AI & Robotics: Spring Break Assignment

Spring 2019

Introduction

This is an optional, but highly recommended assignment. In this task you are going to be working with the Fast AI library. More specifically the 0.7 version of the library. This document includes (links to) information how to install this library onto your system. It is followed by the assignment.

The functionality we are going to be using from Fast.AI is mostly from the scikit-learn library for classical machine learning. Scikit-learn and Fast AI are used in the notebooks of Week 7.

[EXTRA] Background on frameworks

To give you some extra information, we have added an additional presentation about Frameworks on GitHub. The document will be updated regularly, as we're going to explore more frameworks and libraries in the coming weeks: Linkage

Installing Fast.AI

The installation guide for fastai 0.7 can be found at:

https://forums.fast.ai/t/fastai-v0-7-install-issues-thread/24652

Important remarks

- Make sure to create the CPU environment if your GPU is not supported by your operating system.
- Always activate the Fast.AI environment before running notebook experiments!

```
conda activate fastai
```

Or

```
conda activate fastai-cpu
```

It is highly recommended to create a symlink in the folder where you are storing your notebooks to the folder where you cloned the fastai git repository:

```
$ ln -s <path>/<to>/fastai/old/fastai fastai
```

This way you can import fastai in your notebooks easily like this:

```
from fastai.imports import *
from fastai.structured import *
```

This is done because getting the old version of the Fast.AI library like this does not add the library to your PATH.

Starting up your jupyter notebook

```
$ source anaconda.bash
$ conda activate fastai(-cpu)
$ jupyter-notebook
```

Assignment

We will be using a modified version of the Fifa19 Kaggle dataset. We have edited and removed some attributes to make the data more manageable for this assignment. We will return to the uncleaned dataset in a later stage of the course. The cleaned dataset can be found in the accompanying .tar.gz file.

The dataset contains information about every player in Fifa19. Information like:

- Wage
- Preferred_Foot
- Skill_Moves
- Work_Rate
- Body_Type
- Height
- Weight
- Crossing
- Finishing
- ...

The dependent variable (what we're trying to predict) is the Release_Clause. The metrics we're going to use are Log Mean Squared Error and R^2.

The assignment will consist of 3 different parts.

1. [M] Machine Learning
2. [M] Docker container
3. [M] Flask web service

Important remark: If, at any moment, you encounter difficulties (and a thorough Google search doesn't give you any answers), feel free to contact us at any time via Slack. Make sure to send it to the both of us (Sam and TIm), this way you will get a reply as quickly as possible.

1. [M] Machine Learning

1. Study the example notebook about the RandomForestClassifier for census data in depth.
2. Play around with some of the functions and data.
3. Create a new notebook, use the RandomForest**Classifier** example as a starting point to create your own RandomForest**Regressor.** Take care to keep the differences between classification and regression in mind.
4. Since we are using the Log Mean Squared Error, make sure to edit the Release_Clause value the following way:

   ```
   df_raw.Release_Clause = np.log(df_raw.Release_Clause)
   ```

   Where df_raw is your pandas dataframe, loaded by

```
df_raw = pd.read_csv(f'{PATH}data.csv', low_memory=False)
```

5. Make sure to handle your categorical variables the same way as the RandomForestClassifier example.
6. Split your data into a Training and Validation set (we'll skip the test set for now)
7. Create your RandomForestRegressor
8. Interpret the results from the metrics. Use these functions to get an overview:

```
def rmse(x,y):
      return math.sqrt(((x-y)**2).mean())

def print_score(m):
      res = [rmse(m.predict(X_train), y_train),
            rmse(m.predict(X_valid), y_valid),
            m.score(X_train, y_train), m.score(X_valid, y_valid)]

      print(res)
```

2. [M] Docker

We're supplying you with a Docker build file, containing the Fast.AI library, Flask support and Jupyter notebooks. The idea of this subtask is to get you familiar with Docker containers, and to make it easier to run a Flask application, which will be the final part of this assignment. The Docker build file and its dependencies are in the supplied .tar.gz archive.

