Statistical Learning

ATPESC ML 2022

Sam Foreman

2022-08-12

First Steps

1. Login and submit an interactive job (on Polaris):

```
$ ssh <username>@polaris
$ # ATPESC queue: -q R313446 (in general: -q prod)
$ qsub -A ATPESC2022 -q R313446 -l select=1 -l walltime=01:00:00 -I
```

   this will launch a job with 1 rank ($\times 4$ GPUs) for 1 hour
   (more on queues / scheduling)

2. From the interactive job, clone the github repo:
   ATPESC_MachineLearning

```
$ hostname
x3002c0s31b0n0
$ git clone https://github.com/argonne-lcf/ATPESC_MachineLearning
$ cd ATPESC_MachineLearning/00_statisticalLearning/
```

Setup / Install

```
$ tree ATPESC_MachineLearning/00_statisticalLearning/
📁 src/
└── 📁 atpesc/
    ├── 🐍 common.py
    ├── 📁 notebooks/
    │   └── 📕 statistical_learning.ipynb
    └── 📁 utils/
        └── 🐍 plots.py
```

- **Goal:** Use functions located in `common.py` and `utils.py` from within our Jupyter notebook.

- To do this we:
  1. Create a python `venv` which we will use to launch our `jupyter notebook`
  2. From within this `venv`, perform a local (editable) install `python3 -m pip install -e .`

Jupyter Notebooks

1. Load base conda environment (as a starting point):

```
$ module load conda/2022-07-19  # from our interactive job
$ conda activate base
```

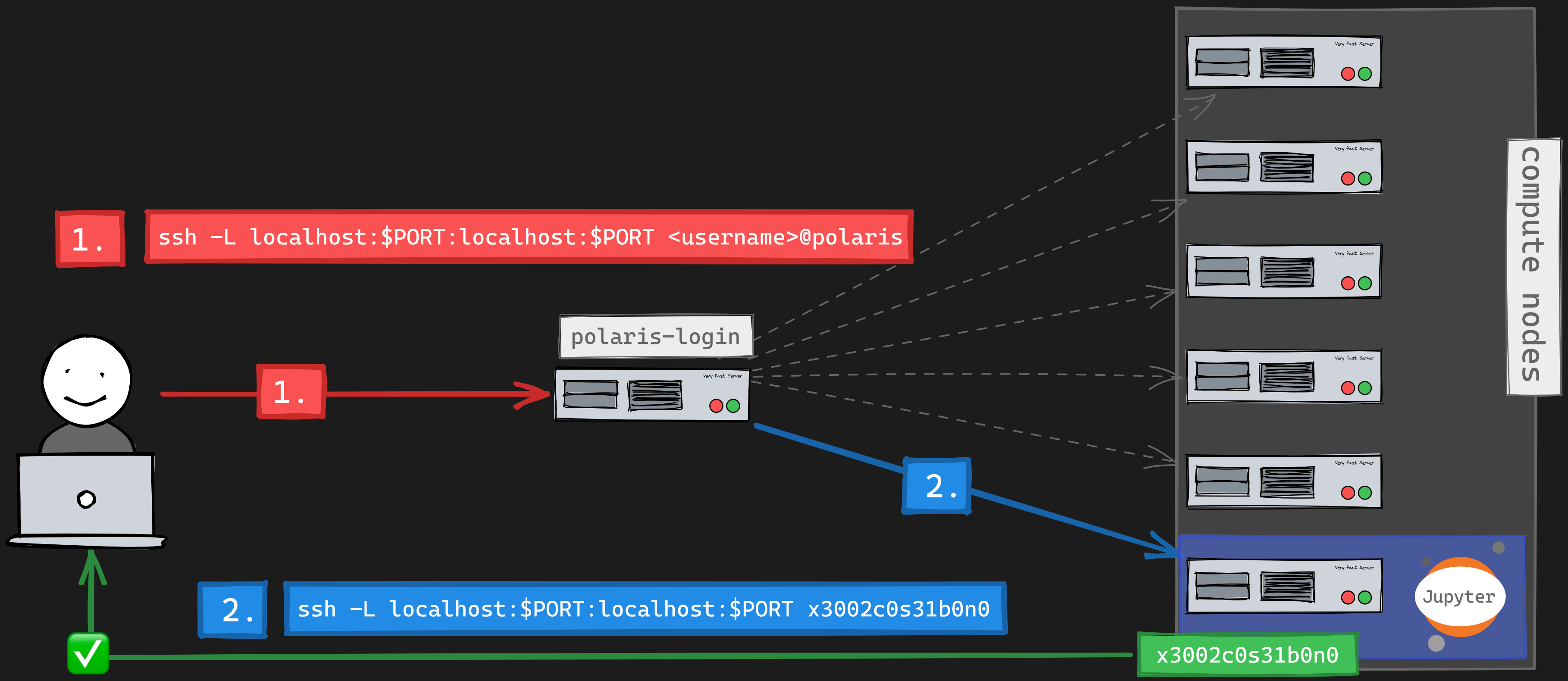
2. Create (isolated) venv and perform local install:

```
$ cd ATPESC_MachineLearning/00_statisticalLearning/
$ python3 -m venv venv --system-site-packages
$ source venv/bin/activate
$ # will install `atpesc` into the `venv`
$ python3 -m pip install -e .
```

