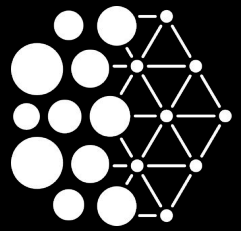
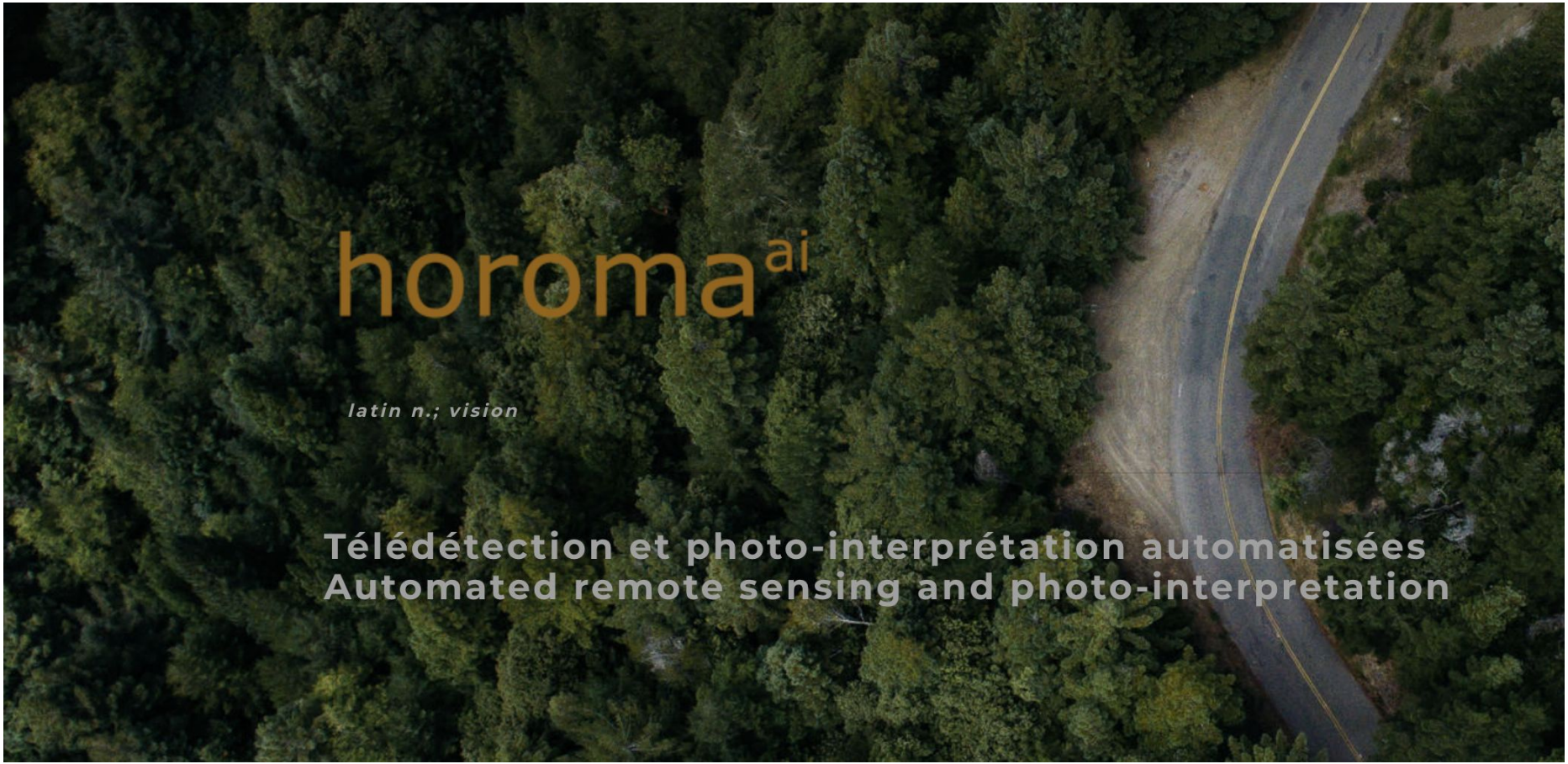
Institut
québécois
d'intelligence
artificielle



Mila

Horoma project
Block 1

Francis Grégoire
Mathieu Germain


horomaai
latin n.; vision
Télédétection et photo-interprétation automatisées
Automated remote sensing and photo-interpretation

Horoma project

- **Task:** Analysis of images of forest canopy to determine:
  - Tree specie.
  - Tree density.
  - Tree height.

