# Machine Learning 101

Pierre-François Gimenez
CentraleSupélec/Inria

Hands-on Machine Learning for Security
September 24, 2021



CentraleSupélec

# Why machine learning?

Artificial intelligence, machine learning and neural networks are not new (50's) but suffered from cycles of hype, overpromise and disillusion

## Why the current surge of machine learning?

- Cheaper data storage and digitalization of our lives lead to an explosion of available data
- The explosion of processing power, notably with GPU and distributed computing
- New developments in models and learning algorithms

## A few domains that were revolutionized by deep learning

- Image, speech and video processing
- Language translation
- Games playing (Go, StarCraft II)
- Protein folding

CentraleSupélec

# Why machine learning for security?

## Machine learning in security

- Intrusion and anomaly detection
- Malware identification
- Synthetic data generation
- Natural language processing (log summary, spam detection, website analysis)

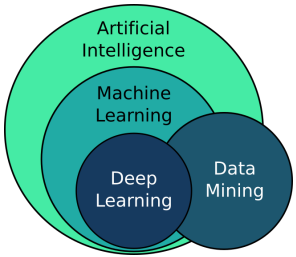
## Data mining for security

- Community identification
- Incident data analysis

## No revolution (yet)

- Lot of research with good progress
- Commercial solutions (IDS, antivirus) don't use much machine learning

# Machine learning? Artificial intelligence? Deep learning?




## Artificial intelligence (AI)

Aims at solving complex problems (e.g. planning with constraints, path finding, knowledge representation)

## Machine learning (ML)

Data-driven construction of a "model" to perform various task after-hand (e.g. intrusion detection)

## Deep learning (DL)

Machine learning techniques based on neural networks with multiple layers. Data-hungry and processing-power-hungry, but generally gets the best performances

## Data mining (or KDD, Knowledge Discovery in Databases)

Using machine learning techniques to explore, understand and interpret the data

CentraleSupélec

# What in this ML 101?

CentraleSupélec

# Organizing the toolbox

CentraleSupélec

# A way to organize ML techniques

## A big toolbox

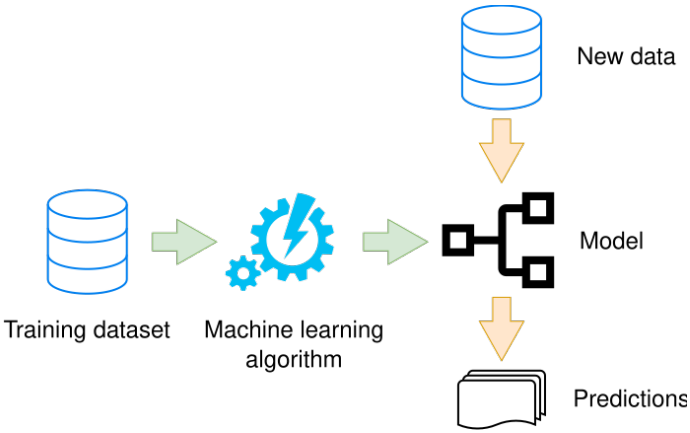
- Machine learning techniques are a toolbox for various problems
- But it's a big toolbox... it's not easy to find what is relevant
- The principal way of organizing this toolbox is to divide it into "problem types"

## Learning settings

- These problems types are called *learning settings*
- A learning setting specifies what is the goal of the ML technique and on what data it relies
- Most methods are adapted for one learning setting, so identifying the correct learning settings is generally the first think to do!

But first, we need to talk about how ML works in general

# The principle of Machine Learning




Two main steps in machine learning

- In the learning phase, a learning algorithm transforms a training dataset into a model
- In the inference phase, the model makes predictions on new data

# Supervised learning

## Supervised learning

In the training set, there is a *label* for each instance. The label is what we seek to predict.

