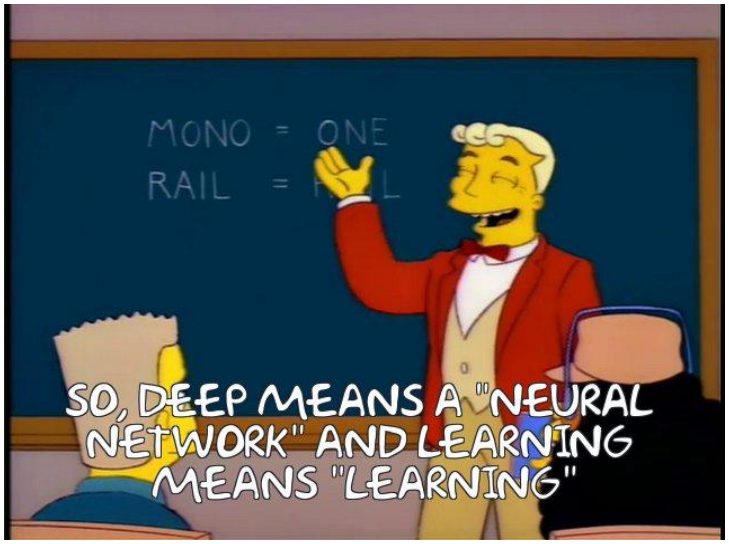
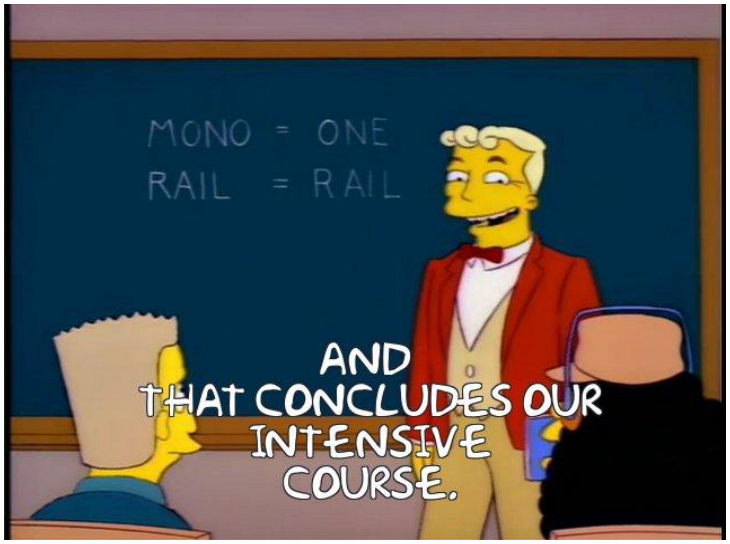
Machine Learning and MC Generators

Manuel Szewc
25/06/2025


MONO = ONE
SO, DEEP MEANS A "NEURAL NETWORK" AND LEARNING MEANS "LEARNING"


MONO = ONE
RAIL = RAIL
AND THAT CONCLUDES OUR INTENSIVE COURSE.

Plan of the lecture

The lecture contains a lot of material, which is also meant to be there for you later.

- Introduction to Machine Learning
- Examples of applications to MC Generators
  - Phase space sampling
  - ME computation
  - PDFs
  - Hadronization
  - Detector simulation
  - Tuning
  - Inference

Intro to ML

What is Machine Learning?

