Practical Aspects of Machine Learning

Markus Hinsche, Senior ML Engineer, Freelance, datascienceretreat.com 2023

Agenda

We'll talk about:

- A workflow for developing machine learning models and putting them in production
- Solving common problems in machine learning projects, such as modularization of code, managing configuration, hyperparameter optimization, and distributed learning, and working with big data
- Making available trained models through scalable and secure web services

Topics we can discuss: mixed-precision, multi-gpu training, TPU training, scale up model training, model ensembling, hyperparameter tuning, search space

Tools/Programs: ls, git, ssh, scp, vim, PDB, docker, docker-compose, Postman/Insomnia

Download slides and code

Download this repository to be able to run code examples:

```
$ git clone https://github.com/markus-hinsche/tut-productive-ml.git
```

Slides are online at:

https://markus-hinsche.github.io/tut-practical-aspects-of-ml/

whoami

- Digital Health Entrepreneur building Nailvision https://nailvision.de
- Did Machine Learning Freelancer before (website, github)
  - Hands-on Implementation and Consoluting (Machine Learning)
- Projects: Welthungerhilfe, Charité radiology (both computer vision), …
- Strengths:
  - Startups (since 2015)
  - Python (since 2009)
  - Deep Learning (since 2018)
  - Software development best practices (testing, pairing, clean code)
  - Mentoring

Why ML in production?

"87% of data science projects never make it to production"

[https://stackoverflow.blog/2020/10/12/how-to-put-machine-learning-models-into-production/]

"Model deployment is more and more expected from a Data Scientist"

[Jose]

What is the difference?

- code for researching models
- code for deploying models into production

Iris dataset

In our first example in `step1`, we will make use of the classic "Iris" machine learning dataset from 1936.

`step1/iris.data` contains the data in CSV format. Note that the first line for column names is missing:

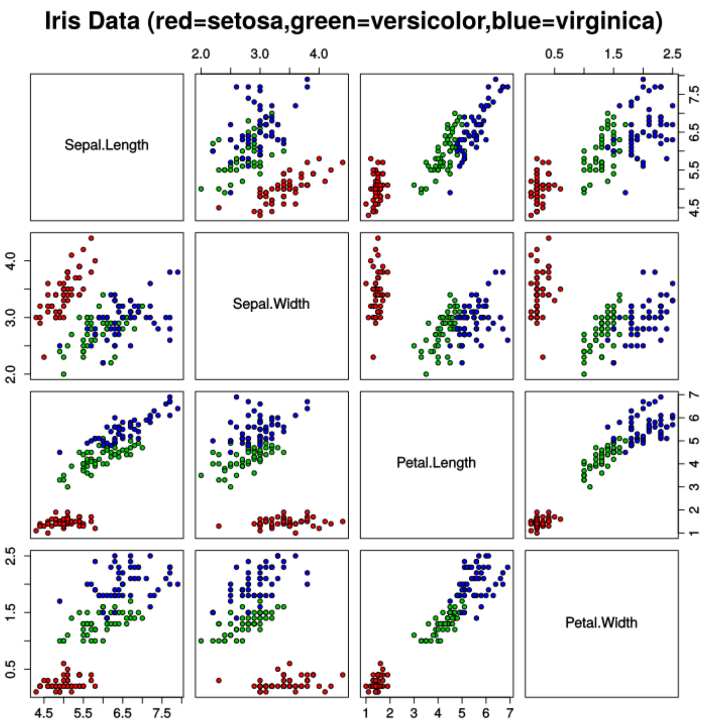
```
5.2,3.5,1.5,0.2,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.6,3.0,4.5,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
...
```

The columns are:

- `sepallength`
- `sepalwidth`
- `petallength`
- `petalwidth`
- `species`

Iris as a classification problem

We want to implement a classifier that predicts the species based on the four attributes.



Hyperparameter search

The search for optimal hyperparameters is a very common task in machine learning. In our example, doing the search by hand was still easy, but more complex models and pipelines often have a big number of these parameters.