3. Install Jupyter kernel and launch notebook

```
$ python3 -m pip install ipykernel
$ python3 -m ipykernel install --user --name="2022-07-19-ATPESC" \
  --display-name="2022-07-19 ATPESC"
$ jupyter notebook --port=8899 --no-browser > /tmp/jlab8899.log 2>&1 &
$ hostname
x3002c0s31b0n0
```

Port Forwarding



Port Forwarding

Connect `localhost` to compute node running Jupyter.

6. Starting from your **local machine**

```
# 1: localhost <--> polaris-login-01
ssh -L localhost:8899:localhost:8899 <username>@polaris.alcf.anl.gov
# 2: polaris-login-01 <--> compute node
ssh -L localhost:8899:localhost:8899 <username>@x3002c0s31b0n0
```

7. From a web browser on your local machine, navigate to: `https://localhost:8899/`

> ⚠ **Warning!**
>
> Only **one** port (8899 in this example) can be used at a time.
>
> Because of this, if someone is already using the port you try and specify, your connection **WILL NOT WORK**
>
> To remedy this, (randomly?) choose a different port (e.g. 8891, 8873, etc.)

Linear Regression

Line Fitting

Linear regression via *stochastic gradient descent* (SGD).

1. Load data into pandas DataFrame:

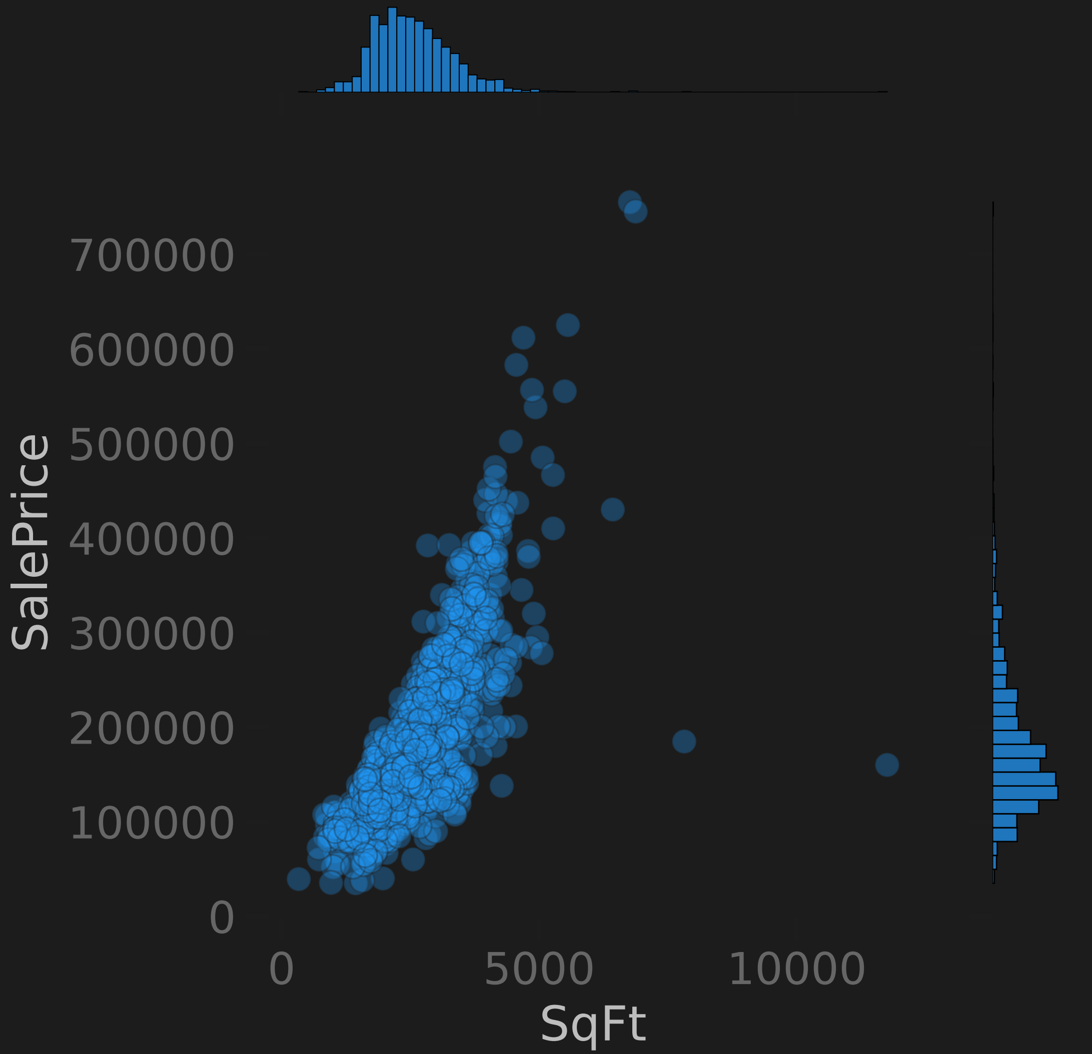
```
import pandas as pd
from pathlib import Path
from atpesc.common import DATA_DIR
data_file = Path(DATA_DIR).joinpath('realestate_train.csv')
df = pd.read_csv(data_file)
```