Prerequisites

Docker is the most used way to create reproducibility, isolated containers. It's a handy and lightweight way to encapsulated an application to make it easy to (re)deploy it time after time with the same outcome. The Docker website has some more information on containers:

https://www.docker.com/resources/what-container

A. Install Docker

It's easy to install in a Ubuntu system. Digital Ocean has a very good tutorial on how to install it on Ubuntu version 18.04. Follow "Step 1 - Installing Docker" and "Step 2 - Executing the Docker Command Without Sudo". Use Step 4, 5 and 6 to get a hang of how to work with Docker on a daily basis. Step 7, and so on are out of the scope of this assignment.

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04

B. Create an image

First install Docker and make it executable without sudo. This is the previous section: Step A!

It's possible to create your own Docker Image. We provided a Dockerfile to help you on this front. It will build a Docker Image based on Ubuntu 18.04 with Python 3, fastai (CPU based), Jupyter and Flask.

To build the image go in to the directory containing the Dockerfile supplied by us and execute the following command, do not forget the period at the end. This command will take quite a long time to complete. Feel free to look inside this, and all other, scripts. It's very useful to dissect the contents.

```
$ cd to/the/dir/containing/our/Dockerfile
$ ./build_spring_break_image.sh
```

If you get the following error:

```
error pulling image configuration: Get
https://production.cloudflare.docker.com/registry-v2/docker/registry/v2/blobs/sha256/94/94e814e2efa8845d95
b2112d54497fbad173e45121ce9255b93401392f538499/data?verify=1554917105-QRNCJFKyFYDb08nyuvGpxsQl%2Bx0%3D:
read tcp 10.30.0.34:57506->104.18.121.25:443: read: connection reset by peer
```

This is probably nothing more than a network error. Just rerun the script.

C. Create a container based on the spring_break image

Normally the container will contain the complete software (sub)system. But you are going to use the container as an reliable manner to install/use fastai & Flask and you're still exploring/developing the (sub)system. Therefore, the following commands will link a directory on your computer with one inside the container. That way you can easily edit files locally, remove a container and recreate a new one without losing all your code.

The first command will start a spring_break container and linking the jupyter directory in the local directory (thanks to pwd) with the /home/user/jupyter directory. Thereafter it will launch an Jupyter notebook which listens to all its IPs on port 9999:

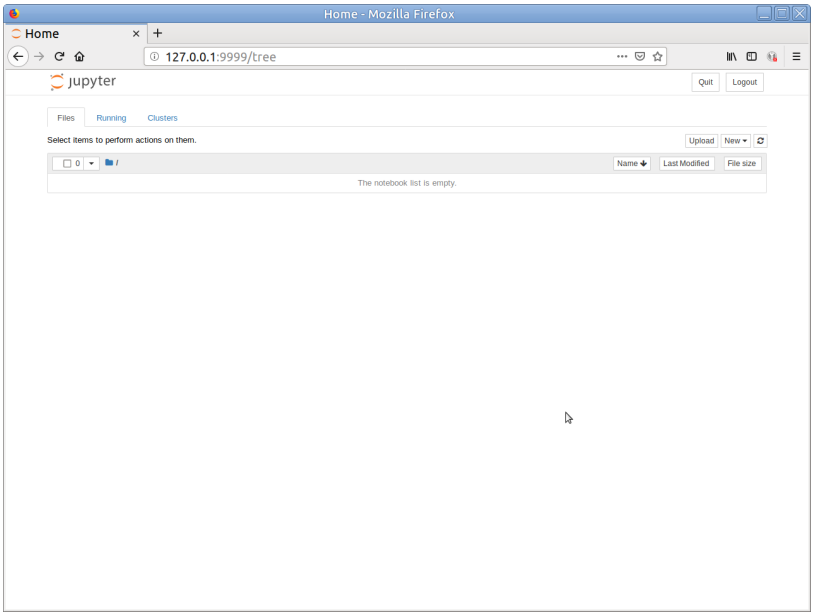
```
$ cd to/the/dir/containing/our/Dockerfile
$ ./start_spring_break_jupyter.sh
```

