AI for Astrophysics:
Introduction to Machine Learning

David Cornu

LUX, Observatoire de Paris, PSL

Doctoral course – ED AAIF
2025/2026

Provisional planning

Day 1 : Introduction to AI and Machine Learning, simple algorithms hands-on

Day 2 and 3 : Artificial Neural Networks presentation, simple guided implementation, Multi-Layer Perceptron principles and usage

Day 3 and 4 : Deep Learning and advanced neural networks architectures. Frameworks hands-on and application to image based dataset.

Day 5 : Advanced deep learning tasks and network structure.
Application to object detection, image generation, etc.

Lessons materials

Slides, exercises, codes, corrections and datasets are **available on GitHub** and will be updated regularly:

http://github.com/Deyht/AI_astro_ED_AAIF

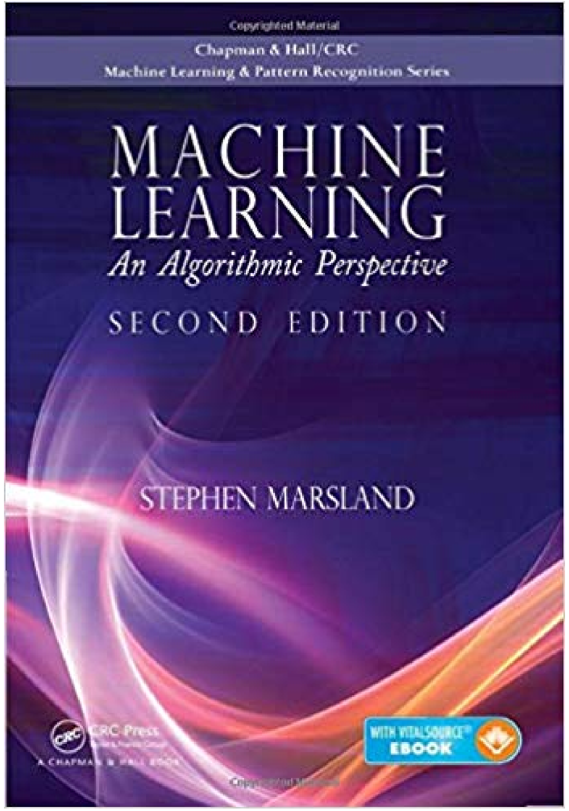
git clone https://github.com/Deyht/AI_astro_ED_AAIF
git pull

Or download the repository in zip file

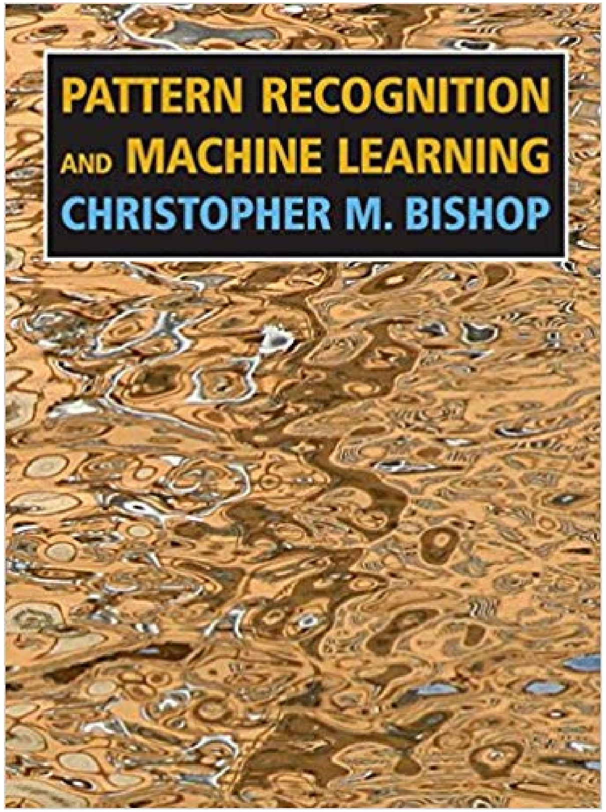
Avoid losing your work on forced pull updates by copying all files from the cloned repository into a working directory!

Do not copy and past content from git-hub pages (lead to format errors and python scripts not running).

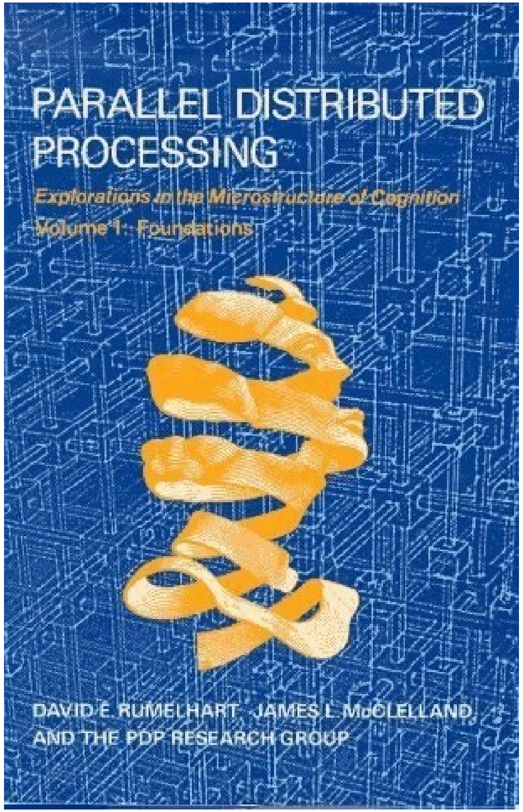
Some learning materials and references



Stephen Marsland. **Machine Learning an Algorithmic Perspective**. Chapman and Hall/Crc, 2015



Christopher M. Bishop. **Pattern Recognition and Machine Learning**. Springer, 2011



David E. Rumelhart et al. **Parallel Distributed Processing**. MIT press, 1989

Some learning materials and references

- My PhD Thesis: https://arxiv.org/abs/2010.01431
  or its updated version: https://share.obspm.fr/s/X3xN6NxJbnZDBWX
  Chapters 2, 4 and 11 can be considered as a handout for this course

- **The Fidle** (Formation Introduction au Deep Learning) **course from the CNRS:**
  https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/

- The Neural Network CS231n course from Stanford:
  https://cs231n.github.io/

- Some more references on a dedicated page on the MINERVA website
  https://minerva-astro.obspm.fr/learning/

Take everything you read about ML and AI with a grain of salt! (This course included)

Lots of materials still contain practical or theoretical approximations and errors.
Crossing references is most of the time necessary.

1) What is Machine Learning ?

Any method that extracts statistical information about a dataset through a learning process.
Another appropriate name is « **statistical learning** » .
But how to define learning ? (still a relatively open question depending on the research field)

And what is **Artificial Intelligence** then ? Tricky and subjective question …
→ Most of the time ML is considered as a sub-field of a larger AI domain.

2) Why is it interesting ?

Learns directly from data:
- Can learn unknown relations that are too difficult to formulate analytically
- Can work efficiently in high dimensional parameter spaces and for large volumes of data
- Can approximate very complex relations (methods that are Universal Function Approximators)

3) For which applications ?

Almost anything you can think of !
BUT it is limited by the data availability and computing power.

Are ML methods better for any application ? Not at all !
Warning: AI is often used as a selling argument, while it is not automatically better than classical data analysis
**in a 2019 report from the London venture capital firm MMC, 40% of « AI startups » don't really use AI or ML*

Attempt of learning definition

The online Cambridge dictionary:

« The process of getting an understanding of something by experience »

The *Tresor de la langue Française informatisé:*

« Acquérir la connaissance d'une chose par l'exercice de l'intelligence, de la mémoire, des mécanismes gestuels, appropriés, etc. »

« Acquire the knowledge of something through the practice of intelligence, memory, appropriate gestures, etc. »

But theses definitions are biased toward a human definition of the learning process...

Attempt of learning definition

In a more generic way, **animal learning** is usually summarized by these capabilities:

- Sensitivity **- Memory** **- Adaptation** **- Generalization**

By adopting the proposed **statistical learning** definition, any algorithm that works using the following elements can be considered as **Machine Learning**:

- An adaptable input construction, which represents the sensitivity of the method.
- A form of memory to remember either the previous situations, or a reduced version of them.
- A way to estimate if its behavior (output prediction) is appropriate.
- A way to adapt its behavior during the learning process.
- A way to generalize its behavior to new outputs.

