wide-spread use, open implementations became available, with libsvm being one of the pioneers. Then more fully-fledged tool-boxes like Shogun and weka appeared, and R and Python started to take hold.



Still, in the early days, many efficient implementations were only available in C, which is quite annoying, because no-one wants to run experiments in C. Experiments are often very interactive and, you know, experimental, and that doesn’t work well with the slow write-compile-debug loop of C and C++. Command line interfaces are slightly better, but encourage the manual stitching together of tools, often with a workflow that’s not written down anywhere. That makes it nearly impossible to rerun any experiments.

As many tools were only available commercially, or



Another big pain point with early machine learning software, and I guess still with a lot of scientific software out there, are installation, testing and documentation.

Scientists often just upload some C or matlab code to their website. If you’re lucky, maybe python code. But that’s it. No build instructions or declaration of dependencies, no documentations. And tests? Most scientists don’t even know that that’s something you should do. Things are starting to get better, but these are still problems in academia today.

As someone who wants to apply the method, it can be

Scikit-learn: User centric machine learning

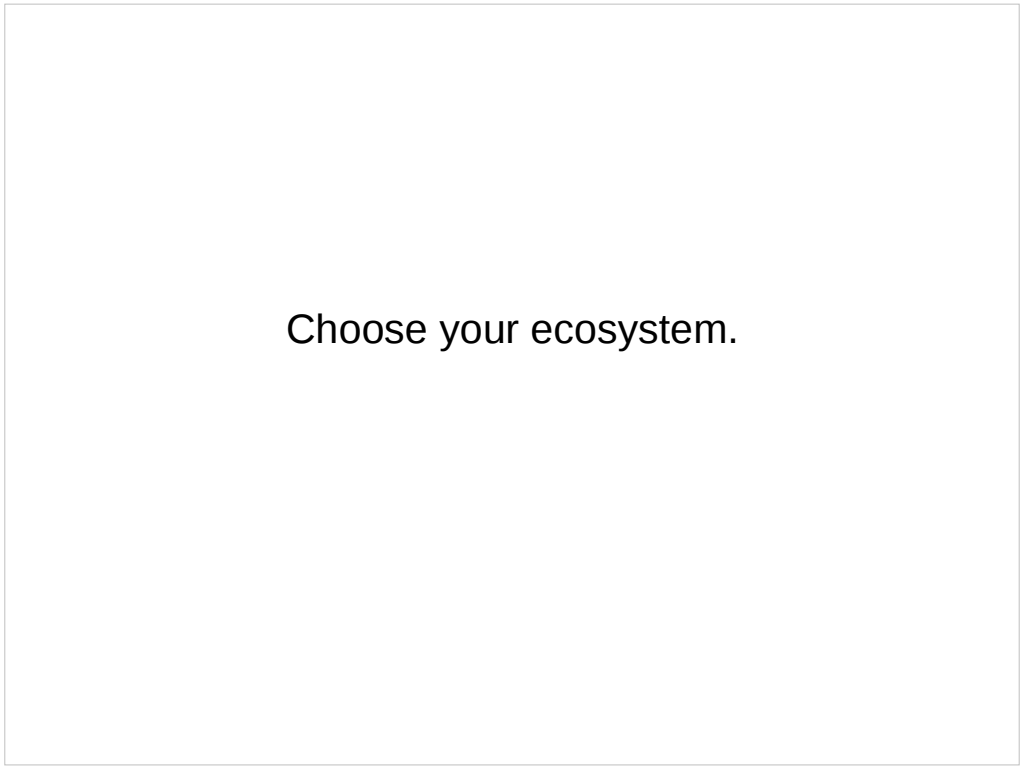
So now let’s talk about scikit-learn.
To me, this is where the commodization really started.
While there were other great libraries for machine learning in python before scikit-learn, I think the approach that scikit-learn took was pretty unique. I can not take any credit for it, though, most of the design goes back to Gael Varoquaux and Alex Gramfort. The big difference in how scikit-learn is developed is that it puts the users first, instead of looking down on them.

```
.fit(X, y)
.predict(X)
.transform(X)
```

I think there are some key ingredients to the success of scikit-learn, which help making scikit-learn as accessible a tool as possible. First, it is easy to install. Second, it has a very simple interface, consisting mostly of three methods: fit, for building models, predict for making predictions, and transform to change the representation of your data. You can’t really make machine learning any more simple than that. The third is documentation. There is really a lot of documentation, to the point where you might not even need to know about the method to be able to apply it