- If the label is continuous, this is a *regression* task. For example: predict the number of inbound IP connections
- If the label is discrete, this is a *classification* task (called *detection* with two label values). For example: predict the family of a malware, detect a spam

## Example

- In security, labels are generally "benign" or "attacks", or some more specific kind of attacks (DDoS, worm, spear fishing...)
- If we can have access to data with labeled attacks, we could use supervised learning to create an IDS

# Unsupervised learning

## Unsupervised learning

There is no label in the training set

- *Outlier detection* task: the goal is to identify instances that don't resemble the others
- *Clustering* task, where similar instances are grouped together
- *Dimension reduction* task: the goal is to reduce the size (dimension) of each instance while retaining as much information as possible
- *Data generation* task: based on some training data, generate new data "close" to them

Sometimes learning and inference phases are merged

## Example

- In security, these tasks are generally done to better understand the data and preprocess it
- Outlier detection can be used to create an anomaly detector that monitors a large and homogeneous population without learning phase

# Semi-supervised learning

## Semi-supervised learning

A catch-all category for learning task with missing labels.

- *Positive-Unlabeled* (PU) learning, where some instances are labeled and others are not. The goal is to label all of them
- *One-class classification*, where all the data belong to only one class
- *Active learning*: the learner asks for labels

## Example

- PU learning can be useful when manual labeling is difficult or costly (e.g., in a SIEM)
- One class classification is useful in anomaly detection, where we only have data from legitimate behavior. It's the base of ML-based behavioral IDS.

# Some other ML tasks...

## Some other tasks

- *Learning to rank*: given a list of items, predict their ranking
- *Reinforcement learning*: when an agent can interact with its environment and receive some "reward" based on its actions. The goal of this learning settings is to understand what action to do depending on the circumstances

## And many variants

- *Federated learning*: privacy-preserving distributed machine learning technique
- *Transfer learning*: use available pretrained model for a related task
- *Few-shot learning*: classification with little training data
- *Multi-label learning*: one instance can correspond to multiple labels at the same time

# The steps of machine learning

CentraleSupélec

# The steps of machine learning

1. Data collection
2. Data preparation
3. Model selection and tuning
4. Model training
5. Model evaluation

# 1. Data collection

## Data collection

- The data collection typically involve experiments
- Depending on the setting, label should be present or not (supervised or not), there should be one class or more, etc.
- Seek to observe as many different phenomenon as possible
- Try to have representative data, i.e., to respect the proportions of phenomenon (e.g. the proportions of malware families)

## Some advice

- Reuse a public dataset when possible!
- In supervised learning: severe class imbalance can cause learning issues
- In unsupervised learning: focus on the diversity of data to get a more informative training dataset by varying the experimental settings

# 1. Data collection

## What to measure?

- As an expert, you should decide what to measure
- Try to measure anything that could help the task
- It's generally easier to remove measures afterward than to rerun experiments with new ones

## Garbage in, garbage out

- The output of ML can only be as accurate as its input
- Always some human or technical errors in experiments
- Especially critical for decision help, as biased input will produce biased output

# 2. Data preparation: feature engineering

## What input for ML learning algorithms?

- Data must be structured in vectors
- Each vector has a *fixed number* of "features".
- Data with variable length (network packet, function parameters, etc.) must be transformed:
  - Padding fills empty features with null values
  - You can use statistically descriptive features (mean, variance, n-grams, etc.) to summarize the data
- Each feature can be
  - discrete: finite number of arbitrary values (categories)
  - continuous: a quantity with meaningful mathematical operations like $+$ and $>$

# 2. Data preparation: feature selection

## Example

- One vector is one network packet. The features we select can include:
  - Length of the packet (continuous)
  - Number of occurrences of some trigrams (continuous)
  - Whether source IP is public or private (discrete)
  - Server port (21, 80, ...) is a discrete feature, even though they are numerical values

## Continuous and discrete feature conversion

- Conversion from discrete to continuous feature can be done with "one-hot encoding": if it has 5 distinct values, replace it with 5 features that can be either 0 or 1
- Conversion from continuous to discrete feature is done by merging intervals into categories
- Notably useful for models that only deals with discrete or continuous features (like neural networks)

# 2. Data preparation: feature selection

## Unsupervised feature selection

- Doesn't rely on the targeted label
- Remove redundant features (features with high correlation)
- Remove features with low variance
- (Can also be used in supervised learning!)