2. Extract total home square footage

```
area = sum([
   df['1stFlrSF'],
   df['2ndFlrSF'],
   df['TotalBsmtSF']
])
area.name = 'SqFt'
price = df['SalePrice']
```

Sale Price vs. Square Footage

```
sns.jointplot(x=area, y=price, alpha=0.33)
```



Linear Regression

- We can fit the data with a line in order to estimate future sale prices based on home size.
- Assume a simple linear relationship ($y = m \cdot x$)

$$\text{price} = m \cdot \text{area}$$

> **How does our prediction fit the data?**
>
> In order to evaluate how well our predictions match the data, we can use the Mean Squared Error (MSE):
>
> $$\text{MSE} \equiv \delta(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}\right)^2$$

Linear Regression

```python
def predict_price(slope, area):
    return slope * area

def evaluate(slope, area, true_price):
    prediction = predict_price(slope, area)
    return np.mean((true_price - prediction) ** 2)
```

Linear Regression

- Recall our prediction is given by $y = m \cdot x$, where:
  - m is the **slope** (randomly initialized)
  - $\hat{y}$ is the true price
  - y is the predicted price
  - x is the input area
  - α is the learning rate

- We update our slope m using the update policy:

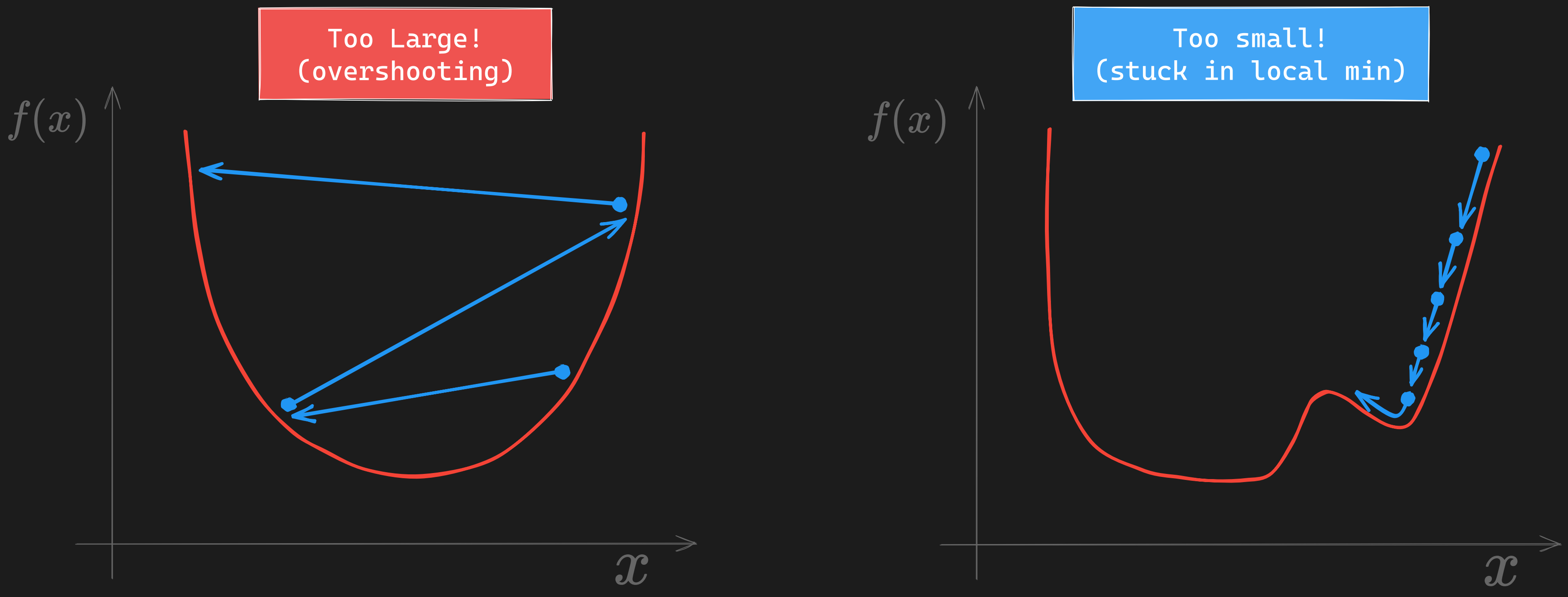
$$
\begin{aligned}
m &\leftarrow m - \alpha \nabla \delta \left(y, \hat{y}\right) \\
&= m - \frac{\alpha}{n} \nabla \left\{ \sum_{i=1}^{n} \left(y_i - \hat{y}\right)^2 \right\} \\
&= m - \frac{\alpha}{n} \sum_{i=1}^{n} 2 \left(y_i - \hat{y}\right) \cdot \frac{\partial y_i}{\partial m} \\
&= m - \frac{\alpha}{n} \sum_{i=1}^{n} 2 \left(y_i - \hat{y}\right) \cdot x_i
\end{aligned}
$$

Linear Regression

```
def learn(
    area,
    slope,
    true_price,
    lr = 0.000001,
):
    prediction = predict_price(slope, area)
    dfdx = 2. * np.mean((prediction - true_price) * area)
    new_slope = slope - learning_rate * dfdx
    return new_slope
```