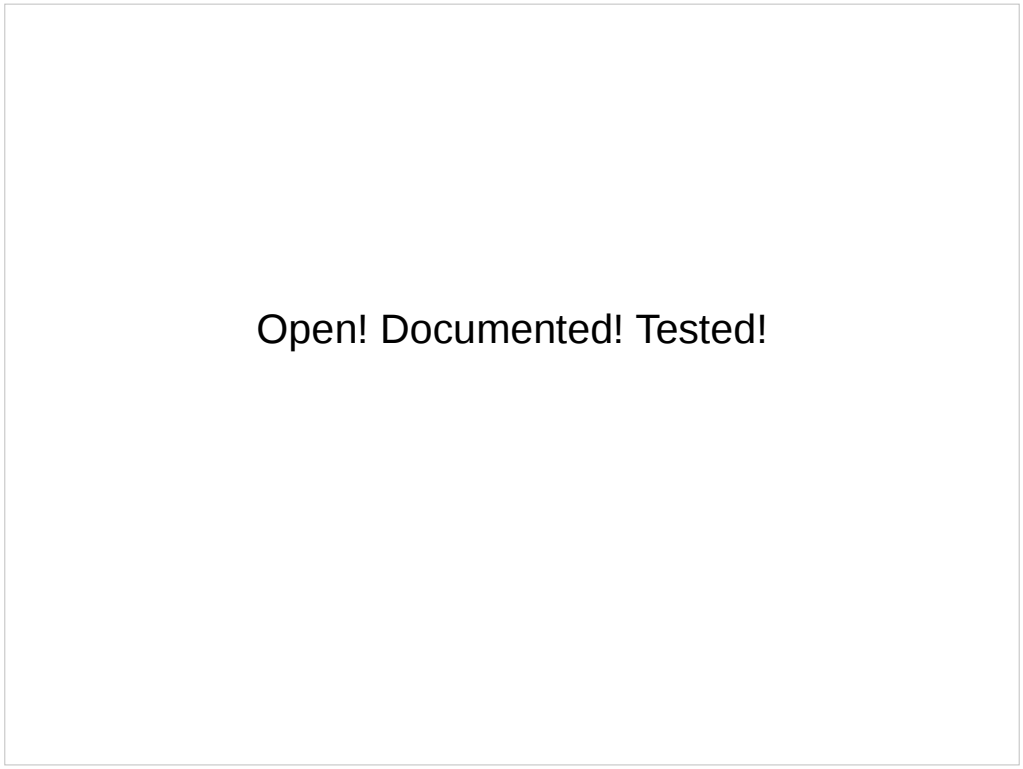
present

Which brings us to the present.



There is now a thriving eco-system of machine learning and data science tools in Python, but not only in Python! There are also R and Julia, two growing languages for data science and scientific computing with their own open ecosystems.

And within these eco-systems there are even more choices for the users available. I'll focus mostly on the python ecosystem in what follows, but I'm glad that we are not the only ones trying to solve data analysis and machine learning questions, and all of these tools are openly available.



One of the things that are great about the python ecosystem in particular is that the community puts a strong emphasis on documentation, testing and good software development practices.