## Supervised feature selection

- Rely on the targeted label
- Remove irrelevant features that are independent of the label
- Can be an iterative process: evaluate the performance of a feature subset, modify the subset and evaluate again

# 2. Data preparation: preprocessing

## Classical preprocessing

- Remove:
  - duplicates
  - NaN
  - erroneous data
  - outliers (if you think they are erroneous)
- Consider regrouping some categorical values if they are too rare
- Standardization (ensure that every features has the same mean and variance) may be necessary for distance-based models

# 3. Model selection

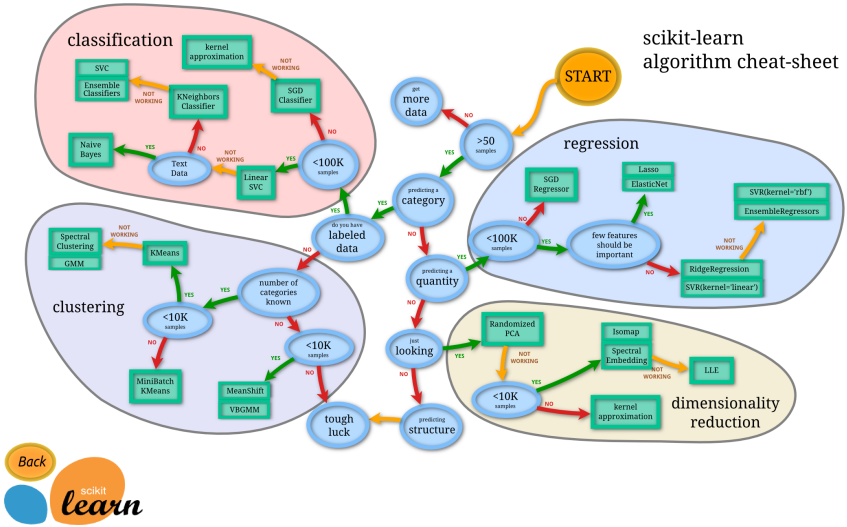
## Advice

- Remember the learning setting (regression, classification, outlier detection...)
- Identify the particularities of your task (temporal analysis, explainable prediction, text analysis...)
- Find the algorithms that fits your needs
- Try them, from the simplest to the most complex
- Don't "overcommit" to one algorithm, be agile

## Why starting from the most simple?

- Simple models / models with lower complexity generally need less data
- They tend to be more robust, i.e. their performance is more consistent over different data distributions
- They are generally easier to parametrize, to learn, and to interpret

# Advice of scikit-learn



# 5. Model evaluation

## Cross-validation for supervised learning

The standard evaluation method

- Use 90% of the dataset to learn a model and 10% to evaluate it
- Do it 10 times to evaluate on all the dataset

Not always possible depending on the domain

## Evaluation metrics

They depend on the task

- Classification: confusion matrix
- Detection: ROC analysis, AUC (area under curve)
- Regression: mean square error

# 5. Model evaluation

To improve your performances, analyze the errors

- Visualize the model. Does it make sense? (not always possible!)
- Analyze the errors
  - Do you need more data for this particular case of errors? $\Rightarrow$ rerun experiments
  - Is the model able to handle these? $\Rightarrow$ test another model
  - Is the evaluation score adapted? $\Rightarrow$ not all error have identical importance
  - Is it an edge case that maybe doesn't need "fixing"? $\Rightarrow$ don't try to "fix" all small errors
  - Is there some missing information (features) for this particular case? $\Rightarrow$ if two categories of attacks are difficult to set apart, think about what information would help the classifier

CentraleSupélec

# AutoML: automation's automation?

## AutoML

- AutoML aims at automating (or at least simplifying) multiple steps:
  - Data preparation
  - Feature selection
  - Model selection
- Bring ML to non-experts
- Can be used as a basis for ML experts
- A relatively new domain of research

## Implementations

- auto-sklearn, MLBox, TPOT (open source)
- Google Cloud AutoML, AutoGluon (Amazon, open source), Azure Machine Learning (Microsoft)