Keywords to explore the notion of Artificial Intelligence from there (beyond the scope of this course) :

- Symbolic and sub-symbolic representations *- Artificial General Intelligence* *- Strong AI*

Types of Machine Learning

Supervised

Learn from labeled examples. Try to find a generalization to get correct prediction most of the time.

Semi-Supervised

Usually a mix of two algorithms. A un-supervised one first that minimize the definition biases. A supervised one that remap to the expected groups.

Un-Supervised

Dataset with examples is provided but with no labels. Try to find similarities in the input in order to create groups.

Self-supervised

Dataset with no external labels. Use the data themselves to define labels and learn using a supervised formalism. (e.g., auto-encoders)

Reinforced

Based on an reward function. Used when there is no easy error function. Corrections are applied based on an external performance measurement.

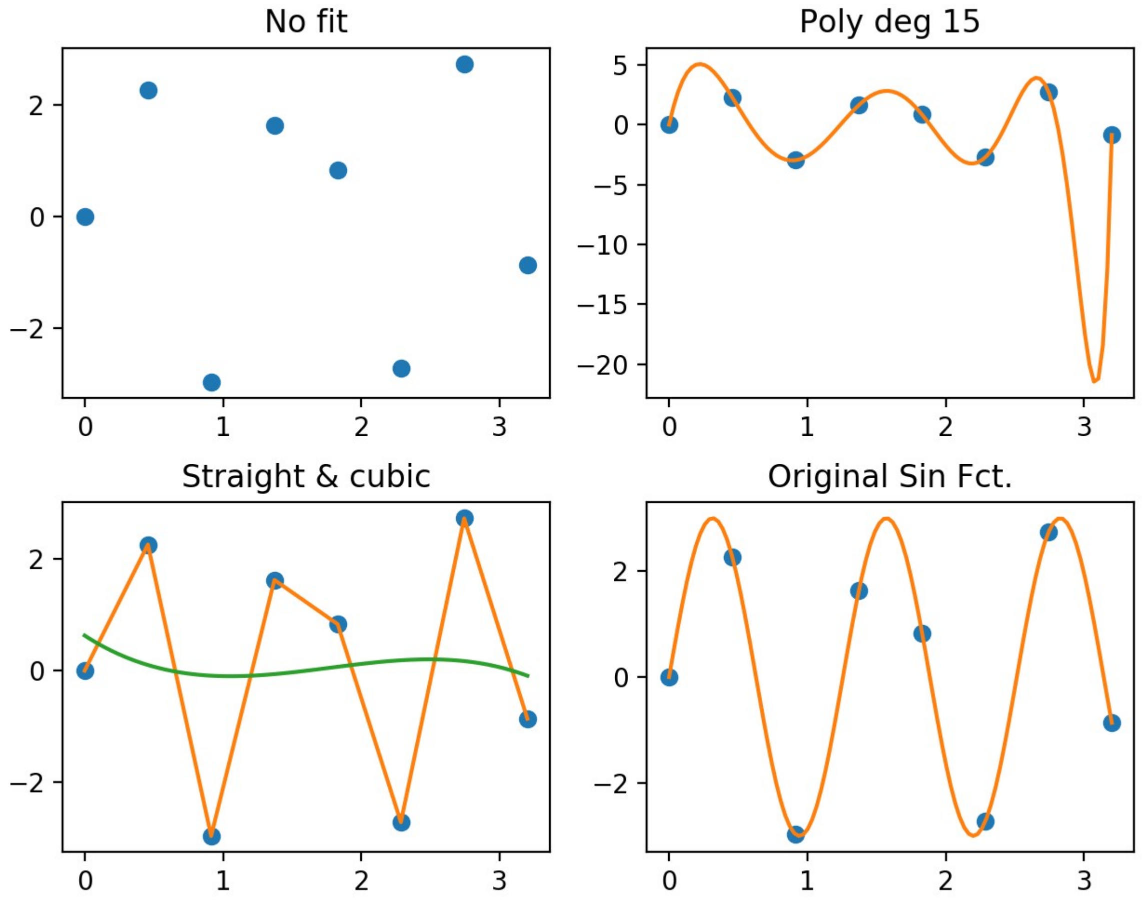
Evolutionary

Similar to reinforced, explore the possibility space based on biological evolution processes. Selection effects, reproduction, mutation, etc.

Use cases

Simple example of regression

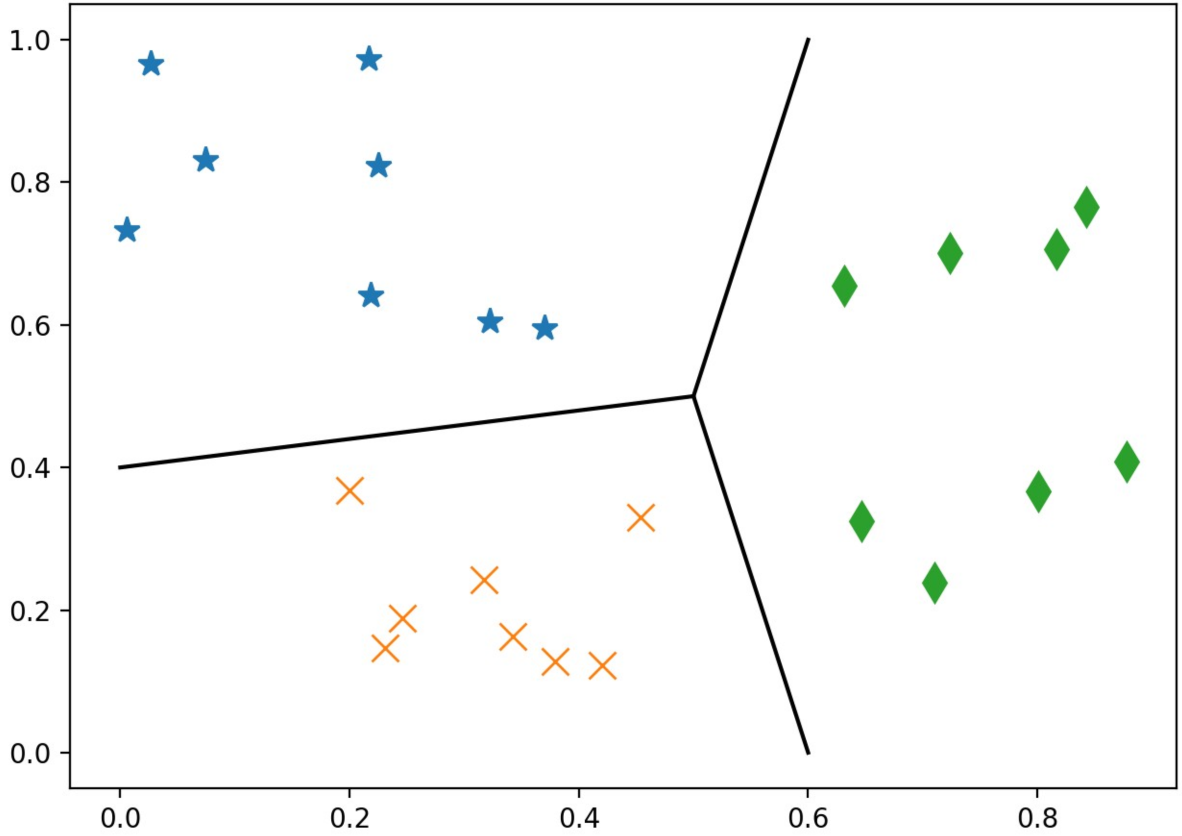
Reconstruct functions from a sample of data points (in any number of dimensions)