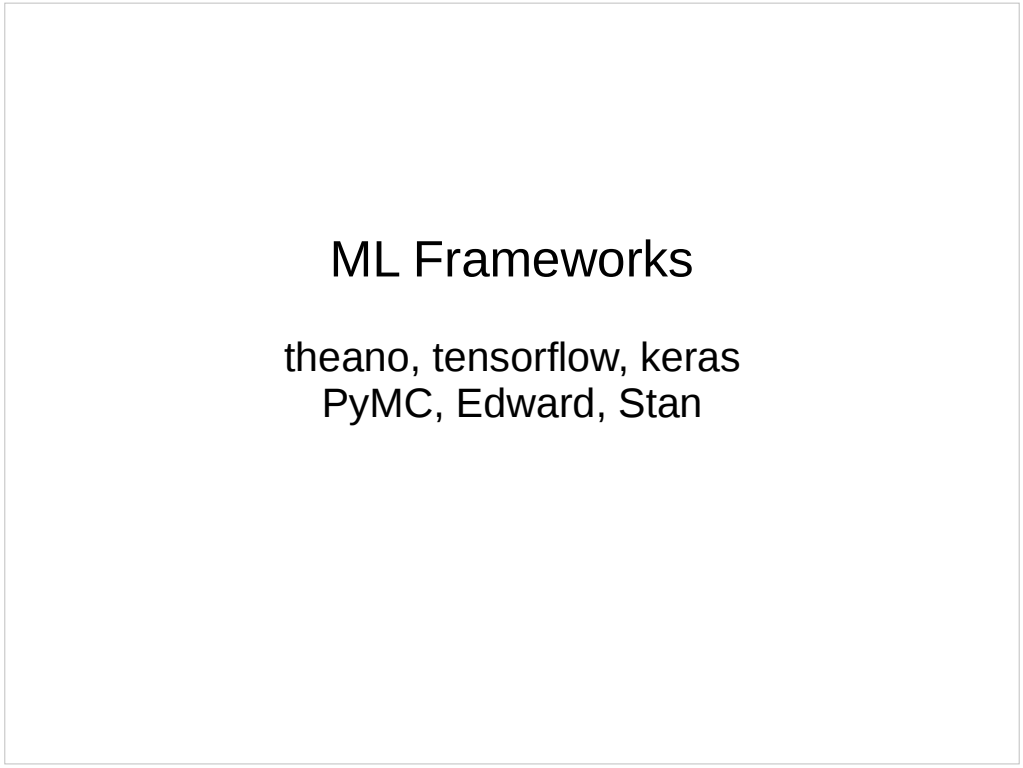
Also, people tend to use version control, usually github. And continuous integration!

That is so much better than downloading a tarball from a website that maybe compiles with one specific operating systems with one specific version of the compiler! (And I’m not joking, this happened to me more than once).

And more than just good software development practices, the data science community learned how valuable it is to focus on usability. So the values that scikit-learn holds high are becoming more wide-spread.

Documentation should be accessible and well-organized. Interface is important. Remember the scikit-learn interface: three methods get you nearly all the way there. Your functionality doesn’t matter if your users can’t figure out how to do the thing they really want to do.