There's different options to perform the search, scikit-learn itself implements a number of these algorithms. One of these algorithms is called *grid search*.

> ⚠️ **Note:** The scikit-optimize package allows us to search for hyperparameters using *Bayesian optimization*.

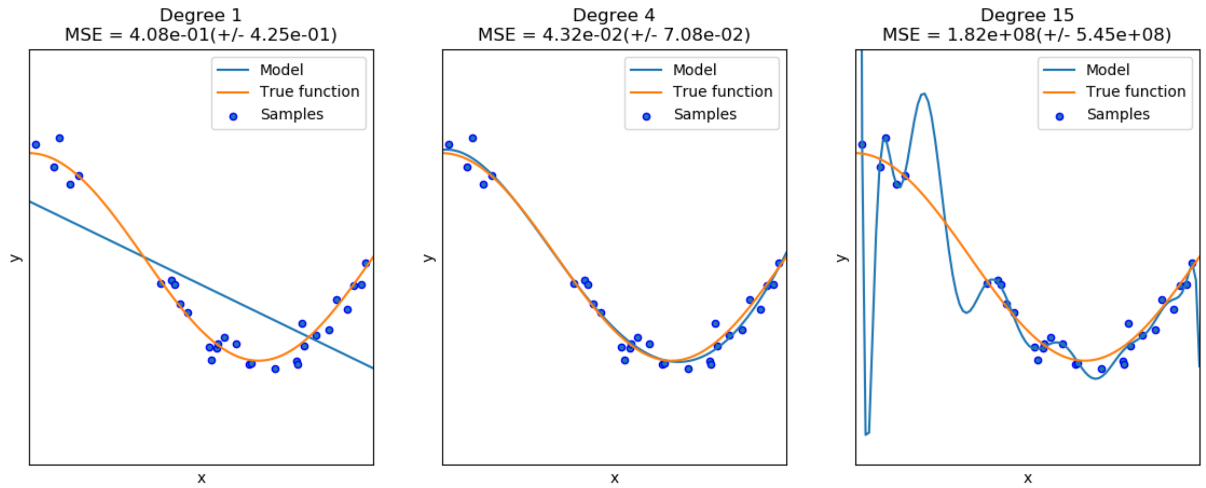
GridSearch in sklearn

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
import pandas as pd
iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 1, 10, 100, 1000]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(iris.data, iris.target)
pd.DataFrame(clf.cv_results_)
```

Grid search: Interpreting the output

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
3	0.000778	0.000188	0.98	0.989990	100
4	0.000860	0.000221	0.98	1.000000	1000
2	0.000691	0.000198	0.96	0.972828	10
1	0.000675	0.000199	0.95	0.955667	1
0	0.001668	0.000427	0.87	0.872882	0.1

A low value for C means stronger regularization. The effect here is "underfitting", which is characterized by a low train and low test score.



Hyperparameter search: keras

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/chapter13_best-practices-for-the-real-world.ipynb

Hyperparameter search: Network hyperparameters (2)

✍️ Exercise

What hyperparameters could we include in the search?

Deploying webservice to production

- web service
- uvicorn

Exercise: Use FastAPI to build a web interface

Solution

```
pip install -U pip
pip install -r step1/requirements.txt
```

Prepare `deploy_iris.py`