- **Main challenge:** Data labeling is made by human interpreters and is therefore costly and time consuming.

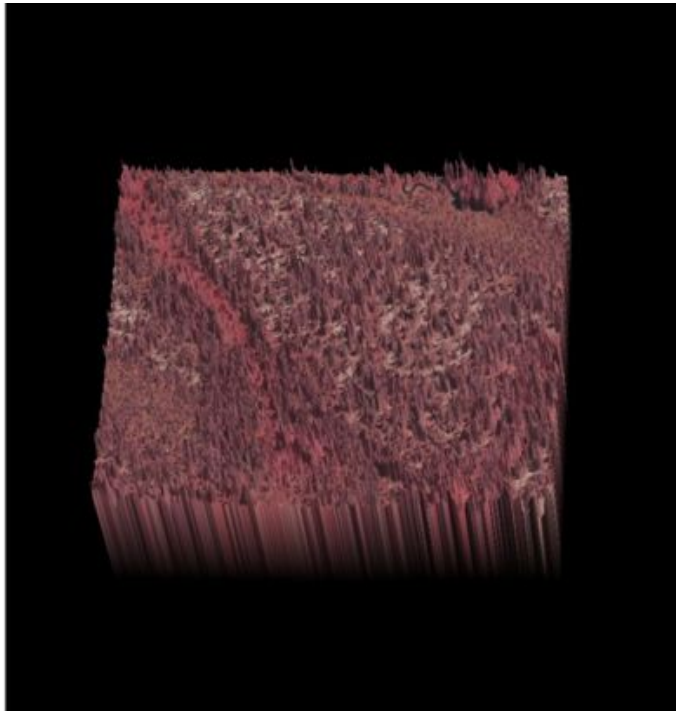
- **Goal:** Develop an unsupervised or semi-supervised machine learning system capable of predicting forest canopy properties with no or small amounts of labeled data.

Project summary

- **Overall task**:
  - Develop a model that predicts the labels of 32 x 32 pixel patches.
- **Block 1**:
  - Unsupervised training (i.e. labels are provided for the validation set; no labels are provided for the training set).
  - Structure data in a way that is useful to develop models for blocks 2 & 3.
  - Apply K-means clustering as a baseline model.
- **Block 2**:
  - Unsupervised training.
  - Develop more advanced clustering algorithms.
- **Block 3:**
  - Semi-supervised training (i.e. labels are still given for the validation set, and labels are also provided for a small number of training examples).
  - Develop semi-supervised classification algorithms.

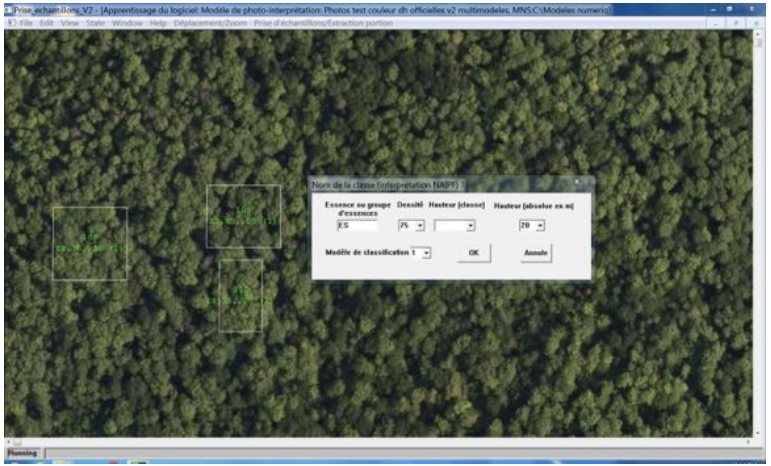
Data (image)

- 3 Digital Surface Models (DSM) of forest areas in the Outaouais, a Western Quebec region.
- An image (or DSM) covers approximately 5km*3km (15km^2) of land, such that a single pixel represents an area of about 30cm*30cm.
- All images were taken during the same period of the year when trees are leafy (thus facilitating image labeling).



Data (image labeling)

- Each image was labeled by a different human interpreter.
- Interpreters only labeled subsections of an image, with each subsection containing trees of the same overall specie, density and height (w.r.t the forest floor).
- The tree specie, density and height values were estimated by the interpreters.