And last but not least, you need to play nice with the

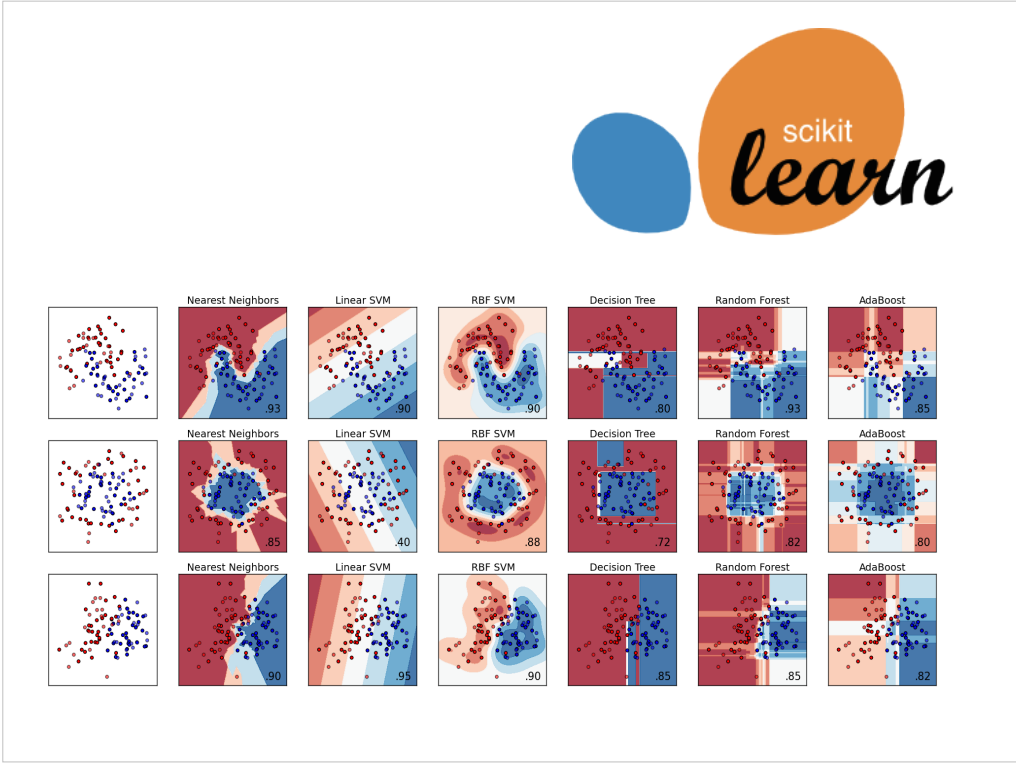


There’s actually a lot more machine learning in Python today in addition to what’s in scikit-learn. There is a lot of tools for deep learning, in particular tensor-flow and theano for the low-level primitives, and then keras and lasagne for higher-level interfaces.

There are also tools for building probabilistic models, like pymc, edward, stan and pommegranade.

All of these are important tools, but they lie a bit beyond what a non-expert can easily use: these are very powerful tools, but they require a lot of knowledge from the user.

So definitely a lot of machine learning there, not so



But let's talk more about what scikit-learn can do for you. Scikit-learn implements tens of models for classification, regression, feature extraction, feature selection, signal decomposition and so on.

And scikit-learn not only provides the models, it also provides the tools to evaluate and select models, and the glue to make them work together.

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

Two of the most helpful tools are GridSearchCV, which implements grid-search with cross-validation, and pipeline, which allows chaining together multiple algorithms.

Both of these tools are implemented as so-called estimator classes themselves, meaning they have the same fit, predict and transform methods as the algorithms themselves!

```
github.com/scikit-learn-contrib/scikit-learn-contrib
```

Because people like scikit-learn, but we're relative conservative about what we want to include in the library, many people built their own extensions for specific models. It's really easy to built your own scikit-learn extensions and it's something we highly encourage.

To give more visibility to these projects we created the "scikit-learn-contrib" umbrella, a github organization to which people can submit their projects. You can find the projects on the scikit-learn-contrib org, and a description of how it works at the project URL I show here.

So enough about the present. Let’s talk about the future. First I want to talk about the near future. Actually the very near future, like, say, tomorrow. Or maybe Tuesday.

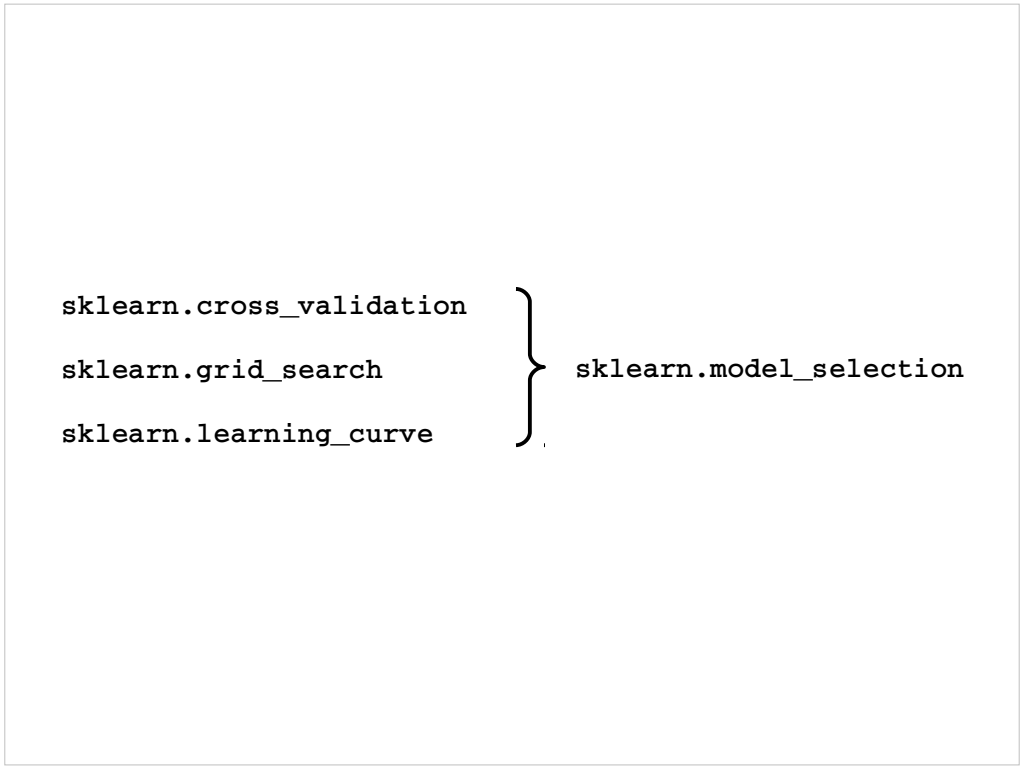
Because early next week, scikit-learn 0.18 will come out!