Use cases

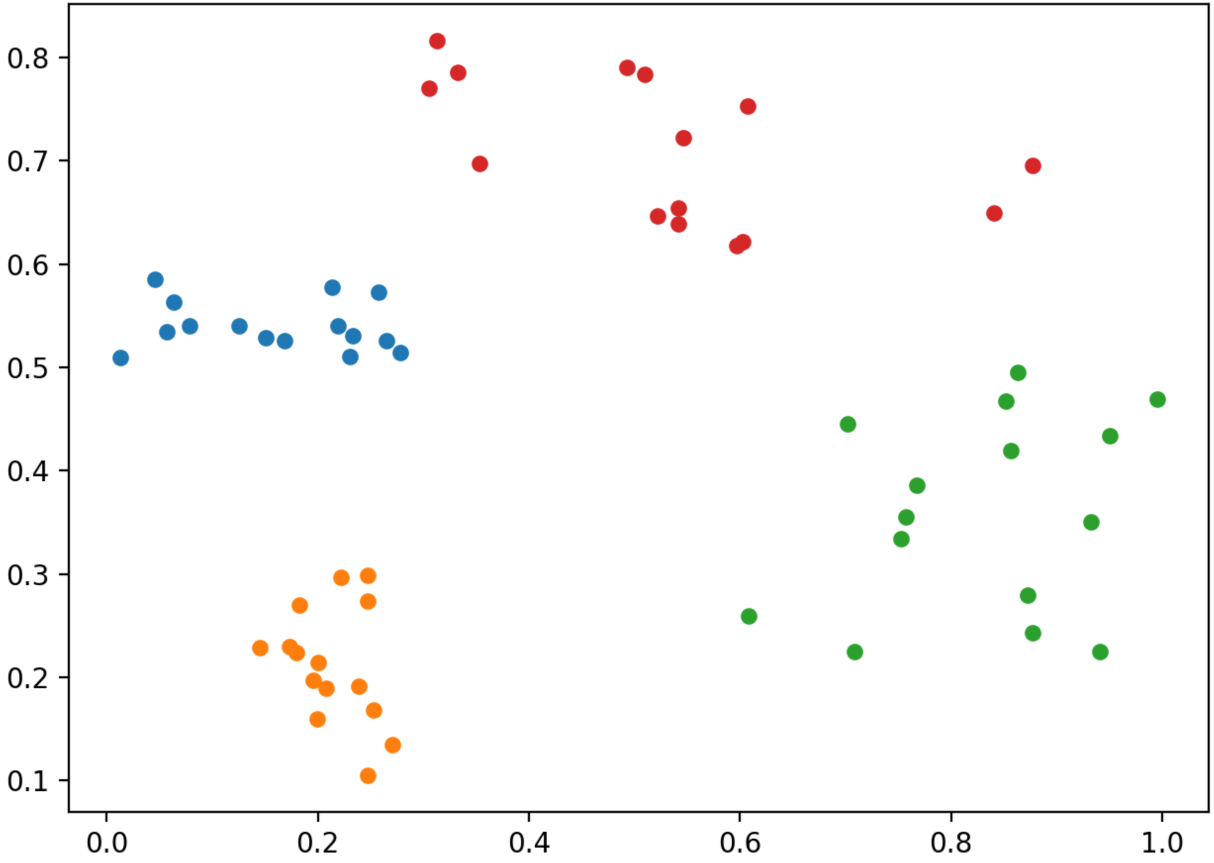
Simple example of classification

Decide from an input vector with several dimensions to which class it should be associated.

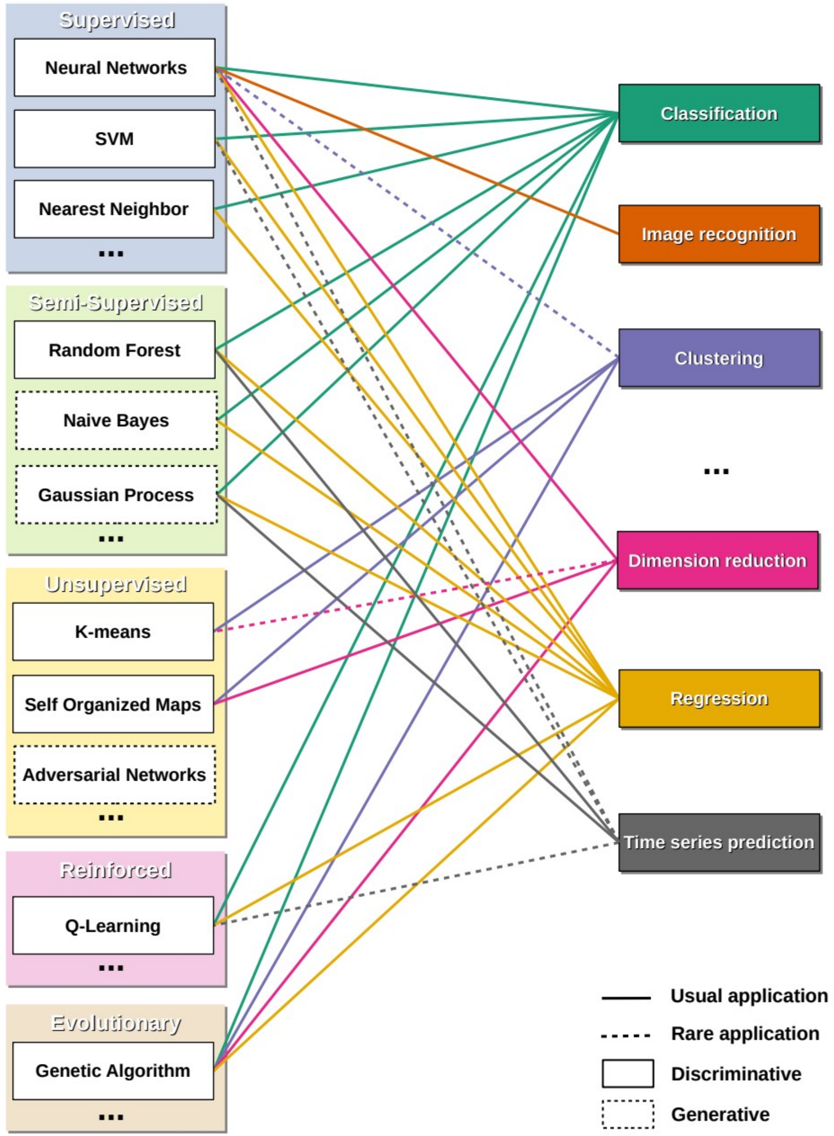


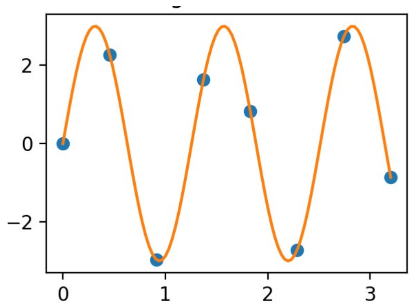
Use cases
Simple example of clustering

Find groups in data based on a predefined strategy (e.g geometric distance) without prior knowledge on the number of groups, nor the solution for any example.

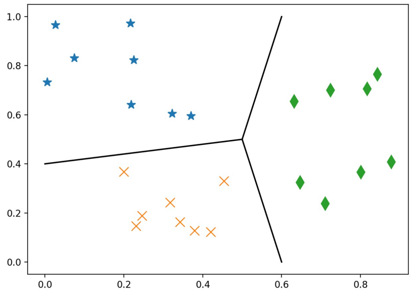


A wide variety of applications ...