Data (image labeling)

- Tree density and height estimated by interpreters are approximate measurements.
- Interpreters base their measurements on the diameter and shape of treetops.
- Tree height measurement (which is rounded at 5m) is difficult to determine when there are no gaps in the image allowing the expert to see the forest floor.

Data (pixel)

- Each pixel of an image has 4 values associated with it:
    - **RGB colors** (3 values). Those values are in [0, 255].
    - **Height** (1 value). The height values were obtained using photogrammetry and were georeferenced; they are measured w.r.t. the sea level.

Data (pixel patches)

- 32 x 32 pixel patches were extracted from labeled image subsections.

- **Inputs**: each pixel in a 32 x 32 pixel patch has 4 dimensions:
  - RGB colors in [0, 255] (3 values).
  - Relative height originally defined as the height w.r.t. the sea level, where we subtracted from it the smallest pixel height within the 1024 pixels of the 32 x 32 pixel patch:

$$relative\ height_j^i = height_j^i - \min(height_1^i, height_2^i, \ldots, height_{1024}^i)$$

    where *i* = index of pixel patch and *j* = index of pixel in pixel patch *i*.

- **Outputs**: each 32 x 32 pixel patch has 3 labels:
  - Tree specie.
  - Tree density.
  - Tree height w.r.t. the forest floor.

Data (pixel patches)

- A 32 x 32 pixel patch always contains trees of the same specie, density and height.
- **Image 1**: 430,768 patches.
- **Image 2**: 575,996 patches.
- **Image 3**: 1,011,008 patches.
- **Total**: 2,017,772 patches.

Data (train, valid and blind test sets)

- **Blind test set**:
  - 43,077 patches randomly selected from the 1st image.
  - 57,600 patches randomly selected from the 2nd image.
  - 101,101 patches randomly selected from the 3rd image.

- **Validation set**:
  - 43,077 patches randomly selected from the 1st image (not in test set).
  - 57,600 patches randomly selected from the 2nd image (not in test set).
  - 101,101 patches randomly selected from the 3rd image (not in test set).

- **Train set**:
  - 344,614 patches (leftovers from the 1st image).
  - 460,796 patches (leftovers from the 2nd image).
  - 808,806 patches (leftovers from the 3rd image).

Data (input format)

- Inputs are provided as binary numpy.memmap files in **float32**.
- Memory-mapped files are used for accessing small segments of large files on disk, without reading the entire file into memory.
- Each pixel patch has a shape of 32 x 32 x 4 (where 4 = [R, G, B, relative height]).
- **train_x.dat**: 1,614,216 x 32 x 32 x 4.
- **valid_x.dat**: 201,778 x 32 x 32 x 4.

Data (output format)

- Outputs are provided as text files (can be easily read from a terminal).
- **valid_y.txt**: contains 201,778 labels (unidimensional discrete values).
- **definition_labels.txt**: contains a list of triples.
  - **1st value**: tree specie (2 characters).
  - **2nd value**: tree density (percentage*100 rounded at lowest unit of 5).
  - **3rd value**: tree height w.r.t. the forest floor (rounded at nearest 5m).
- The discrete value of each label in **valid_y.txt** is mapped to the triple corresponding to the row index (starting at 1) in **definition_labels.txt** that is equal to the value of the label.
- e.g. if $label_j = 3$ then $triple_j = (\mathrm{TO}, 55, 15)$ (i.e. 3rd row in **definition_labels.txt).**

Data (tips)

- This is a hard unsupervised learning task.
- Find clever ways to exploit the information that is given to you to determine the number of clusters.
- You should document your different approaches and explain your ideas in your report. Don’t be afraid to be original!

Block 1 instructions & expected timeline

	Jan 15 & 17	Jan 23 & 25	Jan 30 & Feb 1	Feb 6 & Feb 8
Tasks	• Start working on the data loader	• Complete data loader • Start working on data visualization (pixel patches, PCA, t-SNE)	• Complete work on data visualization (pixel patches, PCA, t-SNE) • Train K-means clustering to predict the tree specie, density and height of a patch	• Write a short report summarizing the work, and results • Peer review the code of other teams
Objectives / Deliverables	• Proper understanding of the data and its structure	• Data loader ready	• Data visualization ready • K-means clustering for pixel patch classification	• Produce documented code and report summarizing the experimental work • Provide model for blind test set evaluation • Complete peer code review

Deadlines

- Each team needs to provide the deliverable (report + code + best model) corresponding to a block at the latest on **Friday noon (12:00pm)** of the last week of the block.

- Any block deliverable that is provided past **Friday noon (12:00pm)** of the last week of a block will automatically get 0% for the peer evaluation.