And it will have some awesome new features.

We already have a release candidate out, which you can install via pip (if you normally use conda, make sure you uninstall the conda version first). So if you're impatient and want to try it today, run the line up there.

Otherwise, wait until next week and you'll get 0.18 with pip, conda and conda-forge.

For several reasons 0.18 took us a bit longer than we like our releases to take. It's been about a year since



One of the things that will affect all users is our reorganization of the model selection tools.

From 0.18 on, everything relating to cross-validation, grid search and model selection will be inside the new model_selection module. Previously, the organization was a bit odd, with RandomizedSearchCV being in the grid_search module. Another reason we did the move was because we wanted to do a couple of API changes that wouldn’t have been possible otherwise.

So change all your imports from cross_validation, grid_search and learning_curve to model_selection

```
results = pd.DataFrame(grid_search.results_)
```

	mean_test_score	param_C	param_gamma	params	...	split2_test_score	split3_test_score	split4_test_score	std_test_score
0	0.37	0.001	0.001	{'C': 0.001, 'gamma': 0.001}	...	0.36	0.36	0.38	0.01
1	0.37	0.001	0.01	{'C': 0.001, 'gamma': 0.01}	...	0.36	0.36	0.38	0.01
2	0.37	0.001	0.1	{'C': 0.001, 'gamma': 0.1}	...	0.36	0.36	0.38	0.01
3	0.37	0.001	1	{'C': 0.001, 'gamma': 1}	...	0.36	0.36	0.38	0.01
4	0.37	0.001	10	{'C': 0.001, 'gamma': 10}	...	0.36	0.36	0.38	0.01

One of the changes to the grid-search that we did was restructuring the results of grid-search, which are now stored in the cv_results_ attribute. This is now a dictionary where each value is an array with an entry for every parameter setting. From this format, you can easily convert it to a pandas dataframe, which makes it much easier to use.

It will also (at least probably) contain training scores, training time and prediction time.

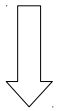
```
labels → groups

n_folds → n_splits
```

We also did a bit of renaming of some of the cross-validation related functions and classes.

There are some cross-validation splitters which are able to handle groups of correlated data, like multiple samples coming from the same patient or the same customer. Previously they were called LabelKFold, LeavePLabelOut and LabelShuffleSplit, where label meant the group label. That was pretty confusing, so we renamed them to GroupKFold, GroupShuffleSplit and so on. We also renamed the labels parameter to groups.

```
from sklearn.cross_validation import KFold
cv = KFold(n_samples, n_folds)
for train, test in cv:
    ...
```



```
from sklearn.model_selection import KFold
cv = KFold(n_folds)
for train, test in cv.split(X, y):
    ...
```

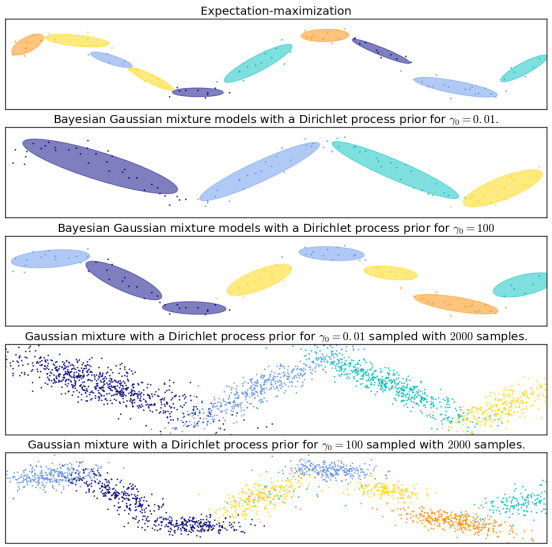
The biggest change we did to the model selection was to the cross-validation splitters. This is a bit technical, so if you haven’t worked with scikit-learn a lot, you might not have seen this.