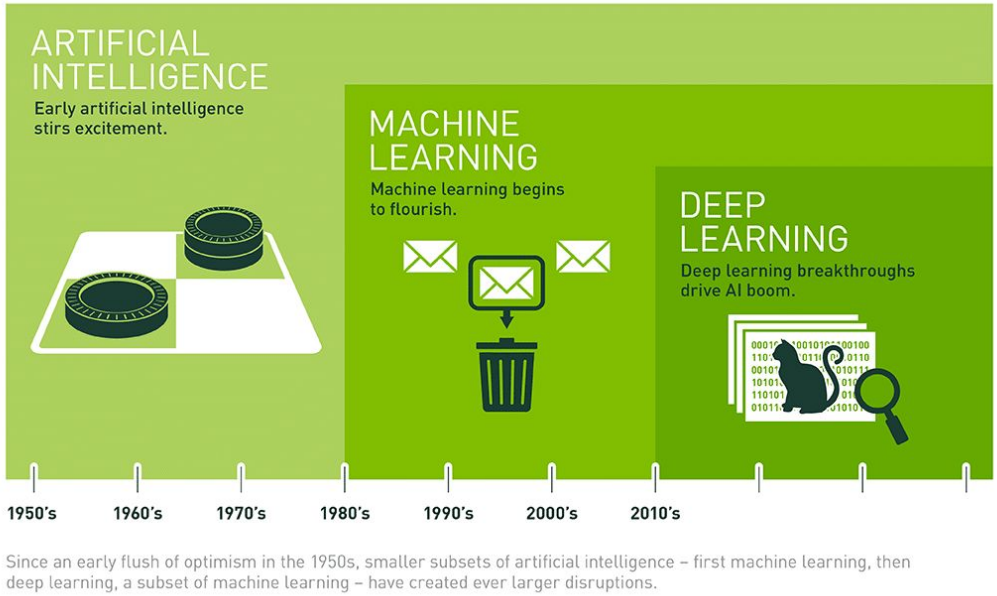


Image from
https://blogs.nvidia.com/blog/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

What is Machine Learning?

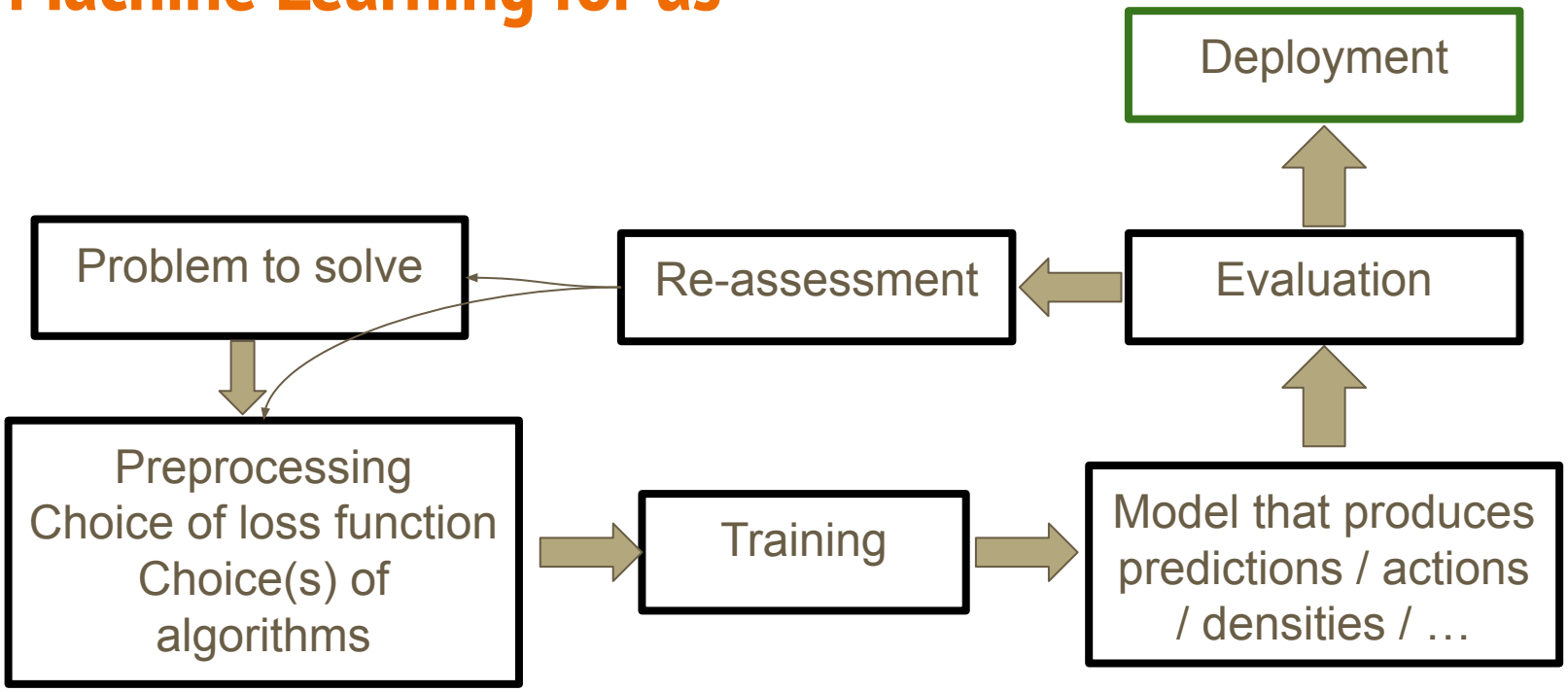
Machine learning is a fairly broad term, but we can think of it as

Getting a computer program to solve a specific task from data without explicit instructions.