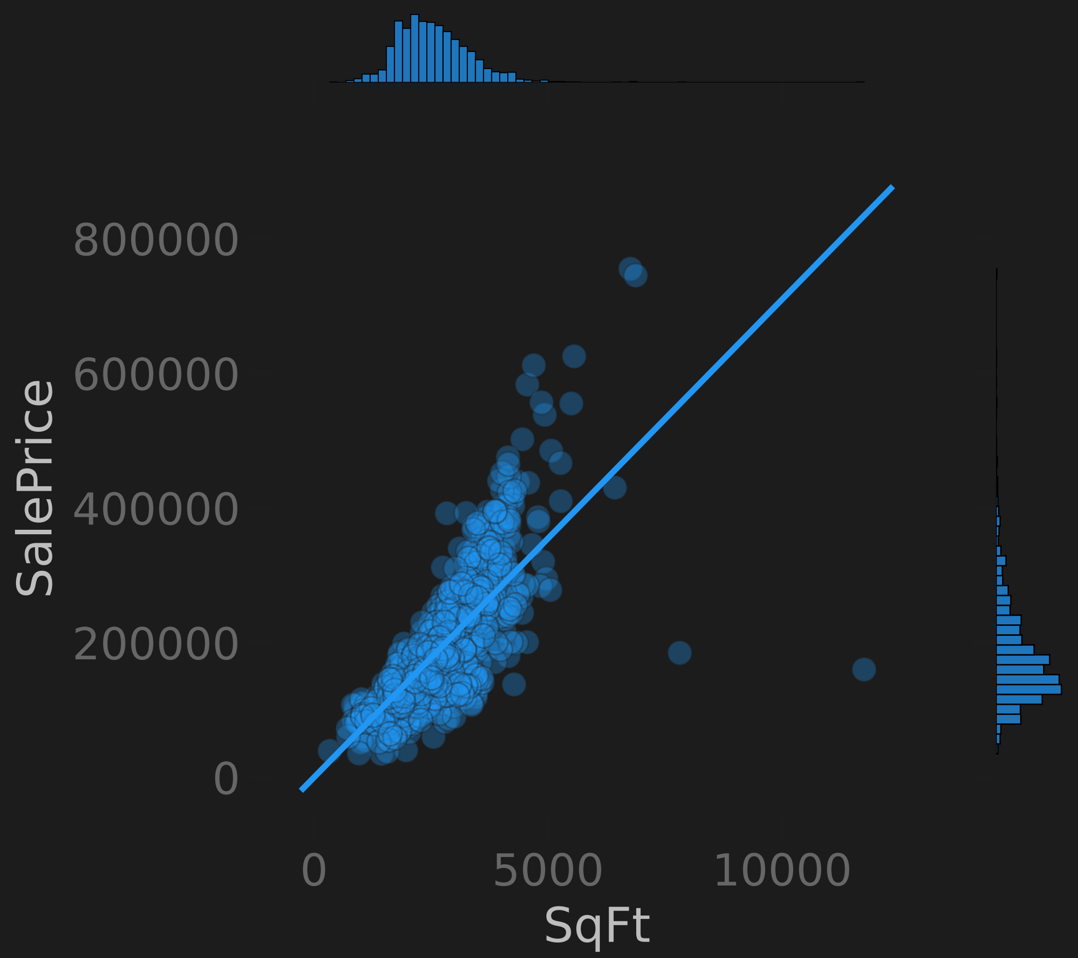
Stochastic Gradient Descent

- Each application of the `learn` function updates the slope and the `learning_rate` dampens that update.
  - This iterative method helps to find the value of x that *minimizes* the gradient $\frac{df}{dx}$.
- The `learning_rate` controls the size of the update step when updating x.