This command will output something like:

```
[I 15:04:49.591 NotebookApp] Writing notebook server cookie secret to
/home/user/.local/share/jupyter/runtime/notebook_cookie_secret
[I 15:04:50.371 NotebookApp] Serving notebooks from local directory: /home/user/jupyter
[I 15:04:50.371 NotebookApp] The Jupyter Notebook is running at:
[I 15:04:50.371 NotebookApp] http://(cea8577996ce or
127.0.0.1):9999/?token=0c79b78c7c78c539a579787e18d3ccaf7e1c329b252a12cc
[I 15:04:50.371 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip
confirmation).
[W 15:04:50.374 NotebookApp] No web browser found: could not locate runnable browser.
[C 15:04:50.374 NotebookApp]

        To access the notebook, open this file in a browser:
        file:///home/user/.local/share/jupyter/runtime/nbserver-1-open.html
        Or copy and paste one of these URLs:
        http://(cea8577996ce or 127.0.0.1):9999/?token=0c79b78c7c78c539a579787e18d3ccaf7e1c329b252a12cc
```

To access the Jupyter notebook surf to the URL display in the output, using the IP 127.0.0.1.



All the notebooks will be created in the local directory called “jupyter”.

To stop the server just Ctrl-C in the console, this will display the following message:

```
Shutdown this notebook server (y/[n])?
```

Answer “y” to this to stop the server.

To start the Flask application execute the next command:

```
$ cd to/the/dir/containing/our/Dockerfile
$ ./start_spring_break_flask.sh
```

```
The output will look like:
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:9999/ (Press CTRL+C to quit)
```

To access the App surf to http://127.0.0.1:9999/ .

The application code resides in the local directory called app.

There is one downside to working with Docker this way. The stopped containers still exists on the system. You can check this with the “docker ps -a” command. We provided a handy script to remove all those containers:

```
$ cd to/the/dir/containing/our/Dockerfile
$ ./remove_spring_break_conainers.sh
```

3. [M] Flask Web Service

1. Export your model from the first part of the assignment. Use
   ```
   from sklearn.externals import joblib
   joblib.dump(m, 'model.pkl')
   ```
   After this command your model will be available in your filesystem.
2. Build a Flask application
   https://towardsdatascience.com/a-flask-api-for-serving-scikit-learn-models-c8bcdaa41daa
   If you want to load a model, copy it to your “app” folder, and load it in app.py with the following command:
   ```
   clf = joblib.load('./app/model.pkl')
   ```
3. Host your RandomForestRegressor model. Follow the steps in the link above.
4. Create a REST interface, using Docker and Flask.
   REST: https://www.youtube.com/watch?v=YCcAE2SCQ6k
   Make sure to try out the example we gave you, and interpret how it works.
5. Create a new football player with the following attributes:
   ID | Name | Age | Nationality | Overall | Potential | Club | Value | Wage | Special | Preferred_Foot | International_Reputation | Weak_Foot | Skill_Moves | Work_Rate | Body_Type | Real_Face | Position | Jersey_Number | Loaned_From | Height | Weight | LS | ST | RS | LW | LF | CF | RF | RW | LAM | CAM | RAM | LM | LCM | CM | RCM | RM | LWB | LDM | CDM | RDM | RWB | LB | LCB | CB | RCB | RB | Crossing | Finishing | HeadingAccuracy | ShortPassing | Volleys | Dribbling | Curve | FKAccuracy | LongPassing | BallControl | Acceleration | SprintSpeed | Agility | Reactions | Balance | ShotPower | Jumping | Stamina | Strength | LongShots | Aggression | Interceptions | Positioning | Vision | Penalties | Composure | Marking | StandingTackle | SlidingTackle | GKDiving | GKHandling | GKKicking | GKPositioning | GKReflexes | Release_Clause

   Study the dataset carefully so you know what kind of data the model expects. Try to automate/randomize as many values as you can, because filling in all the numeric values will take some time.
6. Send the created player to the flask application (i.e. using a JSON POST request) and predict the dependent variable (Release_Clause)
   The only information your flask application has to return is the value of the prediction.

   How to Handle Request JSON Data in Flask: https://www.youtube.com/watch?v=kvux1SiRIJQ