CentraleSupélec

# Some families of models

# k-NN (supervised learning)

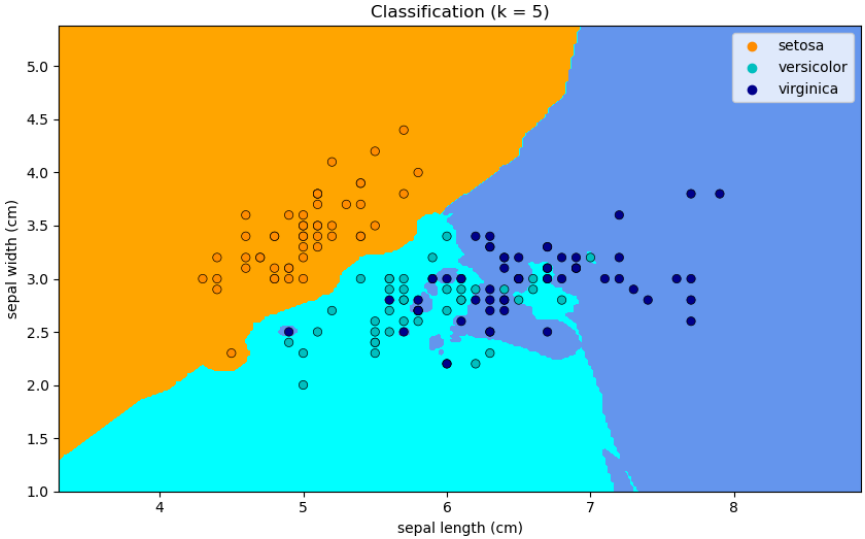
## Principle

To predict the class of an instance, check the $k$ closest instances ($k$ nearest neighbors) in the training set and output the most common classes among them

## Remarks

- Good performances, interpretable output
- Generally used with continuous features
- Normalization of the continuous features is necessary!
- Vote can be weighted by the distance
- Start with $k = \sqrt{|training\_set|}$
- Variation: LOF (based on local density) for outlier detection

# k-NN example


Classification (k = 5)
setosa
versicolor
virginica
sepal width (cm)
sepal length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

# Naive Bayes (supervised learning)

## Principle

Modelize the distribution of the features for each class. For some given data, can give the probability that some class explains such data.

## Remarks

- Moderately good performances due to drastic statistical assumptions
- This model can answer "I don't know" when all probabilities are low (their sum is not equal to 1)
- Easy to interpret, low time/space complexity

CentraleSupélec

# Decision tree (supervised learning)

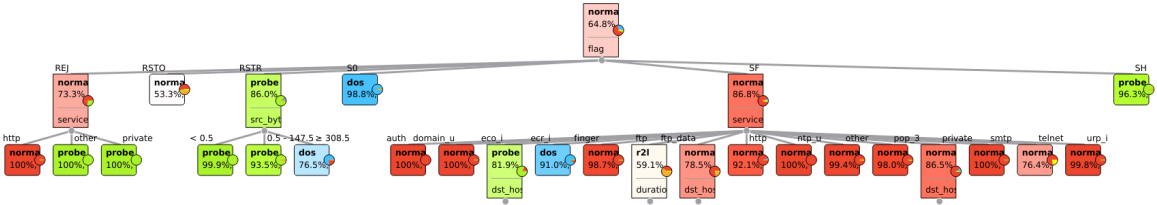
## Principle

- Decision tree are a set of rules in the form of a tree
- Each node contains a simple rule of one feature, like "A=val1" or "B>0.5"

## Remarks

- Very easy to interpret, not the best performances
- Depending on the implementation, can handle continuous data or not
- Works with little data
- Robust to irrelevant or redundant features

# Decision tree example



CentraleSupélec

# Random forest (supervised learning)

## Principle

Learn a bunch of decision trees based of different subset of the training set. Gather the predictions of the classes and output the most common one.