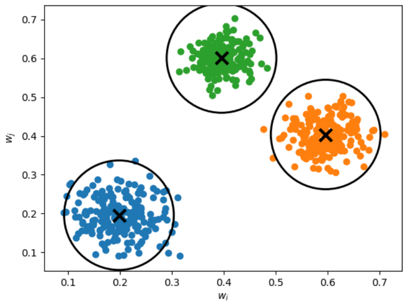




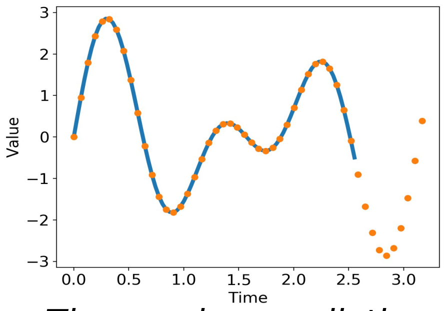
Regression



Classification



Clustering



Time series prediction

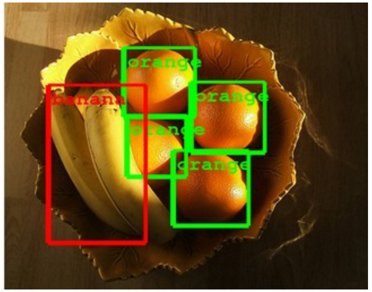
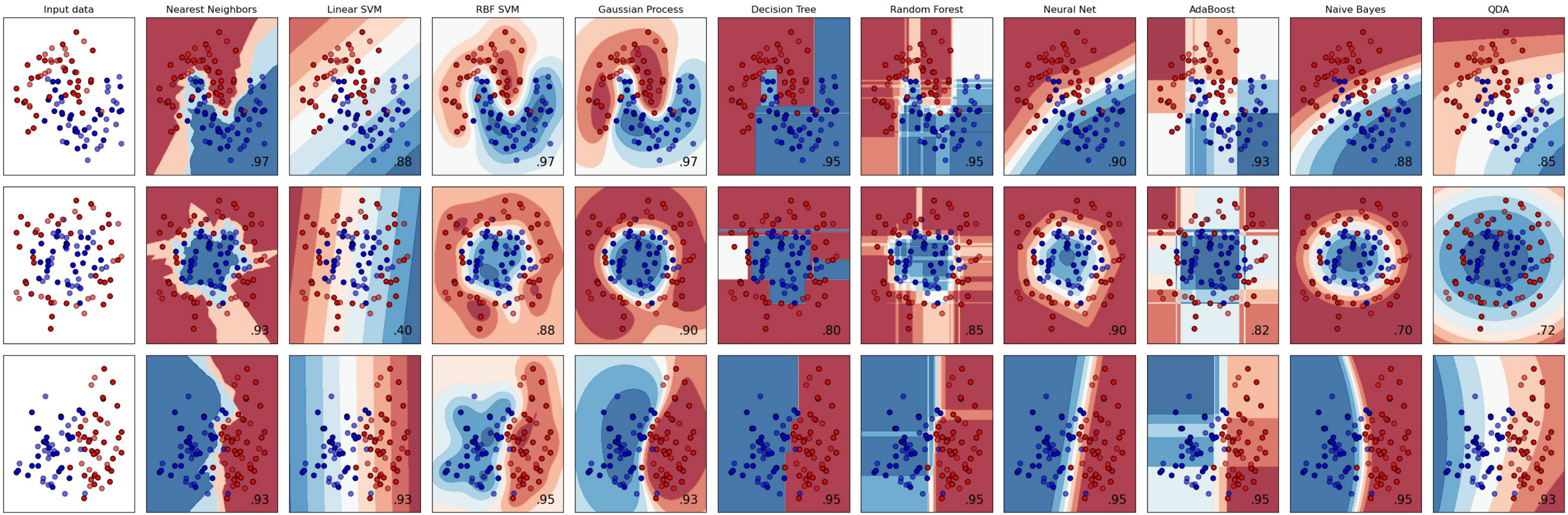


Image recognition

And much more ...

Methods comparison regarding data distribution



Some methods are more suited for specific data distributions, while others work well enough for most data distributions you might encounter.

In any case, every method has its weaknesses that you must identify and understand.

Choosing a method is selecting a balance between:

- The method capability on your data
- The computational efficiency of the method
- The required time of development / integration
- Your own proficiency with the various methods
- ...

How to use Machine Learning ?

Software

Nowadays, there are dedicated ML frameworks for almost every programming language.

They all have specific advantages, but any of them is enough to start learning and using ML.

Hardware

ML methods require a lot of data and computing power during the **training phase**.

However, these methods are very compute efficient when making **predictions**, meaning that it is often possible to deploy already trained models on light hardware (laptops or even modern smartphones)

Still, light resources are usually enough to explore relatively simple ML methods, and many complex problems can be expressed in a simple form, allowing preliminary studies on low-end hardware resources to be conducted.

When training powerful models that work on images, it is helpful to use GPU hardware that is very efficient for most modern ML approaches.

Note that **hardware dedicated to ML methods** are more and more present and accessible.

(Tensor Cores in CPUs and GPUs, Neural Processing Units (NPU), etc ...)

Framework examples: TensorFlow

TensorFlow | Install | Learn | API | Resources | Community | Why TensorFlow | Search

- Overview
- TensorFlow
- TensorFlow JavaScript
- TensorFlow Lite
- TensorFlow Extended
- Swift for TensorFlow

Versions

- Overview
- 2.7 (stable)
- 1.15
- More...

Community supported languages

- Haskell
- C#
- Julia
- R
- Ruby
- Rust
- Scala
- Unsupported

Resources

- Addons
- Datasets
- Federated
- Graphics
- Hub
- Model Optimization
- Neural Structured Learning
- Probability

TensorFlow > API > TensorFlow Core v2.7.0

Was this helpful?

API Documentation

TensorFlow has APIs available in several languages both for constructing and executing a TensorFlow graph. The Python API is at present the most complete and the easiest to use, but other language APIs may be easier to integrate into projects and may offer some performance advantages in graph execution.

A word of caution: the APIs in languages other than Python are not yet covered by the API stability promises.

- Python
- JavaScript
- C++
- Java

We encourage the community to develop and maintain support for other languages with the approach recommended by the TensorFlow maintainers. For example, see the bindings for:

- C#: TensorFlowSharp and TensorFlow.NET,
- Haskell,
- Julia,
- MATLAB,
- R,
- Ruby,
- Rust, and
- Scala.

We also provide the C++ API reference for TensorFlow Serving:

- TensorFlow Serving

There are also some archived or unsupported language bindings:

- Go
- Swift

Framework examples: TensorFlow in Python

Set up TensorFlow

Import TensorFlow into your program to get started:

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

If you are following along in your own development environment, rather than Colab, see the install guide for setting up TensorFlow for development.

Note: Make sure you have upgraded to the latest `pip` to install the TensorFlow 2 package if you are using your own development environment. See the install guide for details.

Load a dataset

Load and prepare the MNIST dataset. Convert the sample data from integers to floating-point numbers:

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Build a machine learning model

Build a `tf.keras.Sequential` model by stacking layers.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

For each example, the model returns a vector of logits or log-odds scores, one for each class.

```
predictions = model(x_train[:1]).numpy()
predictions
```

The `tf.nn.softmax` function converts these logits to *probabilities* for each class:

```
tf.nn.softmax(predictions).numpy()
```

Note: It is possible to bake the `tf.nn.softmax` function into the activation function for the last layer of the network. While this can make the model output more directly interpretable, this approach is discouraged as it's impossible to provide an exact and numerically stable loss calculation for all models when using a softmax output.

Define a loss function for training using `losses.SparseCategoricalCrossentropy`, which takes a vector of logits and a `True` index and returns a scalar loss for each example.

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

This loss is equal to the negative log probability of the true class: The loss is zero if the model is sure of the correct class.

This untrained model gives probabilities close to random (1/10 for each class), so the initial loss should be close to `-tf.math.log(1/10) ~= 2.3`.

```
loss_fn(y_train[:1], predictions).numpy()
```

Before you start training, configure and compile the model using Keras `Model.compile`. Set the `optimizer` class to `adam`, set the `loss` to the `loss_fn` function you defined earlier, and specify a metric to be evaluated for the model by setting the `metrics` parameter to `accuracy`.

```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

Train and evaluate your model

Use the `Model.fit` method to adjust your model parameters and minimize the loss:

```
model.fit(x_train, y_train, epochs=5)
```

The `Model.evaluate` method checks the models performance, usually on a "Validation-set" or "Test-set".

```
model.evaluate(x_test,  y_test, verbose=2)
```

The image classifier is now trained to ~98% accuracy on this dataset. To learn more, read the TensorFlow tutorials.

If you want your model to return a probability, you can wrap the trained model, and attach the softmax to it:

```
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
```

```
probability_model(x_test[:5])
```

Conclusion

Congratulations! You have trained a machine learning model using a prebuilt dataset using the Keras API.

Framework examples: TensorFlow in R

```r
library(keras)

# Data Preparation -----------------------------------------------------

batch_size <- 128
num_classes <- 10
epochs <- 12

# Input image dimensions
img_rows <- 28
img_cols <- 28

# The data, shuffled and split between train and test sets
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# Redefine  dimension of train/test inputs
x_train <- array_reshape(x_train, c(nrow(x_train), img_rows, img_cols, 1))
x_test <- array_reshape(x_test, c(nrow(x_test), img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)

# Transform RGB values into [0,1] range
x_train <- x_train / 255
x_test <- x_test / 255

cat('x_train_shape:', dim(x_train), '\n')
cat(nrow(x_train), 'train samples\n')
cat(nrow(x_test), 'test samples\n')

# Convert class vectors to binary class matrices
y_train <- to_categorical(y_train, num_classes)
y_test <- to_categorical(y_test, num_classes)

# Define Model -----------------------------------------------------------
```

```r
# Define Model -----------------------------------------------------------

# Define model
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu',
                input_shape = input_shape) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = num_classes, activation = 'softmax')

# Compile model
model %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)

# Train model
model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_split = 0.2
)



scores <- model %>% evaluate(
  x_test, y_test, verbose = 0
)