Deadlines

- Any block deliverable that is provided past **Tuesday 11:59pm** following the last week of a block will automatically get 0% for the UdeM evaluation.

- Peer evaluation must be completed by **Monday 11:59pm** following the last week of a block.

Evaluation grid for block 1

- **25% of the final score**.
- 10% Code review [5% of averaged peer evaluation + 5% UdeM].
- 12% Report evaluation [UdeM].
- 3% Model performance evaluation on blind test set [UdeM].

Code review - Peer evaluation

- **10% of the final score** [5% of average peer evaluation + 5% UdeM].
- Random assignation of code reviews.
- The code provided by a team will be evaluated by at least 2 other teams.

Code quality (peer evaluation + UdeM evaluation)	8
Coherent and modular code/file organization (e.g. data processing, model definition, model training, model inference are in different files/modules; no code duplication)	
Code respects the PEP8 standard	
Comments are relevant (see article)	
Proper management of input arguments in the training script (see argparse, python fire, configparser)	
Proper utilization of GitHub (e.g. branching, relevant commits and messages, usage of pull request)	
Meaningful variable and function names	
Executable scripts with a “main” function (see article)	
Reproducible experiments (e.g. seed)	

Introduction	2
Introduction to the project	
Brief introduction to the methods that will be used in the report	
Methodology	6
Description of the algorithms and the experiments (including hyperparameter fine tuning (if appropriate), etc.)	
Data description and data selection (train/valid/test, number of samples, shape/structure of data points)	
Results and discussion	6
Presentation of results (tables, figures, etc.)	
Discussion of results	
Conclusion	2
Recommendation for next steps	
Summary of project state (what was done, what needs to be done)	
Quality of the report	2
Report format (title with team member names, clear sections, flow between sections, figures and tables titled, axes titled, etc.)	
Report is short and to the point (5-7 pages including references, font size 11)	

Report evaluation

- **12% of the final score**.
- 5-7 pages (including figures, tables, and references).
- Single column.
- Font size 11.
- Use the NeurIPS LaTeX format.

Blind test set evaluation

- **3% of the final score**.
- If the best model provided by a team crashes or provides results that are statistically worse than those of the baseline model provided by the TAs, the team gets 0%.
- Otherwise, if the best model provided by a team is statistically equivalent to the baseline model, the team gets 1%.
- Otherwise, if the best model provided by a team is statistically better than the baseline model:
    - The team gets 3% if the model is the best performing one or is statistically equivalent to the best performing model provided by another team.
    - Otherwise, the team gets 2%.

Code execution - Blind test set evaluation

- The **test set** will be structured identically to the **valid set**.
- You will not have access to the **test set** and we will be executing your code on the **test set** ourselves.
- You should create a **eval.py** file with clear instructions to run the code. We will provide you with explicit instructions and examples to run your code. We reserve the right to give 0 if we cannot execute your code.
- Tips: **do not shuffle the test set.** Predictions should be made sequentially.

Official evaluation metric

- Use **valid_y.txt** to calculate your patch classification accuracy:

$$accuracy_{(s,\,d,\,h)} = \#\text{ correctly classified patches} / \text{total }\#\text{ patches}$$

- A patch is correctly classified if the predicted tree **specie**, **density** and **height** are all correct.

- The code of the scoring function will be provided.

Informative evaluation metrics

- Tree specie classification accuracy:

  $accuracy_s$ = # patches with correct tree specie prediction / total # patches

- Tree density classification accuracy:

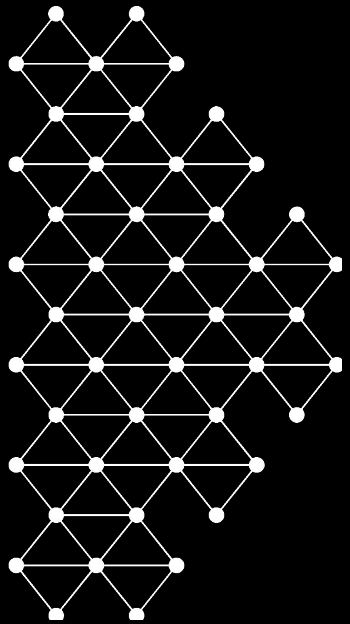
  $accuracy_d$ = # patches with correct tree density prediction / total # patches

- Tree height classification accuracy:

  $accuracy_h$ = # patches with correct tree height prediction / total # patches

- Explore useful metrics for clustering performance evaluation such as Normalized Mutual Information (NMI) and adjusted Rand index (ARI) (see scikit-learn documentation).