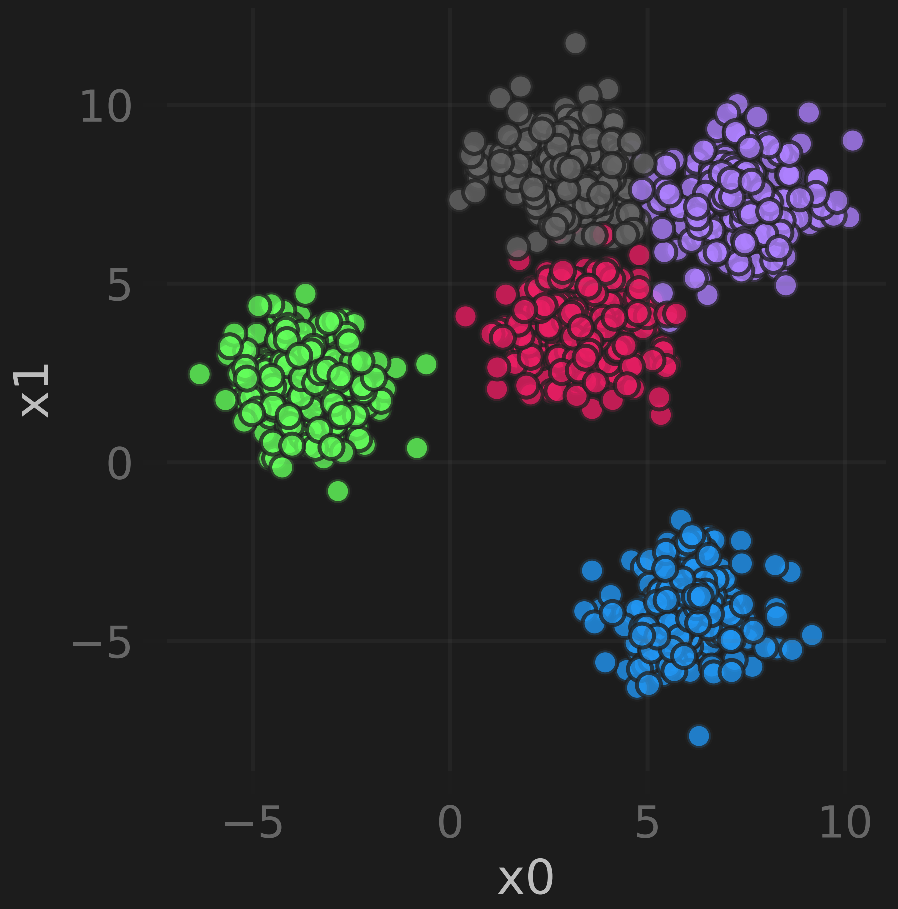
Linear Regression

Iteration: 9, Current Slope: 70.95609



Data Clustering

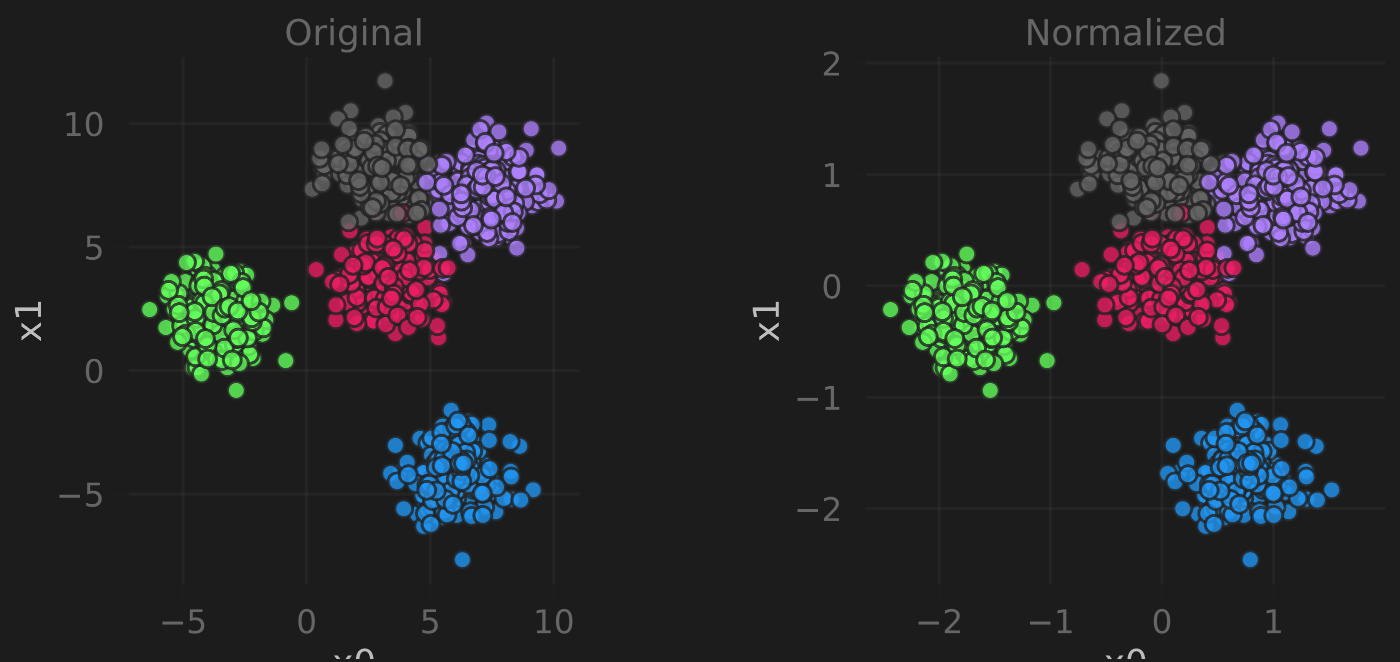
Data Clustering



Data Clustering