# Output metrics
cat('Test loss:', scores[[1]], '\n')
cat('Test accuracy:', scores[[2]], '\n')
```

What about from scratch programming ?

« Opening the box »

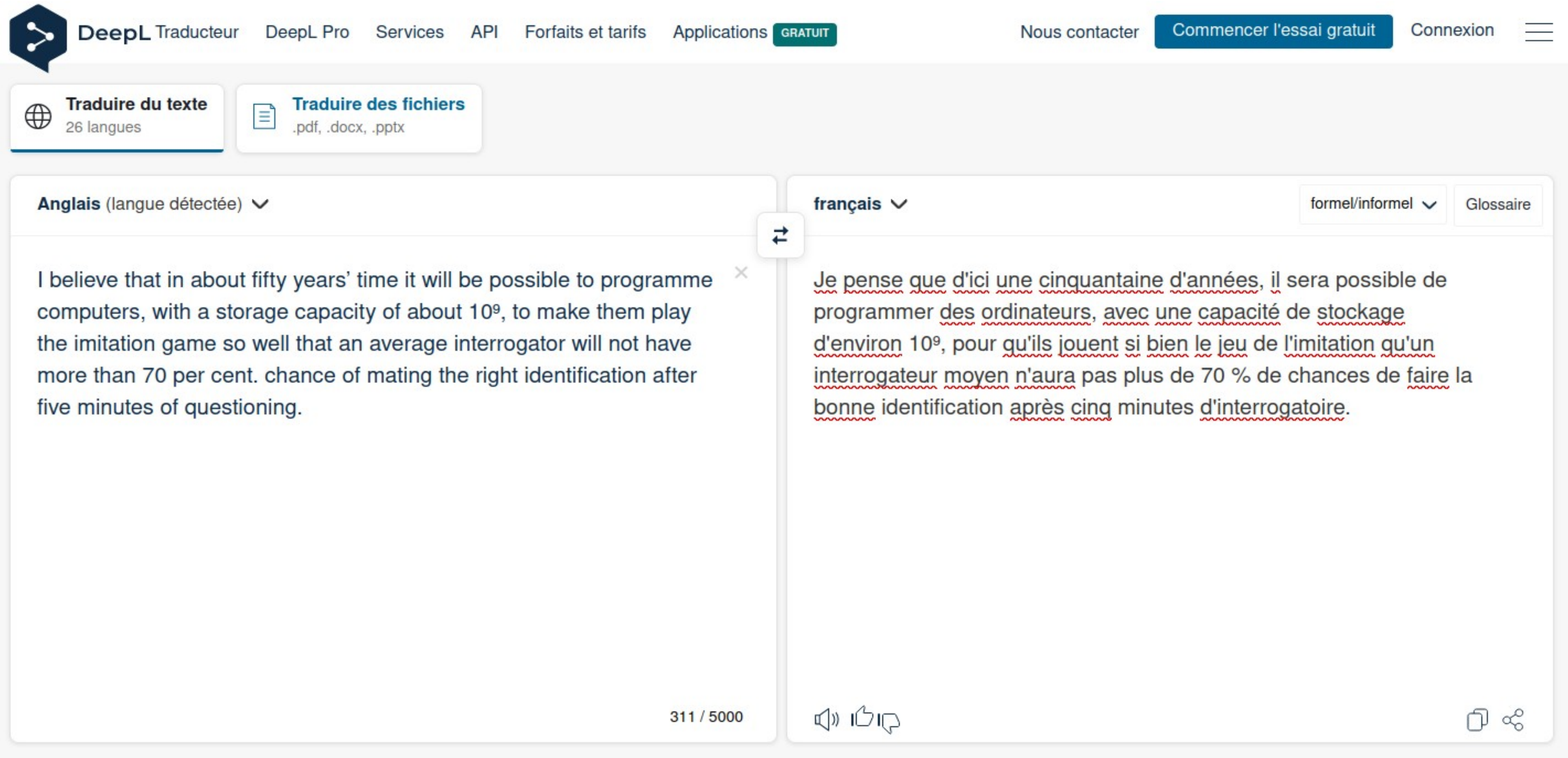
	Frameworks	Custom
Pros	• Lots of learning material and support • Often already parallel and optimized • Low effort for good results • Updated regularly (performance improvement without changes in your own code)	• May grant a deeper understanding • Allow finer control and optimization for specific tasks or hardware • Independent of library lifespan • Grant non-framework-specific capabilities • Can create new methods/paradigms
Cons	• Binned to the supported methods, hardware, and programming paradigm • No control over the supported features and update schedule • If the library dies, all of your code and practical experience dies with it	• LOTS OF SUPPLEMENTARY WORK! • More prone to programming errors • Requires a lot of programming and hardware knowledge to be efficient • Necessity to constantly demonstrate the proper working of your tool

In this course, we will implement simple algorithms AND learn to use widely adopted frameworks.

We will also look at the custom framework CIANNA for specific tasks.

Use cases

Examples of modern applications



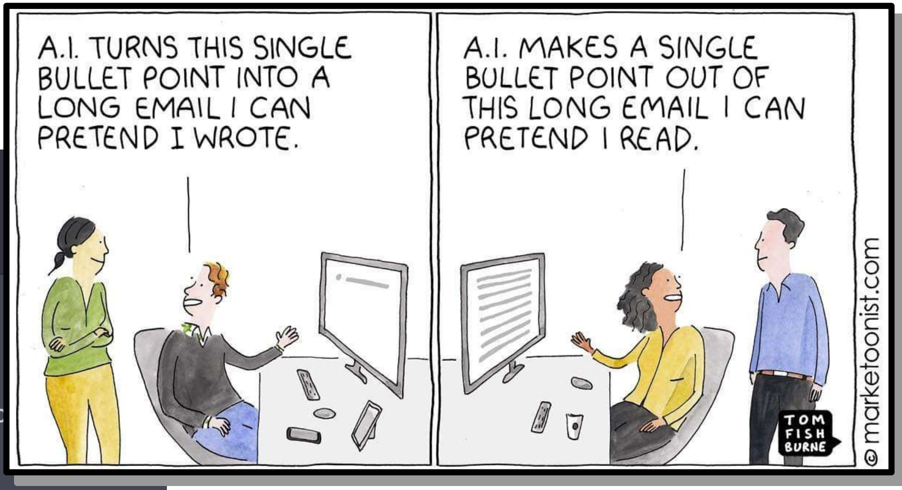
**Private Convolutional Neural Network implementation, trained on the linguee database.*

Use cases
Examples of modern applications

TH Got any creative ideas for a 10 year old's birthday?

Sure! Here are some creative birthday ideas for a 10-year-old:

1. A themed birthday party such as a movie or video game character or a favorite hobb
2. A scavenger hunt with friends, either indoors or outdoors
3. A craft or DIY project, like creating tie-dye shirts or friendship bracelets
4. An outing to an amusement park, trampoline park, or laser tag center
5. A baking or cooking class, where the birthday child and friends can make and enjoy treats together
6. A sleepover with friends, complete with games, movies, and a special breakfast
7. A science or art experiment-themed party, where children can participate in hands-on activities
8. A sporty day with a visit to a rock climbing gym, batting cage, or mini golf course
9. A virtual escape room or game experience, played together with friends over video chat