Scikit-learn has classes that specify how to do cross-validation splits. So far, these were more or less generators, which needed to know how much data there was before you could instantiate them.

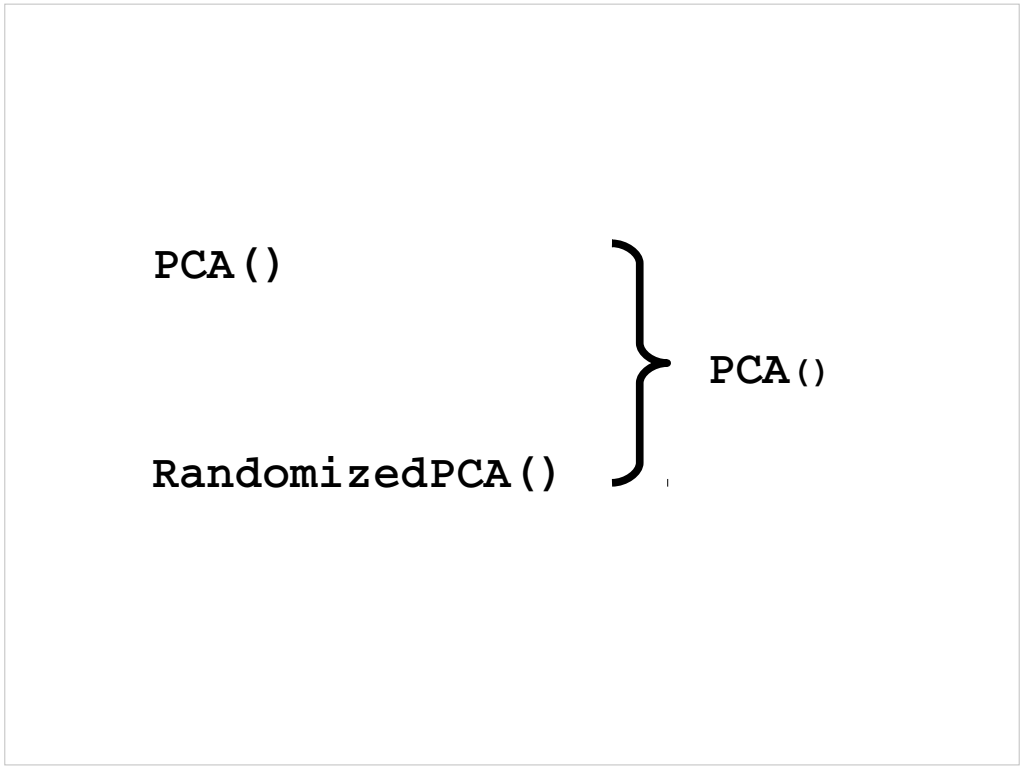
Now, at instantiation you don’t need to know about the data. Only when you actually want to split some particular data, the generators are created.

That allows us to do more complex nested cross-

```
from sklearn.mixture import GaussianMixture
from sklearn.mixture import BayesianGaussianMixture
```

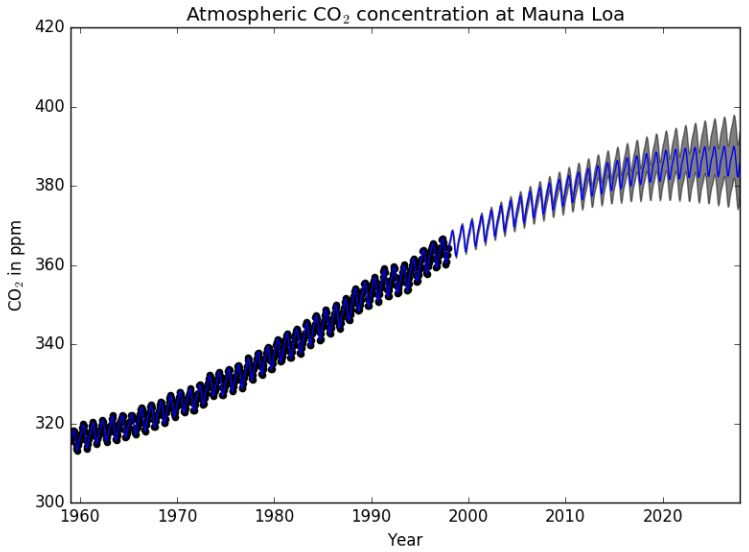


One of the more exciting additions is two new classes for Gaussian Mixture models, called GaussianMixture and BayesianGaussianMixture. The old classes are deprecated. The new classes have a simpler interface, are numerically more stable and have many bugfixes. The GaussianMixture class implements standard Gaussian mixtures with expectation-maximization, while the BayesianGaussianMixture class implements – surprise – a Bayesian model, either with a dirichlet prior, or with an infinite dirichlet process prior.