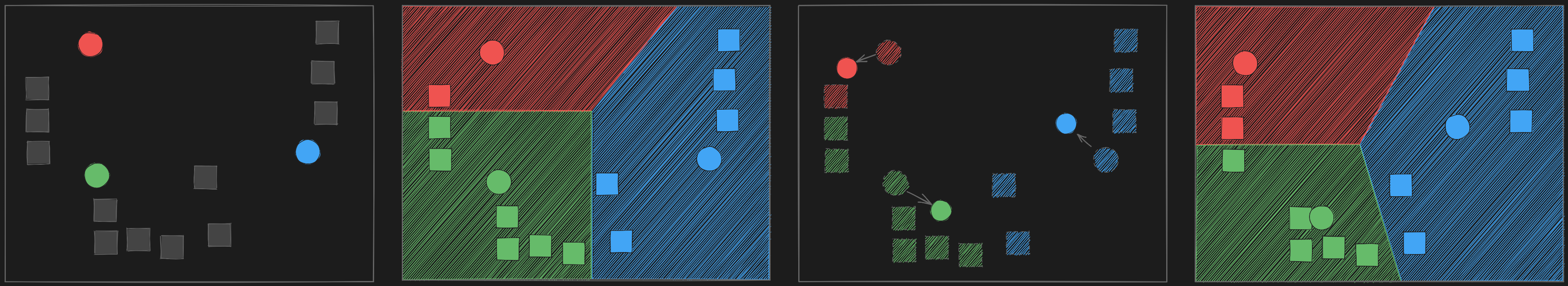
Normalization

Always a good idea to normalize your data



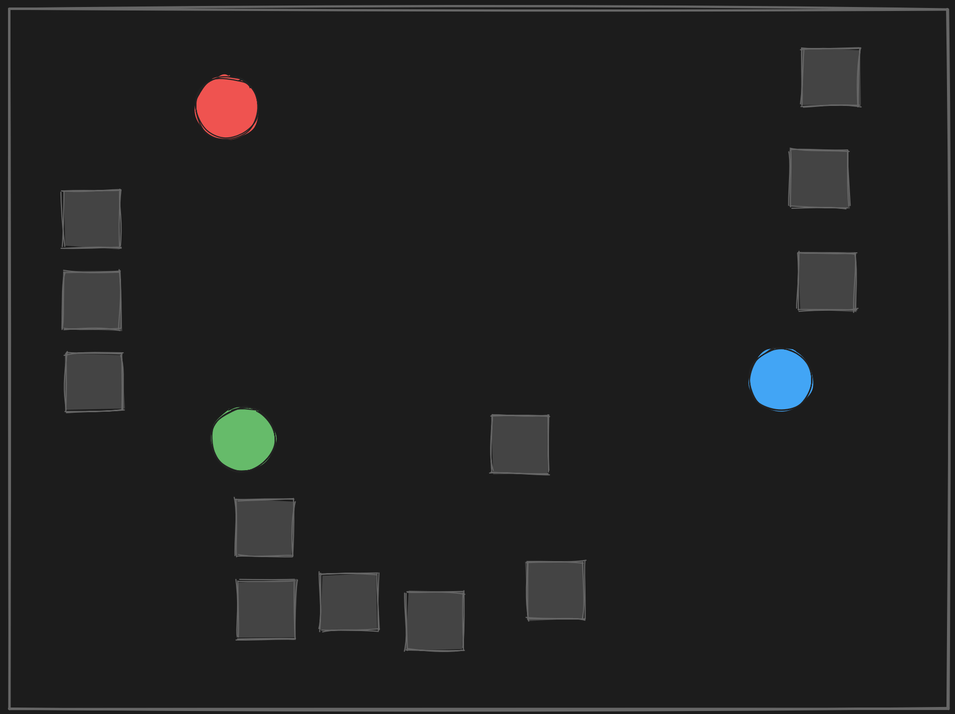
K-Means Clustering

- **Goal:** Partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean.



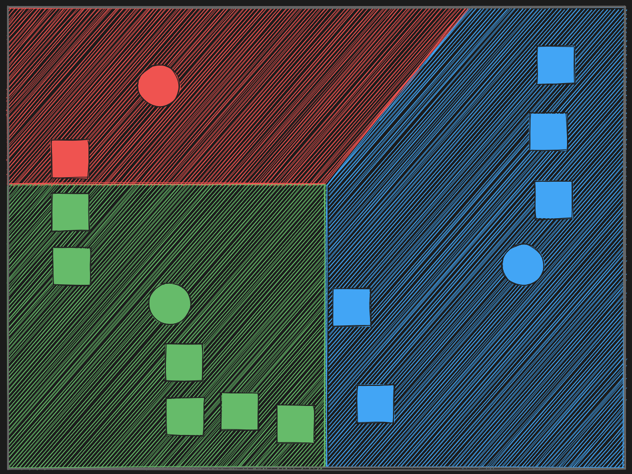
K-Means: Step 1

1. k initial means (in this case, $k = 3$) are randomly generated within the data domain (shown in color)