```
uvicorn deploy_iris:app --reload --log-level=debug
```

Deploying models to production

We will now walk through these steps:

- Set up an AWS account (if you don't have one yet)
- Set up an EC2 micro instance
- SSH into the instance to install dependencies and run the web service

Register and sign in to AWS console

Go to the aws.amazon.com and click *Sign In to the Console*. If you don't have an account yet, you can register by clicking *Create a new AWS account*. It will ask you to put your credit card information and telephone number. However, the micro instance that we use will cost you nothing. (Just make sure you remove it later. It's free for one year.)

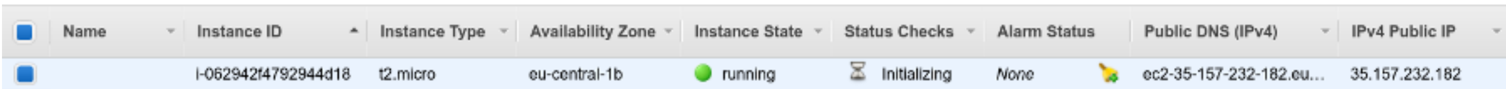
Launch a new EC2 instance

Under the *Services* drop-down, click *EC2*. If you're not using EC2 yet, it will say *0 Running Instances*. Click this link and then click the blue *Launch Instance* button to set up a micro instance.

In the first step, choose *Ubuntu Server 18.04 LTS* as the *Amazon Machine Image (AMI)*. In the second step, choose the *t2.micro* instance type. You can then click the blue *Review and Launch* button.

Lastly, when you now click *Launch*, it will display a pop-up with which you can *Create a new key pair*. The name doesn't matter, maybe choose something like *DSR Key* and click *Download Key Pair*. This will download a file with the ending *.pem*. Keep this file in a safe place; you will need it to log in to your instance.

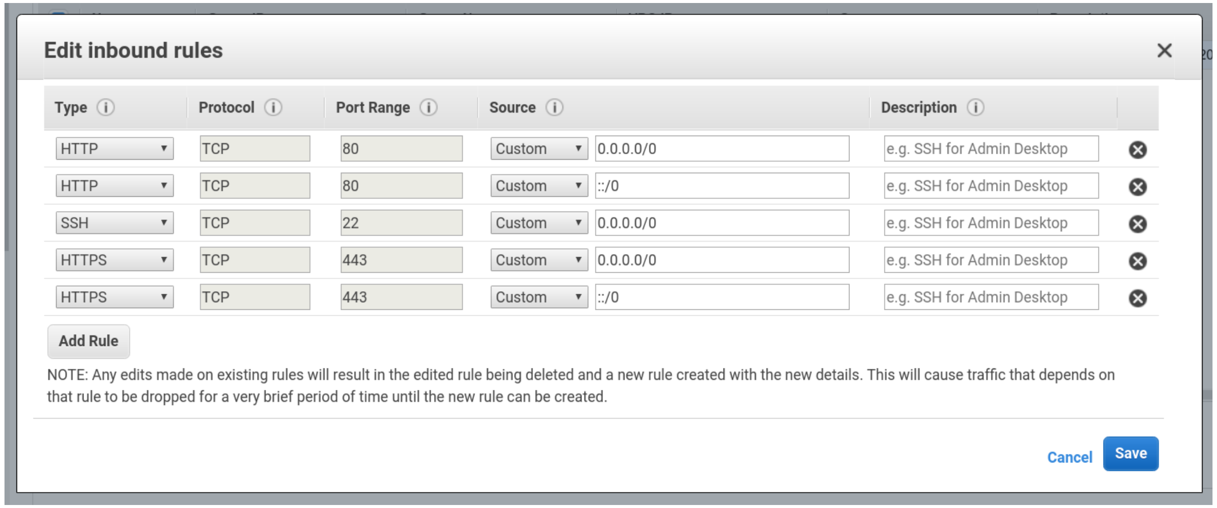
After launching you should be able to view your instance in the overview.



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
	i-062942f4792944d18	t2.micro	eu-central-1b	running	Initializing	None	ec2-35-157-232-182.eu...	35.157.232.182

Open up additional ports to your EC2 instance

In the overview of instances, look at the bottom pane and navigate to *Security groups* and click *launch-wizard-1*. You will now find yourself on a screen that lists your security groups. On the bottom third of the screen, you'll see a tab called *Inbound*. Click *Edit* and then *Add Rule*. From the *Type* dropdown, choose *HTTP*. Then, add another rule and choose *HTTPS*. You can leave the defaults as is and confirm by clicking the blue *Save* button. We have now opened up ports 80 and 443 on our EC2 instance. This will be useful later on, when we access our web service through these ports.



SSH into your EC2 instance

Your EC2 instance has a public IPv4 address. In this example it's 35.157.232.182. Make sure you use your own IP address and not mine in the following examples:

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
		i-062942f4792944d18	t2.micro	eu-central-1b	running	Initializing	None	ec2-35-157-232-182.eu...	35.157.232.182

You may already have an SSH client installed. Try:

```
$ ssh -V
OpenSSH_7.6p1 Ubuntu-4ubuntu0.2, OpenSSL 1.0.2n  7 Dec 2017
```

If you do, you can now log in to your EC2 instance by issuing this command:

```
$ ssh -i path/to/DSRKey.pem ubuntu@35.157.232.182
```

Once you have successfully logged in, you should see a prompt like this one:

```
ubuntu@ip-172-31-42-9:~$
```

SSH into your EC2 instance: possible issues

If you're on Ubuntu and you don't have `ssh` installed, you can install it like so:

```
$ sudo apt install openssh-client
```

If you encounter an error saying: *It is required that your private key files are NOT accessible by others*, then run this command (once) before you run `ssh`:

```
$ chmod 600 path/to/DSRKey.pem
$ ssh -i path/to/DSRKey.pem ubuntu@35.157.232.182
```

On newer versions of Windows, the OpenSSH client should be installed by default in `C:\Windows\System32\OpenSSH`. If it's not installed, try navigating to *Settings App* > *Apps* > *Settings & Apps* > *Manage Optional Features* > *Add Feature* and select the *OpenSSH Client*.

Once you have successfully logged in, you should see a prompt like this one:

```
ubuntu@ip-172-31-42-9:~$
```

Install dependencies on our EC2 instance

First, let's install a couple of dependencies that we'll need:

```
$ sudo apt update && sudo apt upgrade
$ sudo apt install python3-venv python3-dev build-essential
```

Next, we'll check out this repo from Github and create a virtualenv inside of it:

```
$ git clone https://github.com/markus-hinsche/tut-productive-ml.git
...
$ python3 -m venv tut-productive-ml
$ cd tut-productive-ml
$ source bin/activate
```

The last command will change our prompt to look something like this:

```
(tut-productive-ml) ubuntu@ip-172-31-42-9:~/tut-productive-ml$
```

We can now proceed to install Python dependencies:

```
$ pip install -U pip
$ pip install -r step1/requirements.txt
...
Successfully installed ...
```

Use Gunicorn to serve the web service

```
$ pip install gunicorn
$ # TODO make sure you have a trained model
$ gunicorn step4.deploy_gentle_iris:app -k uvicorn.workers.UvicornWorker -b 0.0.0.
```

Try to navigate your browser to this URL, and you'll notice that it's not yet working as expected. There's no response: http://35.157.232.182/predict?sepallength=6.3&sepalwidth=2.5&petallength=4.9&petalwidth=1.5

The problem is that we've asked Gunicorn to serve on port 8080, but the standard HTTP port which our browser connects to is port 80. The issue is that only the `root` user can bind to port 80, and we don't want to run our application as root. What can we do? We can use iptables:

```
$ sudo iptables -A PREROUTING -t nat -p tcp --dport 80 -j REDIRECT --to-ports 8080
```

But is it fast? Use `locust` to find out!

A tool for benchmarking websites is `locust`. Let's install and simulate many requests against our web service ⇶

Let's start to profile an app that runs locally

```
pip install locust
locust -f step4/benchmark.py
```

Headless mode can be handy, e.g., when profiling our AWS setup:

```
locust -f step4/benchmark.py --headless --host http://0.0.0.0:8000 -u 1000 -r 50
```

Ressource: https://calmcode.io/locust/introduction.html

Locust output (in headless mode)

```
(.venv)$ locust -f step4/benchmark.py --headless --host http://0.0.0.0:8080 -u 20 -r 10
[2022-03-15 09:47:55,518] MarkusMBP16/INFO/locust.main: No run time limit set, use CTRL+C to interrupt
[2022-03-15 09:47:55,518] MarkusMBP16/INFO/locust.main: Starting Locust 2.8.3
 Name                                                                           # reqs      # fails  |     Avg     Min     Max  Median  |   req/s failures/s
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregated                                                                          0     0(0.00%)  |       0       0       0       0  |    0.00    0.00