## Remarks

- High performances, difficult to interpret
- Can be used to estimate the relevance of the features
- Variation: Isolation Forest for outlier detection

# Support Vector Machine (SVM, supervised learning)

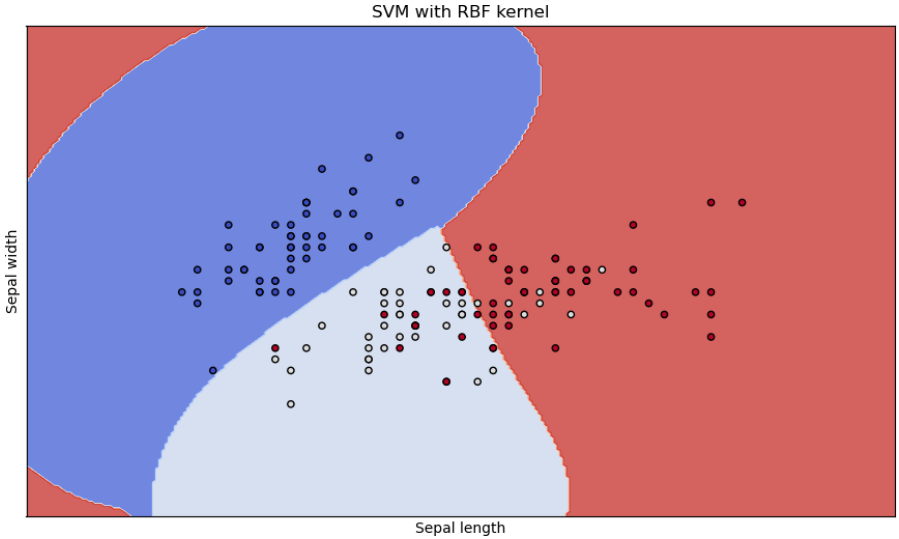
## Principle

Originally, they separate two classes with a linear separator. Now they use the "kernel trick" to separate classes with more complex shapes.

## Remarks

- High performances, difficult to interpret
- Now designed to work with any number of classes
- Normalization of the features is necessary
- Variation: OCSVM (one-class SVM) for one-class classification

# SVM example



# Neural networks (supervised learning)

## Principle

Each neuron does some simple computation. Neurons are arranged in layer, where the output of layer N is the input of layer N+1. Generally outputs continuous numbers (regression task) or a probability distribution (classification task)

## Remarks

- The best performances... if you know how to parametrize it
- Can crunch GB of data but needs good processing power (best if GPU)
- Robust to irrelevant and redundant features
- Impossible to interpret without an expert
- Only works with continuous features
- Variations:
  - Autoencoder for one-class classification and dimension reduction
  - Variational autoencoder and Generative Adversarial Networks for data generation

CentraleSupélec

# Ensemble techniques

## Principle

Multiple techniques that leverage multiple models at once:

**Bagging** Multiple models of the same class trained on subset of the data

**Boosting** A sequence of models where each one handle the wrong instances of the previous ones. AdaBoost and XGBoost are classical examples

**Stacking** Multiple base models of different class trained on the whole dataset. Another model learns the final predictions based on the predictions of the base models

## Remarks

- Random forest are a popular example of bagging
- These techniques lower the explainability
- Not always worth the hassle

CentraleSupélec

# Hurdles in ML

CentraleSupélec

# Why is it difficult to find a good model?

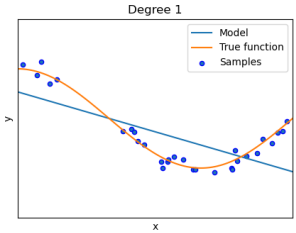
## What is a model

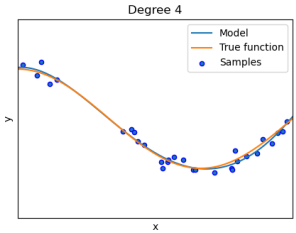
- A model is a simplification of the reality
- A model can be created by an expert, derived from a specification or inferred from data
- In ML, a model is generally inferred by a *learning algorithm* that searches a model in a model class (e.g., a learned SVM is a *model* and the set of all SVMs is the *model class*)
- Not all ML techniques use explicitly a model (e.g., k-NN)

## Where does the error comes from?

In classification and regression, the error of a model can decomposed in two parts: its *bias* and its *variance*

# The error of a model: the bias



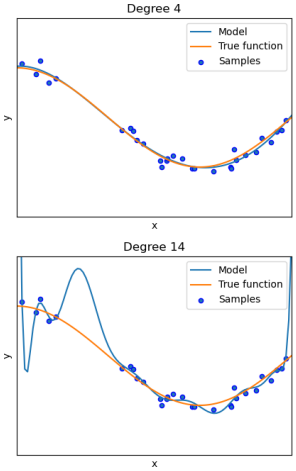

## Bias

The *bias* is the distance between the reality and the closest model in the model class