K-Means: Step 2

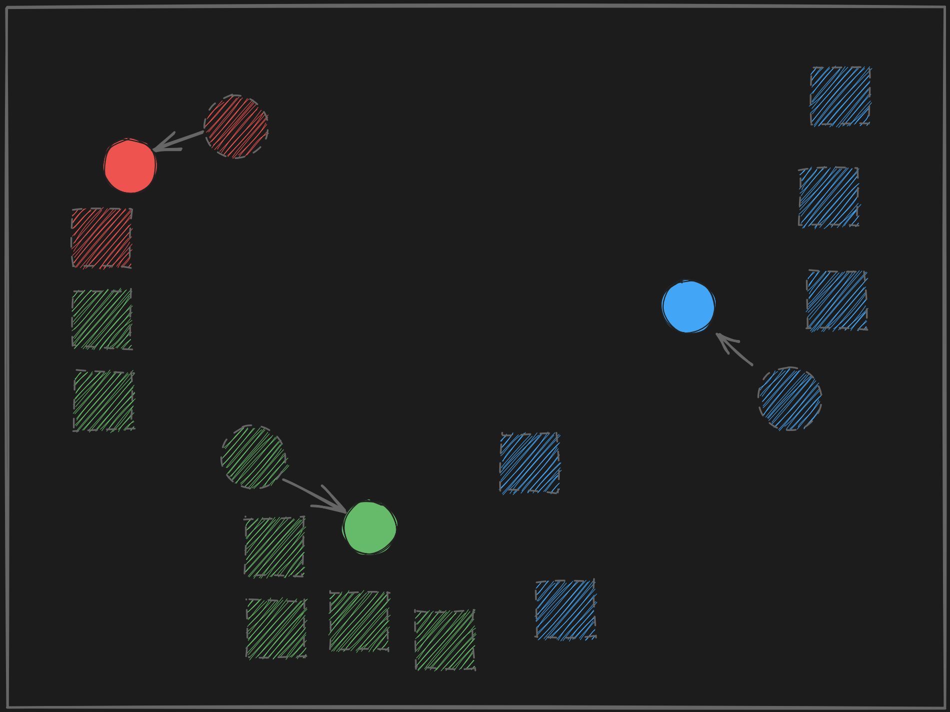
2. Calculate distance to each centroid:[1]
    1. k clusters are created by associating every observation with the nearest mean.
    2. Find nearest cluster for each point.



1. The partitions here represent the Voroni diagram generated by the means

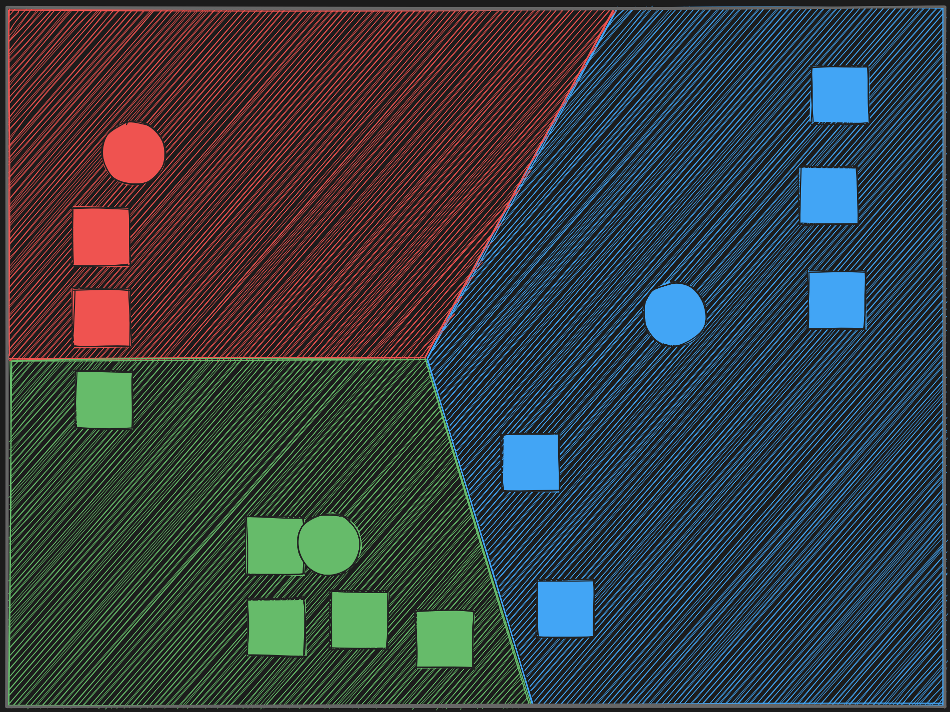
K-Means: Step 3

- Calculate the new centroids
  - The **centroid** of each of the k clusters becomes the new mean



K-Means: Step 4

Repeat until convergence



Hands-On / Live Demo

- 📊 slides
- argonne-lcf/ATPESC_MachineLearning/
  - 00_statisticalLearning
  - 📕 statistical_learning.ipynb