[2022-03-15 09:47:55,520] MarkusMBP16/INFO/locust.runners: Ramping to 20 users at a rate of 10.00 per second
[2022-03-15 09:47:56,526] MarkusMBP16/INFO/locust.runners: All users spawned: {"WebsiteUser": 20} (20 total users)
 Name                                                                           # reqs      # fails  |     Avg     Min     Max  Median  |   req/s failures/s
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 GET /predict/?sepallength=6.3&sepalwidth=2.5&petallength=4.9&petalwidth=1.5        20     0(0.00%)  |      57      24      86      57  |    0.00    0.00
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregated                                                                         20     0(0.00%)  |      57      24      86      57  |    0.00    0.00

 Name                                                                           # reqs      # fails  |     Avg     Min     Max  Median  |   req/s failures/s
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 GET /predict/?sepallength=6.3&sepalwidth=2.5&petallength=4.9&petalwidth=1.5        34     0(0.00%)  |      39      12      86      34  |   10.00    0.00
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregated                                                                         34     0(0.00%)  |      39      12      86      34  |   10.00    0.00

KeyboardInterrupt
2022-03-15T08:48:00Z
[2022-03-15 09:48:00,113] MarkusMBP16/INFO/locust.main: Shutting down (exit code 0)
 Name                                                                           # reqs      # fails  |     Avg     Min     Max  Median  |   req/s failures/s
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 GET /predict/?sepallength=6.3&sepalwidth=2.5&petallength=4.9&petalwidth=1.5        38     0(0.00%)  |      36      12      86      24  |    8.54    0.00
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregated                                                                         38     0(0.00%)  |      36      12      86      24  |    8.54    0.00

Response time percentiles (approximated)
 Type     Name                                                                                  50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|--------------------------------------------------------------------------------|---------|------|------|------|------|------|------|------|------|------|------|------|
 GET      /predict/?sepallength=6.3&sepalwidth=2.5&petallength=4.9&petalwidth=1.5               28     51     61     67     78     84     87     87     87     87     87     38
--------|--------------------------------------------------------------------------------|---------|------|------|------|------|------|------|------|------|------|------|------|
 None     Aggregated                                                                            28     51     61     67     78     84     87     87     87     87     87     38
```

Profiling our web app

To see where time is being spent on each request, let's install the py-spy Python profiler:

```
$ cd tut-productive-ml
$ source bin/activate
$ pip install py-spy
```

We keep the Gunicorn process running in another window. Note that Gunicorn prints out the process id (PID) of its worker when it starts up. It looks something like this:

```
[2020-11-18 16:12:05 +0000] [5883] [INFO] Booting worker with pid: 5883
```

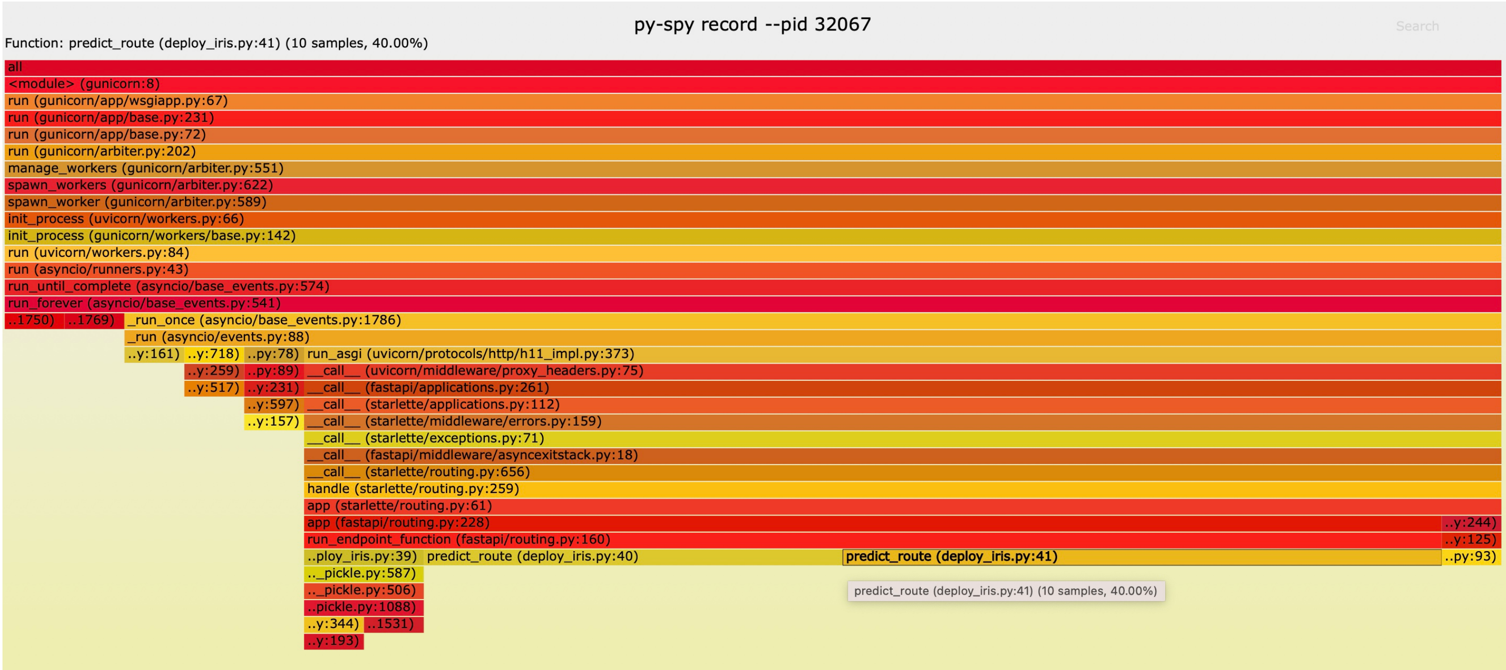
We can now use `py-spy` to attach to the Gunicorn worker and see what it's doing:

```
$ sudo su
$ source bin/activate
$ py-spy record --pid 5883  # replace with the actual Gunicorn PID you just saw
```

Profiling our web app (2)

You should see output similar to `top`. The web server isn't doing much right now, but we can send another round of requests to it to see what it's doing when it's busy. In a new terminal window, run another multiple requests using `locust` and switch back to the `py-spy` window to see what's happening.

```
$ locust -f step4/benchmark.py --headless --host http://0.0.0.0:8080 -u 20 -r 10
```



Profiling our web app (3)

Alternatively using py-spy's **top** command:

```
$ py-spy top --pid 5883  # replace with the actual Gunicorn PID you just saw
```

```
Collecting samples from 'pid: 5883' (python v3.6.7)
Total Samples 3300
GIL: 0.00%, Active: 100.00%, Threads: 1

  %Own   %Total  OwnTime  TotalTime  Function (filename:line)
 12.00%  12.00%    1.77s     1.77s   do_execute (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/sqlalchemy/engine/default.py:536)
  9.00%   9.00%   0.730s    0.730s   connect (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/sqlalchemy/engine/default.py:437)
  3.00%   3.00%   0.420s    0.420s   write (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/gunicorn/util.py:304)
  3.00%   5.00%   0.460s     1.04s   instances (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/sqlalchemy/orm/loading.py:65)
  2.00%   9.00%   0.200s     3.70s   _compiler_dispatch (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/sqlalchemy/sql/visitors.py:90)
  2.00%   2.00%   0.260s    0.260s   do_close (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/sqlalchemy/engine/default.py:492)
  2.00%   2.00%   0.100s    0.100s   handle_request (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/gunicorn/workers/sync.py:184)
  2.00%   8.00%   0.370s     1.89s   read (/home/ubuntu/dsr-2019/lib/python3.6/site-packages/palladium/persistence.py:421)
```

What we did so far with our EC2 instance:

- Set up the EC2 instance using the AWS console
- Opened up additional network ports
- Logged in using SSH and cloned this repo from Github
- Installed system dependencies (`apt install`) and Python project dependencies (`pip install`) on our EC2 instance
- Used Gunicorn to serve the FastAPI web service
- Benchmarked (`locust`) and profiled (`py-spy`) our web service

What we'll need to do next:

- We're running the `gunicorn` process in our terminal in foreground. If we close the terminal window, Gunicorn will stop working. To fix this issue, we'll use Supervisor to start up the Gunicorn process and keep it running.
- Anyone who knows our web service's address can now use it. What's worse, we're sending the data to predict for and the results over the wire, without encryption. We'll use HTTPS instead of HTTP, and use basic access authentication to prevent eavesdropping and unauthorized use.

Configure Supervisor to keep Gunicorn running

Use `Ctrl+C` to shut down the `gunicorn` process (or alternatively, close the terminal window in which you're running it).

On the server, create this supervisor config file and save it as `/home/ubuntu/tut-productive-ml/supervisor.conf`:

```
[program:webservice]
command=/home/ubuntu/tut-productive-ml/bin/gunicorn -k uvicorn.workers.UvicornWork
directory=/home/ubuntu/tut-productive-ml/
user=ubuntu
```

You can find out more about [Supervisor's configuration files in the documentation](#).

Install and run Supervisor

We install Supervisor with `apt install`:

```
$ sudo apt install supervisor
```

And we link the configuration file that we just wrote into a directory where Supervisor will pick it up:

```
$ sudo ln -s /home/ubuntu/tut-productive-ml/supervisor.conf /etc/supervisor/conf.d
```

Now let's restart Supervisor and check if it's running our process:

```
$ sudo /etc/init.d/supervisor restart
$ sudo supervisorctl status
webservice                       RUNNING   pid 6630, uptime 0:00:22
```

At this point, we can log out from the server and the web service will keep running! What's more, Supervisor will start up our web service when the EC2 box is ever restarted, and it will restart our web service should it ever crash with a Python exception or the like.

Set up secure communication to our web service (HTTPS)

Here's what we'll do to secure our web service with TLS/HTTPS:

- Use certbot/Let's Encrypt to obtain an HTTPS certificate
- Use the NGINX web server to handle encryption and forward to Gunicorn
- Use Docker and docker-compose to wire it all together

Install Docker and docker-compose

Installing Docker involves adding Docker's proprietary package repository to our Ubuntu 18.04 system and installing the `docker-ce` package from there. Here's the steps:

```
$ sudo apt install apt-transport-https ca-certificates curl software-properties-co
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubun
$ sudo apt update
$ sudo apt install docker-ce
```

To install `docker-compose`, make sure you're inside your virtual environment and run this:

```
$ source bin/activate
$ pip install docker-compose
```

Setting up a domain

Before we can ask *Let's Encrypt* for an SSL/TLS certificate, we'll have to set up a domain for our EC2 instance. If you have your own domain, chances are that you can easily set up a subdomain that points to the EC2 instance. Here's a guide on how to set up a subdomain with Namecheap.

Here's the *A Record* that I set up such that `dsr2021test.markushinsche.de` now points to the IP of my EC2 instance, which is `35.157.232.182`:

Type	Host	Value	TTL
A Record	@	46.105.124.140	30 min
A Record	dsr2019test	35.157.232.182	1 min

After this, we should be able to access our web service using the new name: http://dsr2021test.markushinsche.de/alive

Prepare config files before we obtain the certificate

Open up `step8/init-letsencrypt.sh`. At the top of the file, you'll find the definition of domains. Change this to match the subdomain that you just set up:

```
domains=(mysubdomain.markushinsche.de)
```

Similarly, replace all occurrences of `mysubdomain.markushinsche.de` with your actual domain name inside of `step8/data/nginx/app.conf`.

Remember the `iptables` rule that we used to forward ports? We need to get rid of it, because we'll have NGINX serve port 80:

```
$ sudo iptables -L -t nat
...
$ sudo iptables -D PREROUTING -t nat -p tcp --dport 80 -j REDIRECT --to-ports 8080
$ sudo iptables -L -t nat
...
```

Obtain the certificate from *Let's Encrypt*

Now we're ready to run the `init-letsencrypt.sh` script. You have to do this as the root user and while inside the virtual environment:

```
$ sudo su  # become superuser
$ source bin/activate  # activate virtualenv
$ cd step8
$ ./init-letsencrypt.sh
```

(Make sure you log out as root whenever you no longer need it.)

If successful, this will print something like this:

```
- Congratulations! Your certificate and chain have been saved at:
  /etc/letsencrypt/live/mysubdomain.markushinsche.de/fullchain.pem
  Your key file has been saved at:
  /etc/letsencrypt/live/mysubdomain.markushinsche.de/privkey.pem
  Your cert will expire on 2022-05-20. To obtain a new or tweaked
  version of this certificate in the future, simply run certbot
  again. To non-interactively renew *all* of your certificates, run
  "certbot renew"
- Your account credentials have been saved in your Certbot
  configuration directory at /etc/letsencrypt. You should make a
  secure backup of this folder now. This configuration directory will
  also contain certificates and private keys obtained by Certbot so
  making regular backups of this folder is ideal.
```

It just works, but how?

When open the following page in your browser (use your own subdomain!), note that your browser will show a secure connection now: https://mysubdomain.markushinsche.de/predict?sepallength=6.3&sepalwidth=2.5&petallength=4.9&petalwidth=1.5

Let's have a look at the individual components and pieces of configuration that made this work.

`docker-compose` is a tool that's maybe similar to Supervisor, but instead of managing processes on your host, it manages Docker containers. Run `sudo docker-compose ps` from within `step8` to see what containers it's running.

`step8/docker-compose.yml` is set up to run an `nginx` container and a `certbot` container. Note how NGINX is set up to handle requests to ports 80 and 443. The NGINX configuration itself lives in `step8/data/nginx/app.conf`, and it sets up NGINX to act as a proxy for our application (`app_server`).

`init-letsencrypt.sh` is a shell script that uses the `certbot` container to obtain the certificate and save it.

An update to our Supervisor configuration

Take a look at `step8/supervisor.conf`. In addition to our `program:webservice` section, we now also have `program:docker-compose`:

```
[program:docker-compose]
command=/home/ubuntu/tut-productive-ml/bin/docker-compose up
directory=/home/ubuntu/tut-productive-ml/step8/
```

Let's link this file into the right place such that Supervisor picks it up:

```
$ sudo rm /etc/supervisor/conf.d/supervisor.conf
$ sudo ln -s ~/tut-productive-ml/step8/supervisor.conf /etc/supervisor/conf.d/
```

We shut down the running instance of Supervisor and `docker-compose` before we start up Supervisor again (as root and inside the active virtual environment):

```
$ cd step8
$ docker-compose down
$ /etc/init.d/supervisor stop
$ /etc/init.d/supervisor start
```

Require authentication for our web service

Now that our connection is secure from eavesdropping, how can we restrict access such that only people with a valid username and password can use it?

Programmatic use of the web service

Our prediction web service will typically be called from another program, be it a smartphone app or a website. This is how you would use the final web service from Python:

```python
import requests

data = [
    {'sepallength': 6.3, 'sepalwidth': 2.5,
     'petallength': 4.9, 'petalwidth': 1.5},
    ]
result = requests.get(
    'https://mysubdomain.markushinsche.de/predict',
    json=data,
    auth=('myuser', 'mysecret'),
    ).json()

print(result)
# result is a Python dictionary:
# {'metadata': {'status': 'OK', 'error_code': 0}, 'result': ['Iris-virginica']}
```

End

- https://github.com/scikit-learn/scikit-learn

- https://keras.io/

- mail@markushinsche.de

- linkedin.com/in/markushinsche
- twitter.com/markus_hinsche
- markushinsche.de
- https://nailvision.de

Thanks to Daniel Nouri for initial versions of the slide deck

Images

- scikit-learn
- Machine Learning course on Coursera