Institut
québécois d'intelligence
artificielle



K-means clustering

Unsupervised learning

- We assume that a data generating process contains internal variables that are not observed, but that still influence the data.

- These unobserved variables are called hidden variables or latent variables.

- We postulate that there is a latent variable $\mathbf{z}_i$ associated with each training example $\mathbf{x}_i$.

K-means clustering

- Most widely used clustering algorithm.
- We consider $\mathbf{z}_i$ as a discrete latent variable.
- The inference of $\mathbf{z}_i$ is known as clustering because it leads to assigning each training example $\mathbf{x}_i$ to a group among K, where $\mathbf{z}_i$ represents the assigned group (cluster).
- We use one-hot encoding to model $\mathbf{z}_i$.
- e.g. if $K = 5$ and $\mathbf{z}_i$ is associated to group 4, then $\mathbf{z}_i = [0, 0, 0, 1, 0]$ ($z_{i4} = 1$).

K-means clustering

- We suppose that there are *K* prototypes $\mathbf{u}_1, \dots, \mathbf{u}_K$ which represent the means of the clusters (centroids).
- The dimension of a prototype $\mathbf{u}_k$ is equal to the dimension of $\mathbf{x}_i$.
- Each $\mathbf{x}_i$ belongs to the cluster with the nearest prototype (i.e. shortest Euclidean distance).
- The loss function of *K*-means clustering minimizes the average distance between the training examples and their associated $\mathbf{u}_k$:

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \|\mathbf{x}_i - \mathbf{u}_k\|^2$$

K-means clustering

- The iterative learning process corresponds to finding the prototypes $\mathbf{u}_k$ as well as the associations $\mathbf{z}_i$ that minimize J.

- It is common practice to initialize the $\mathbf{u}_k$ by randomly choosing K training examples and using these as initial values.

K-means clustering

- Finding the optimal $\mathbf{z}_i$:
  - By knowing the $\mathbf{u}_k$ we can optimize all $\mathbf{z}_i$ independently.
  - *J* is minimized if z_{ik}= 1 with the closest $\mathbf{u}_k$ ($\mathbf{z}_i$ = [z_{i1}, z_{i2}, … , z_{iK}]) and 0 otherwise.
  - The optimal solution is:

$$z_{ik} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_i - \mathbf{u}_j\|^2 \\ 0 & \text{else} \end{cases}$$

K-means clustering

- Finding the optimal $\mathbf{u}_k$:
  - By knowing the $\mathbf{z}_i$ we can minimize J by setting the derivative w.r.t. $\mathbf{u}_k$ equal to zero:

$$2\sum_{i=1}^{N} z_{ik}(\mathbf{x}_i - \mathbf{u}_k) = 0$$

  - The optimal solution is:

$$\mathbf{u}_k = \frac{\sum_i z_{ik}\mathbf{x}_i}{\sum_i z_{ik}}$$

  - The optimal $\mathbf{u}_k$ is the mean of all examples associated to the cluster k.

K-means clustering

- The learning process consists of iterating the inference of $\mathbf{z}_i$ and the update of $\mathbf{u}_k$ until the convergence of J:

1. Initialize $\mathbf{u}_k$

2. Repeat until convergence {

Find the optimal $\mathbf{z}_i$

Find the optimal $\mathbf{u}_k$

}

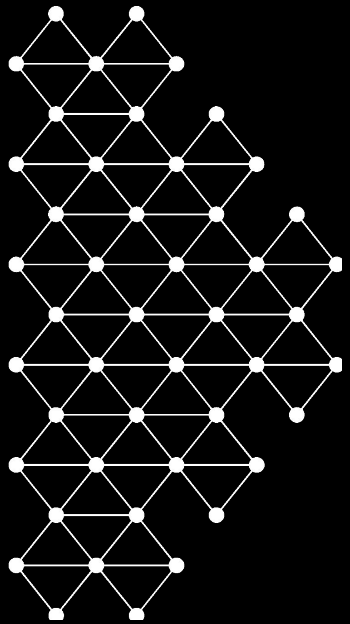
- This procedure is guaranteed to converge because J is positive and at each iteration, either J decreases or does not change.