10. A wildlife or nature exploration outing, such as a hike or a trip to a local zoo or aquarium.

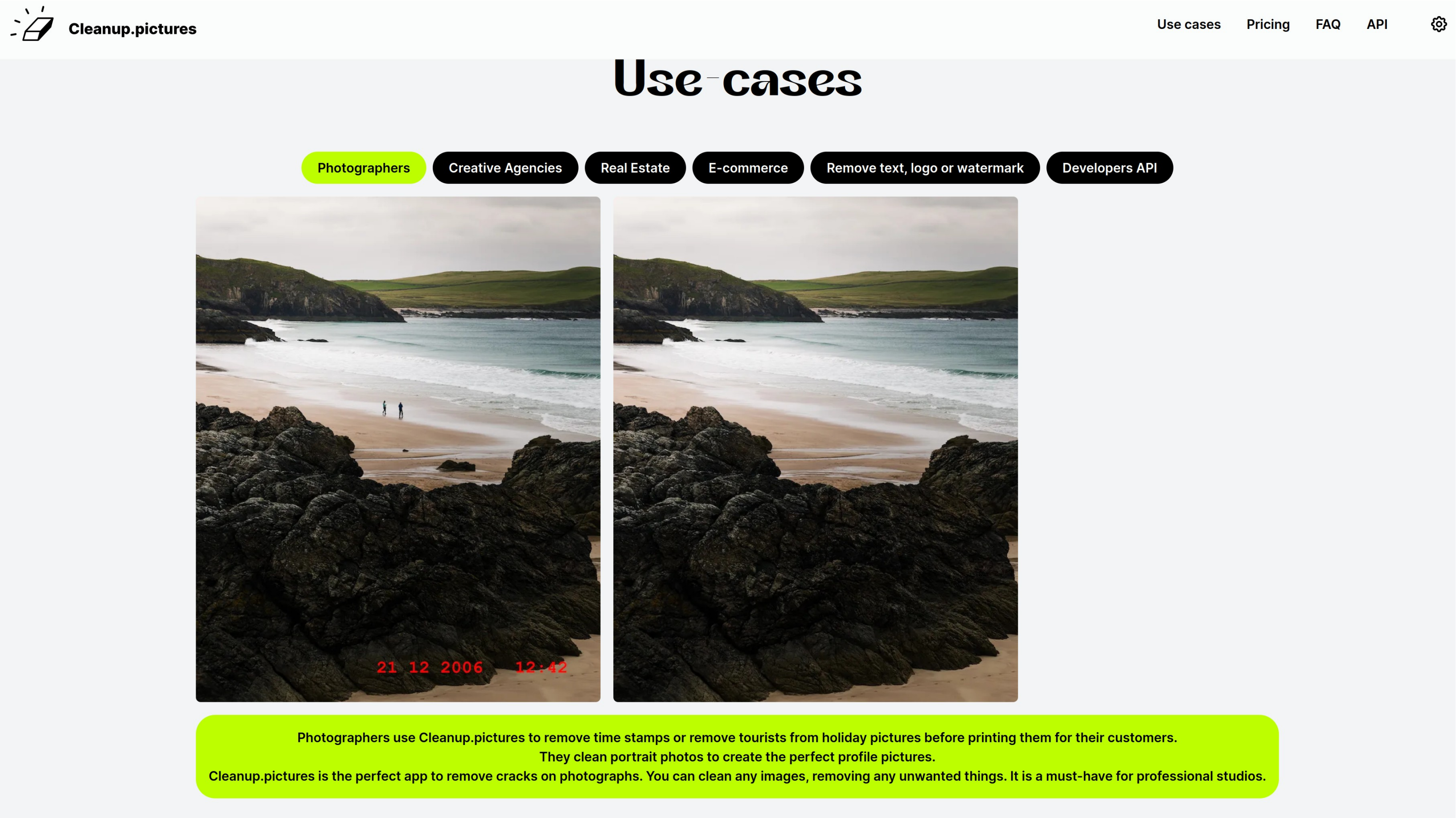


Modern AI LLMs generated text is now quite convincing for human reader, but it remains detectable with automated tools based on … AI LLMs (adversarial attack)!

**ChatGPT, based on GPT3 (Generative Pre-trained Transformer) => https://arxiv.org/abs/2005.14165*

Use cases

Examples of modern applications



**Based on the LaMa: Resolution-robust Large Mask Inpainting with Fourier Convolutions method*
→ https://arxiv.org/abs/2109.07161 and for fun https://cleanup.pictures

Use cases

Examples of modern applications

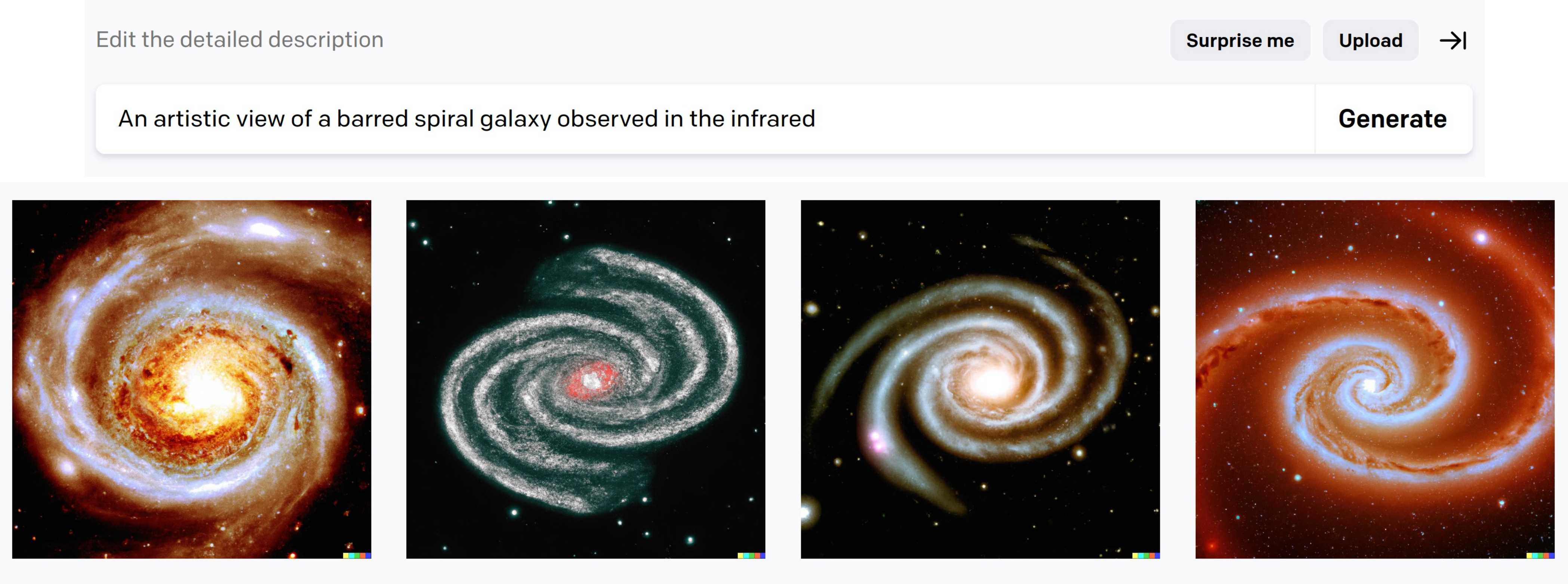
These persons do not exist !

**Based on the Style Generative Adversarial Network model*

→ *https://arxiv.org/abs/1812.04948* *and for fun* *https://thispersondoesnotexist.com/*

Use cases

Examples of modern applications



**Dall-E, based on CLIP word embending and diffusion decodeur => https://arxiv.org/abs/2204.06125*

Use cases

Examples of modern applications

Generate an image of "An artistic view of a barred spiral galaxy observed in the infrared"

Here are the artistic views of a barred spiral galaxy observed in the infrared. The images highlight the central bar structure, swirling spiral arms, and vibrant infrared colors with distant stars and cosmic dust in the background.

Generate a more realistic image that would resemble an astronomical observation

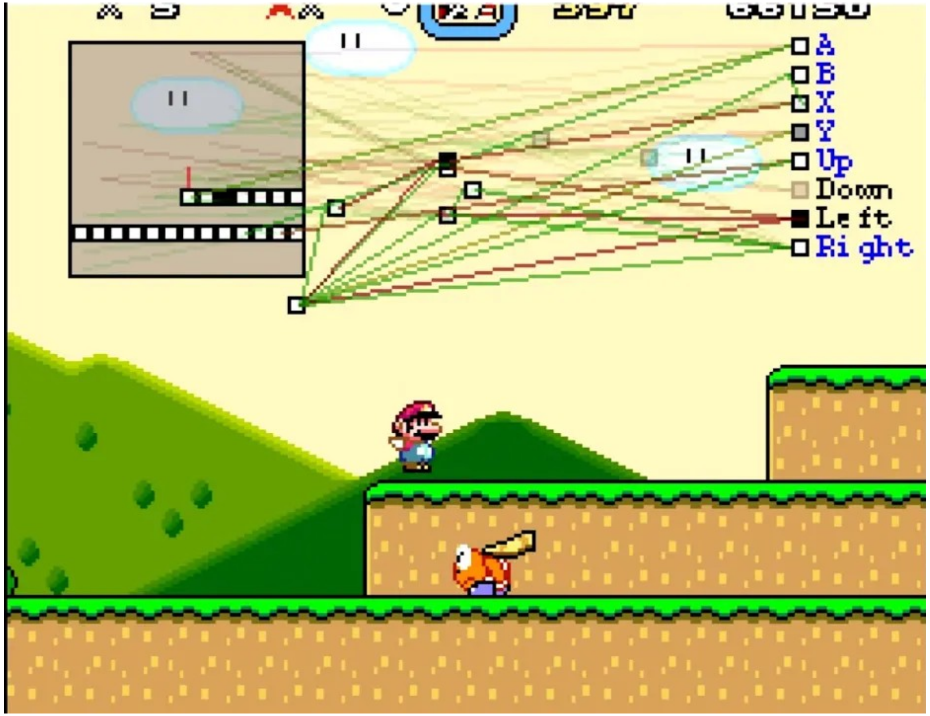
Here are the more realistic images of a barred spiral galaxy observed in the infrared, resembling actual astronomical observations.

With DALL-E 3 through ChatGPT 4 interface

Use cases

Examples of modern applications

The same processes are used to train robots to walk. But can also be used to infer any function from which you can only define a global posterior fitness score.



**Based on Genetic Neural Evolution*
→ https://www.youtube.com/watch?v=qv6UVOQ0F44

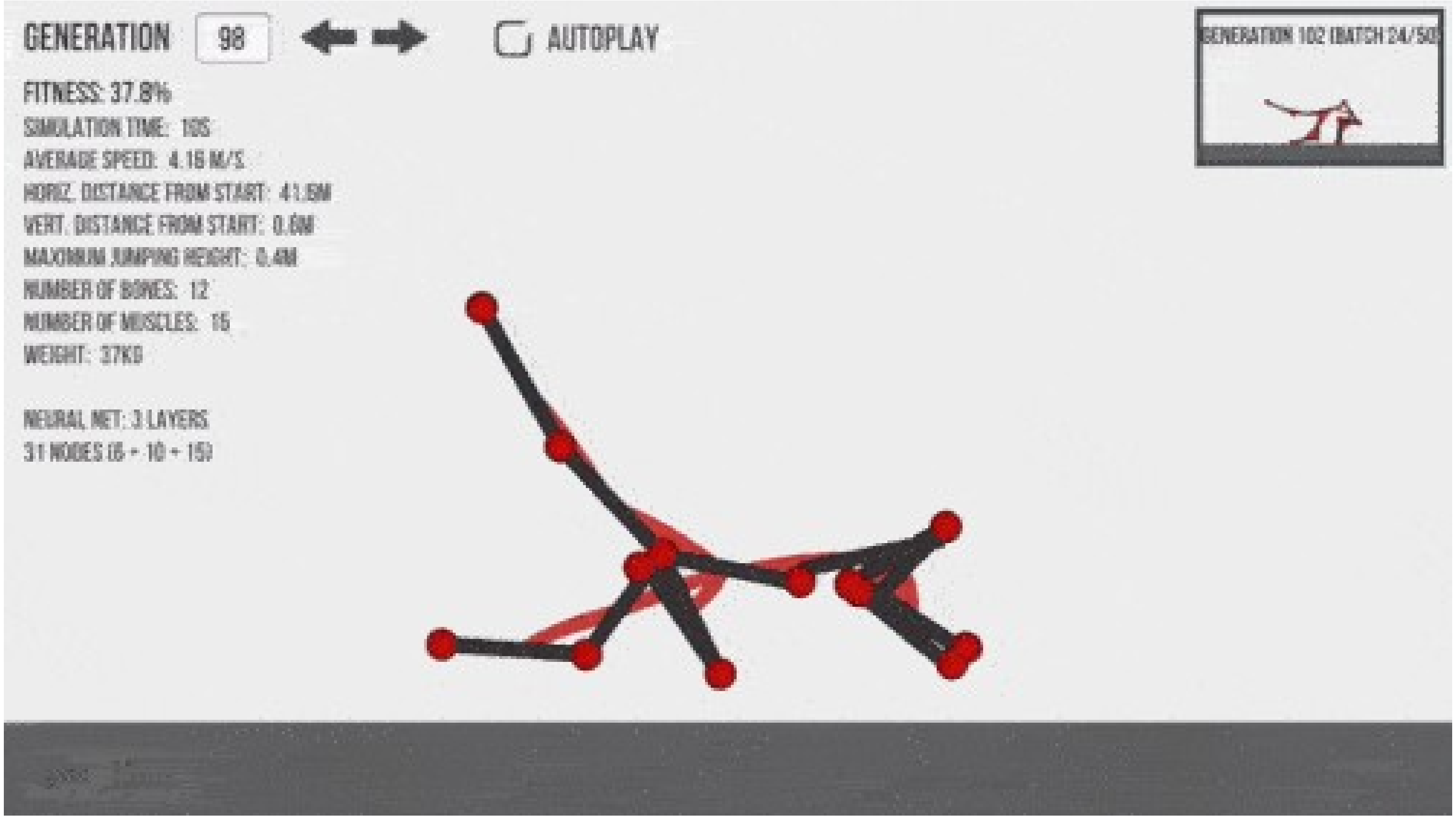
**Based on Deep Reinforced Learning method AC3*
→ https://arxiv.org/abs/1602.01783v2
→ https://youtu.be/nMR5mjCFZCw

Use cases

Examples of modern applications

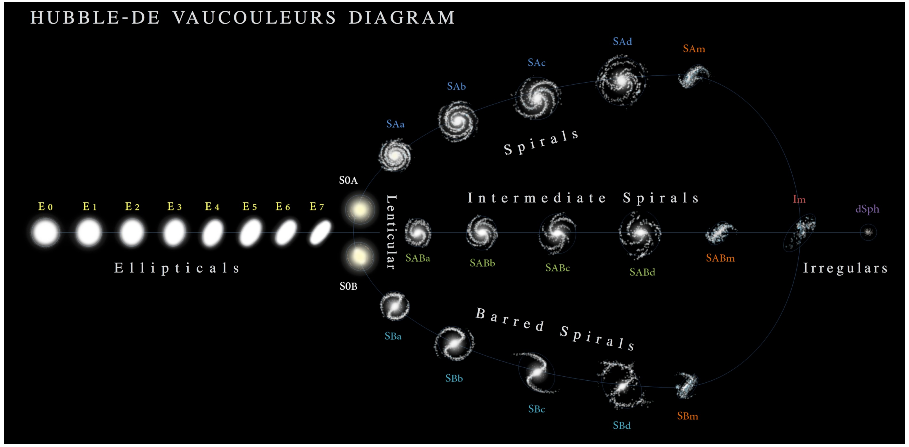