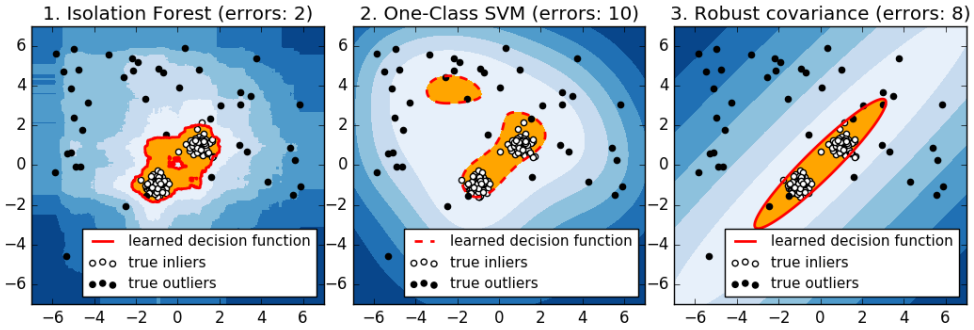
Another change that I think is quite cool is that we collapsed the PCA class and the Randomized PCA class, and now it's just called PCA, and it will automatically use the fasted algorithm based on the shape of your data and how many components you want.

Gaussian Process Rewrite



One of the most exciting changes for me is the rewrite of the Gaussian process module. There are two new classes, GaussianProcessRegressor and GaussianProcessClassifier, and a whole bunch of different kernel functions that you can use to create new custom kernels. Now it is also much easier to learn the parameters of complex kernels with scikit-learn. You see an example of that here, with the classical Mauna Loa CO2 dataset.

Isolation Forests



Another great new addition are Isolation Forests.

Isolation Forests are a outlier detection method. In case you're unfamiliar, outlier detection tries to find data that looks different from "normal". This can be used, for example, for fraud detection in finance or for intrusion or failure detection in distributed systems.

Here you can see Isolation forest compared to two methods that we had previously.

Isolation forests are great because they work very well without a lot of parameter tuning, and they are very flexible. Recently I was at a conference where there

Play

```
from sklearn.neural_network import MLPClassifier
```

Work

```
import keras
```

Another thing many people have been waiting for is the multilayer perceptron, also known as vanilla neural network, which is implemented in MLPClassifier and MLPRegressor.

This is meant as an easy way for you to dip your toes into deep learning. However, because scikit-learn doesn’t support GPUs, this is not a high performance implementation, so it’s more just for playing around. However, there are great deep learning libraries out there. So if you are serious about deep learning, definitely try out keras or maybe even tensorflow directly.

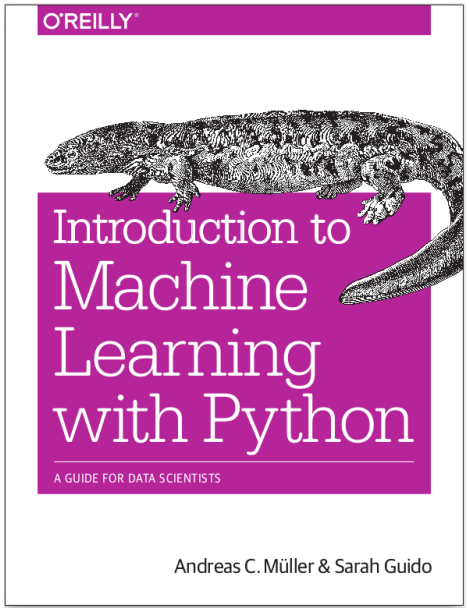
```
pipe = Pipeline([('preprocessing', StandardScaler()),
                 ('classifier', SVC())])

param_grid = {'preprocessing': [StandardScaler(), None]}

grid = GridSearchCV(pipe, param_grid)
```

The last change that I want to mention is that we improved how pipelines and grid-search interact. You can now search over the steps in a pipeline! This means you can not only search the parameters of a classifier, for example, you can also search over which classifier, or which preprocessing to use!
I'm quite excited about that.