K-means clustering

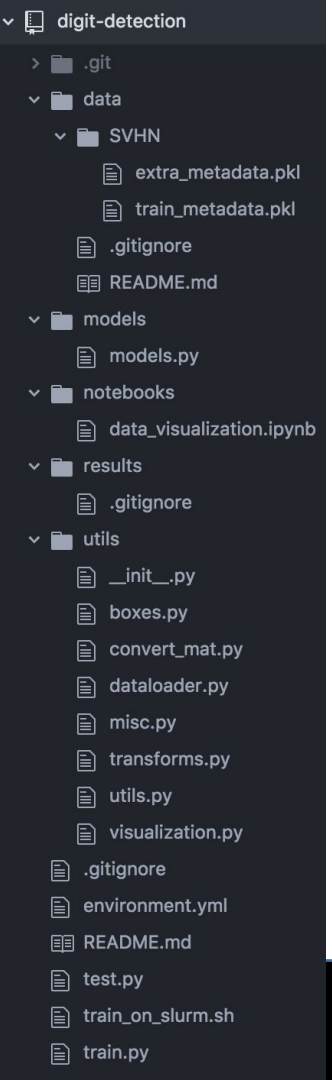
PLAY WITH ME!

Institut
québécois d’intelligence
artificielle

Mila



Good to know



How the code should be organized

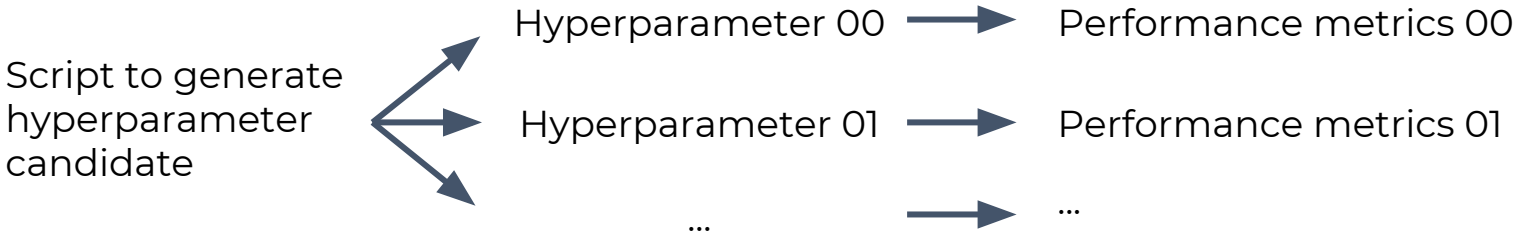
- Have separate folders: config, data, models, notebooks, results, trainer, utils ...
- In each folder, separate the code into several files to simplify reading.
- Use meaningful names for your directories and files.
- Avoid code duplication => use object-oriented programming (e.g. parent and child classes).
- Use .gitignore to avoid overloading the GitHub directory.

Running a deep learning experiment

- Training diagram for a model instance:

Running a deep learning experiment

- Exploration of hyperparameters to obtain the best model:



- Each experiment must produce enough logs to:
  - Diagnose errors and suspicious behaviour.
  - Allow the selection of a set of hyperparameters that is considered the best to generalize to new data.

Implementation - To keep in mind

General advice

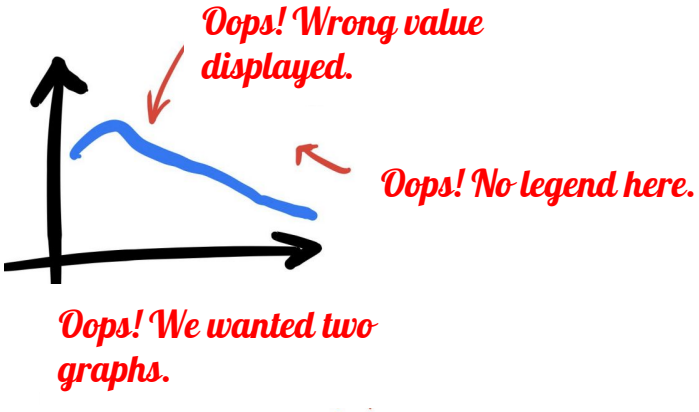
- See what other people are doing and use it as inspiration.
- Develop your own style from this.

Implementation of a deep learning model

- Reuse as much as possible models already available online. Search for "model zoo".
- Sometimes we even want to use (or start from) a pre-trained model. Search for "pretrained model" or "pretrained model weights".

Implementation - To keep in mind

- When you run several large-scale experiments, you must store the data necessary to generate graphs / figures in case you need to modify / update them.



- An experiment that can be interrupted in the middle and then restarted without being affected by the interruption is much easier to manage.