**Based on the YOLO Convolutional Neural Network architecture for object detection*
→ *https://www.youtube.com/watch?v=MPU2HistivI and https://arxiv.org/abs/1804.02767*

Use cases
Examples of modern applications

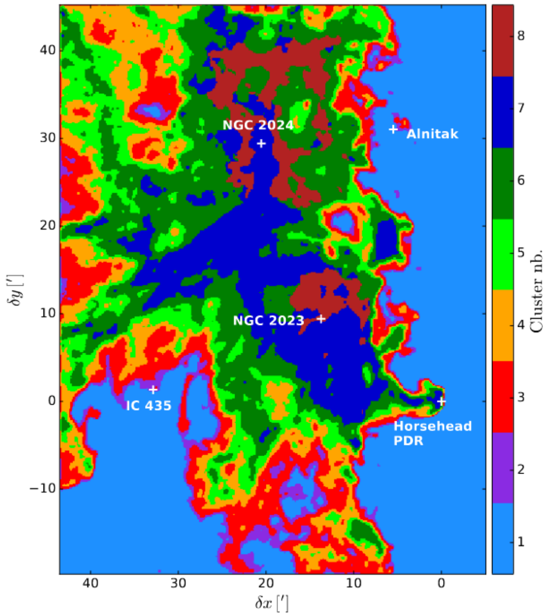


**Based on Genetic Neural Evolution*
→ Interactive creature creation at https://keiwan.itch.io/evolution

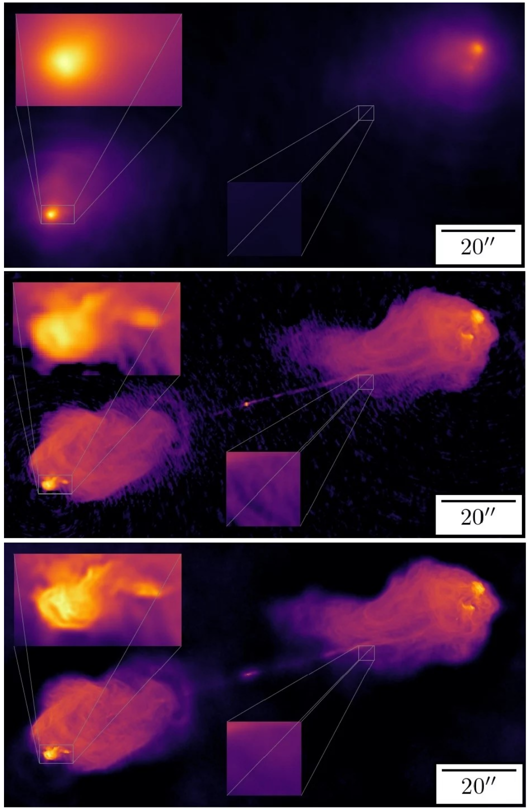
What about astronomical applications ?



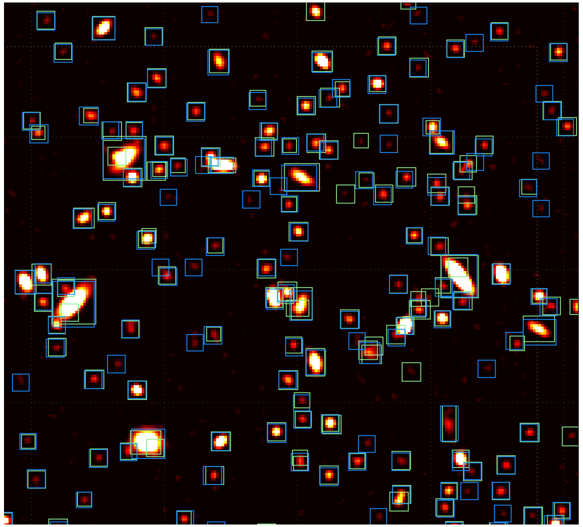
Morphological classification of galaxies



Molecular cloud emission clustering



Radio-astronomical image enhancing



Detection and characterization of galaxies

And much more :

- *Cosmological simulation acceleration*
- *Inverse modeling for cosmological models*
- *Stellar classification based on spectra*
- *ISM turbulence regime classification*
- *Transient events time series prediction*
- *...*

Wide diversity of objects, methods, scales, data representation, etc...

Supervised algorithm example : SVM

Description

The Support Vector Machine algorithm is a supervised ML method used for classification.

Objective

Find the best separation between predefined classes in a multi-dimensional parameter space.

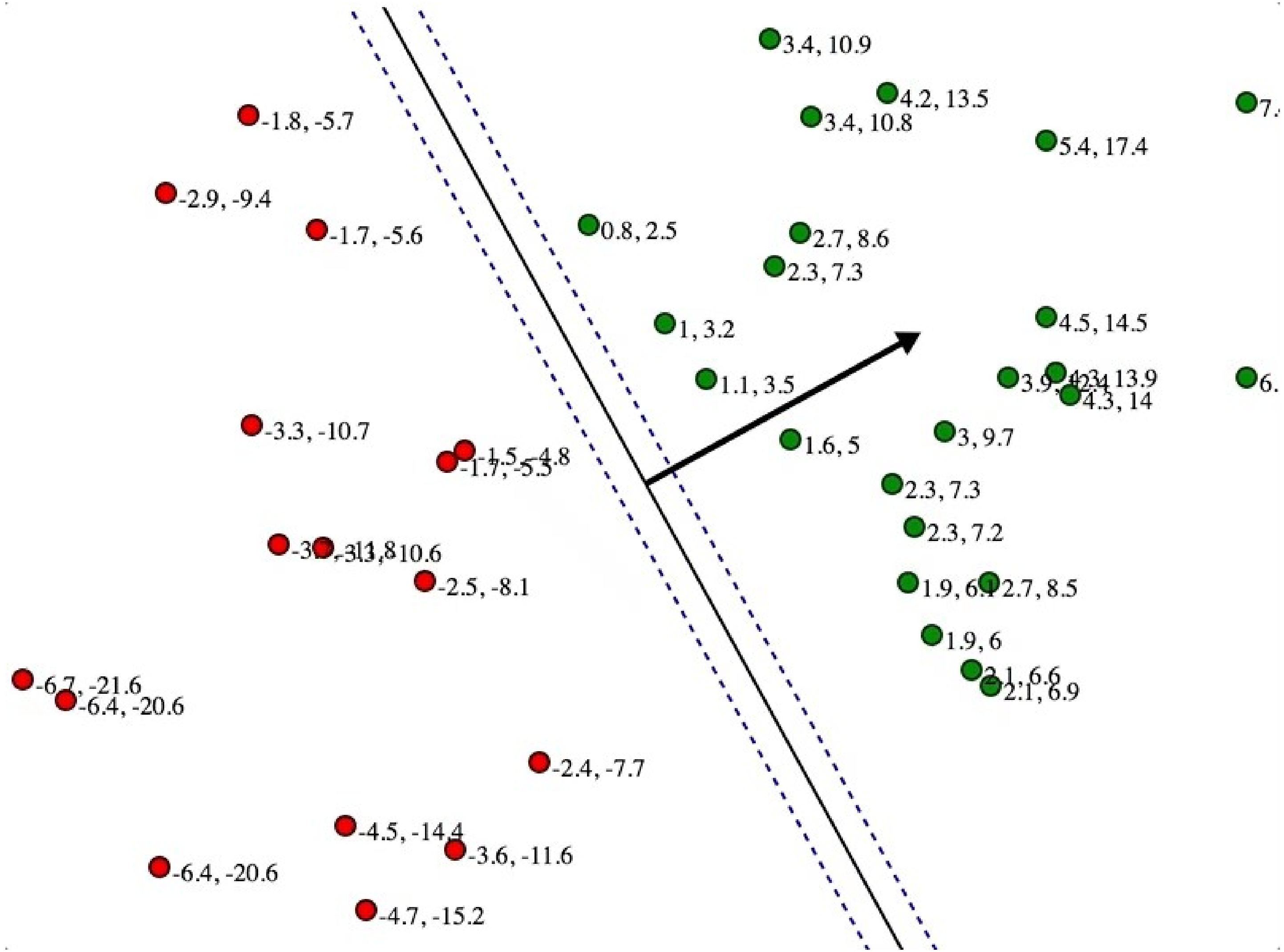
Method (simplified)

Define anchor points representing the closest to the separation between the two classes. The algorithm then searches for the optimal linear separation. For this, it will iteratively try to maximize the orthogonal distance around the separation that does not contain any anchor point.

Why is it ML?

This algorithm respects our definition of ML. It aims to generalize knowledge from specific data points into a continuous parameter space. It is equipped with a metric that evaluates the quality of the splitting and an iterative adaptation scheme based on the orthogonal distance to the anchor. The memory is characterized by the changing value of the linear separation parameters and of the maximum orthogonal distance.

Supervised algorithm example : SVM



Unsupervised algorithm example : k-means clustering

Description

The k-means algorithm is an unsupervised ML method that is used for clustering.
**Not to be confused with the k-nearest Neighbors (kNN) algorithm*

Objective

Find groups in a dataset based on a geometrical distance in a multi-dimensional parameter space.