This list of additions is not exhaustive, and if you are interested in knowing about all the changes, check out the whatsnew on the scikit-learn website.



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

This book is different, there are no formulas, but a lot of code. It's actually using the 0.18 release, so it contains a lot of the new stuff I just talked about.

The goal of the book is to enable you to apply machine learning, without wading through a lot of theory first, and just dive in, but still do the right thing and be successful

The e-book comes out next week, the paper copy in two weeks. If you're impatient, you can get an early

(further) Future

So now, lets talk to the somewhat further-out future.
What are the plans and ideas we have?

- One thing that I think will help users a lot is better support of feature names or column names. Scikit-learn builds on numpy, which doesn’t know about feature names. However in many applications, the features or column have semantic names that are important to keep track of, like age, gender, time spend online and so on. You might have stored these in a pandas dataframe, where each column has an associated header.
- Because scikit-learn doesn’t handle these names, it’s sometimes tricky how the original features relate to the output, in particular in complex pipelines

```
from __future__ import sklearn.plotting
```

We are also working on more integrated plotting. There are some plots, like confusion matrices, or model coefficients, feature importance or the result of a grid-search, that you need all the time. We will include some of these directly in scikit-learn, to make machine learning even more interactive.

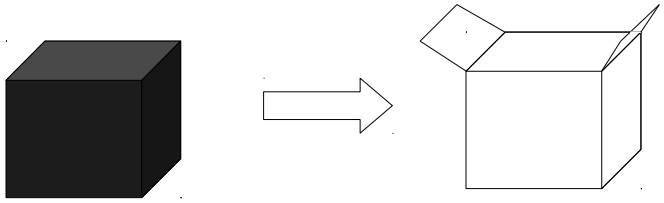
```
from __future__ import AutoClassifier
```

Another direction that I think will be quite important is more automation in machine learning. Scikit-learn embraces the python paradigm “explicit is better then implicit”.

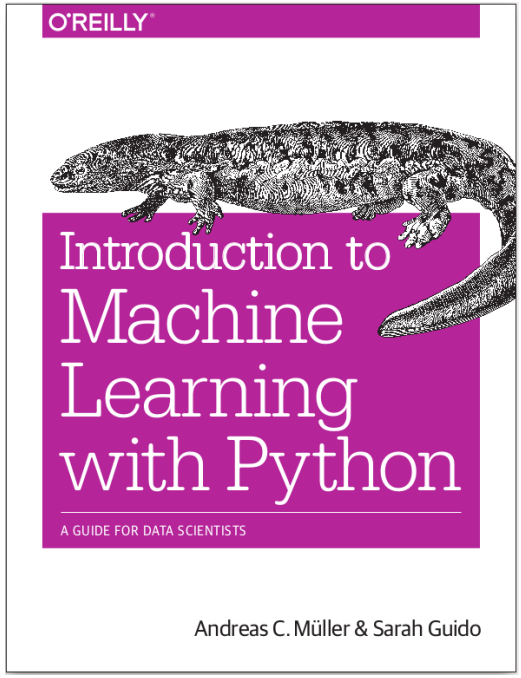
This behavior is more tuned toward expert users, that want fine-grained control over whats happening. For beginners, it might be more important to get something reasonable done quickly, with little domain expertise and before reading some 366 page book.

I want the machine learning to get out of the way, so that people can focus on their data and the application, instead of tuning parameters and trying

More Transparency



- Automation helps creating better black box models faster, which is great, but sometimes you really want to open up the black box and understand what's going on. Why did a particular point get classified in a certain way? Which features are important? What kinds of confusions happen? Can we maybe even replace a complex model by something a human can fully understand? There is no way to understand a random forest with hundreds of tree, but maybe a small rule-based system can work nearly as well.
- Unfortunately there is very little work on this in the machine learning community so far, but it's



amueller.github.io

@amuellerml

@amueller

t3kcit@gmail.com

Alright, as I said, buy my book.
You can also find a lot of free videos and notebooks about machine learning and scikit-learn on my website. All the code from the book is BSD-licensed and on github.

I'll hang around for the rest of the day if you want to chat about machine learning.
Or if you want to take a selfie, that seems to be even more popular than machine learning these days.

Alright, thank you for your attention