## Example

- The true data follows a sinusoid and are noised
- The straight line has a high bias: it *can't* approximate the reality very well
- The 4-degree polynomial has a low bias: it can approximate the reality

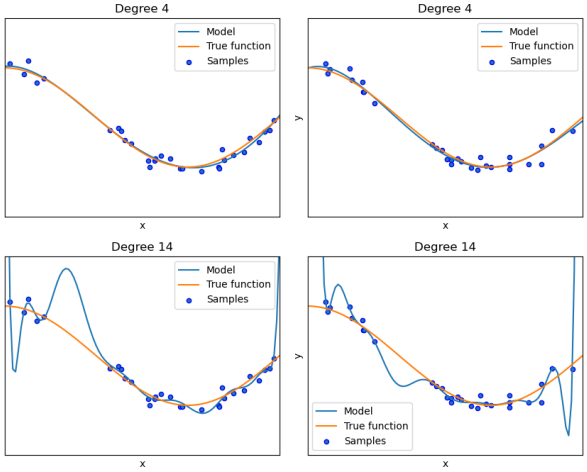
# The error of a model: the variance




**Variance**

The *variance* is the sensitivity of the model class to the training data

# The error of a model: the variance




## Variance

The *variance* is the sensitivity of the model class to the training data

## Example

- The 4-degree polynomial has a low variance (low sensitivity)
- The 14-degree polynomial has a high variance (high sensitivity) and a low bias (it *could* fit the data with little error)

# Overfitting and underfitting

## And what?

- Simple model class (with a few parameters) have a high bias but low variance. They have generally good generalization properties (Occam's razor)
- Complex model have a low bias but high variance. They fit closely the dataset but have issues with generalization
- Ensemble techniques generally reduce variance and bias (but lower the explainability)

## Two classical problems

- Underfitting: where the error mainly comes from the bias $\Rightarrow$ experiment with a model class with less bias
- Overfitting: where the error mainly comes from the variance. The generalization capability is poor and **the performance on the test set is lower than the performance on the training set** $\Rightarrow$ see next slide

CentraleSupélec

# How to remove the overfitting?

## For neural networks

- Decrease the number of learnable parameters (e.g., remove a layer)
- Add some dropout layer
- Use an variational autoencoder instead of an autoencoder

## For other model classes

- The parameter $k$ in $k$-NN controls the bias-variance. Reduce $k$ for lower variance and higher bias
- In statistical models, BIC (or AIC) score should limit the overfitting
- In decision trees, limit the maximal depth
- In random forests (and other ensemble techniques), reduce the number of base models
- ... or get more data

Overfitting can also happen if you tweak a lot the (hyper-)parameters of a model!

# Security of ML

**Using ML opens a new attack surface for attackers**

ML can be attacked in several ways:

- Training data poisoning
- Classification evasion with adversarial example
- Privacy loss: membership inference attack
- Intellectual property loss: surrogate models
- . . . and more

$\Rightarrow$ Teddy Furon will get more into details this afternoon

CentraleSupélec

# Conclusion and how to get started

CentraleSupélec

# Conclusion

## Take-aways

- ML didn't revolutionize security (yet) but can still help you
- Check the learning settings to identify whether and how ML could help you
- ML needs good quality data: search for a dataset or expect running experiments
- Take the time to design your experiment and inspect your data
- Try different learning algorithms starting from the most simple. Sometimes the simplest is to not use ML!
- Understanding how the models work will help you understand when and why they fail
- Overfitting is a risk you need to know how to identify and tackle
- Machine learning requires tries and errors, so don't lose hope :)

CentraleSupélec

# How to get started?

## Data mining and ML software

- Orange (open source)
- Weka (open source)

## Best languages for ML

**python**, R, Matlab

## Best Python libraries for ML

- General ML: scikit-learn
- Deep learning: Theano, Tensorflow, Pytorch, Keras...
- Also: Pandas (data manipulation), matplotlib (visualization)

$\Rightarrow$ We will continue with some practical exercises after the coffee break!