Method

Define a number “k” of centers (define the maximum number of groups/clusters). For each point, it searches the closest « cluster center » based on a geometrical distance. The cluster centers are moved toward the gravity center of their associated data points. This operation is repeated until the cluster centers stop moving.

Why is it ML ?

This algorithm respects our definition of ML. It aims to generalize knowledge from specific data points into a continuous parameter space. It is equipped with a metric that evaluates the convergence of the method and an iterative adaptation scheme based on the averaged geometrical distance to a list of data points. The memory is characterized by the changing value of the cluster center positions in the parameter space.

Unsupervised algorithm example : k-means clustering

Hands-on : k-means algorithm

Main Loop: until the cluster centers stop moving

- **Identification phase:** loop over the data points
  - Compute the current point distance to all the cluster centers (k)
  - Associate the current point to the closest center

- **Update phase:** loop over the cluster centers
  - Compute the centroid of the data point list for the current cluster
  - Move the cluster center to the obtained centroid position

Hands-on : k-means algorithm

Data: a data file ***kmeans_input_file.dat*** is provided (in 2D and 3D), along with the Python code that was used to generate it.

Code: incomplete codes are provided in Python, C, and Fortran. It includes the reading of the data file and distance compute function. Some useful variables are declared, but some necessary ones are still missing.

Objective: Complete a working k-means program based on the given algorithm.

Key points of the exercise:

- Identify the additional variables that are required for the algorithm.
- Identify the algorithm loops and phases and code them.
- Take the time to think of the best way to find the minimum distance in a list.
- Find how to end the main loop properly.
- While the first version can be specific to 2D data, try to have a non-dimension-specific implementation at the end.

Results: A visualization tool is provided with the script ***kmeans_visual.py***

Provided example 2D data distribution