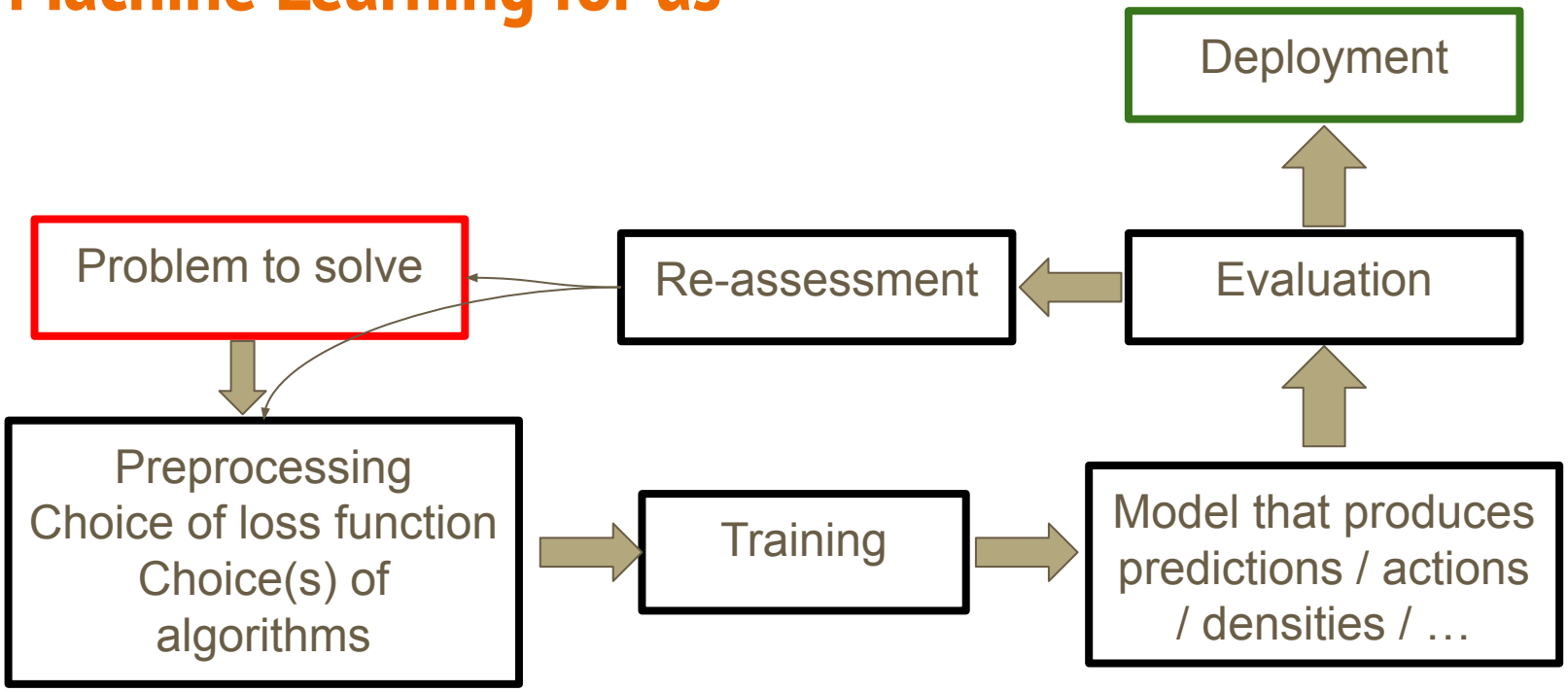
By this definition, even a simple least squares fit for a linear model is machine learning. A lot of Machine Learning is applying **probability** and **statistics.**

There is a vast amount of very nice books, reviews etc. Hard to keep up to date though. I greatly enjoy Aurelien Geron's book as an introduction, it's a classic that keeps being updated.

Machine Learning for us



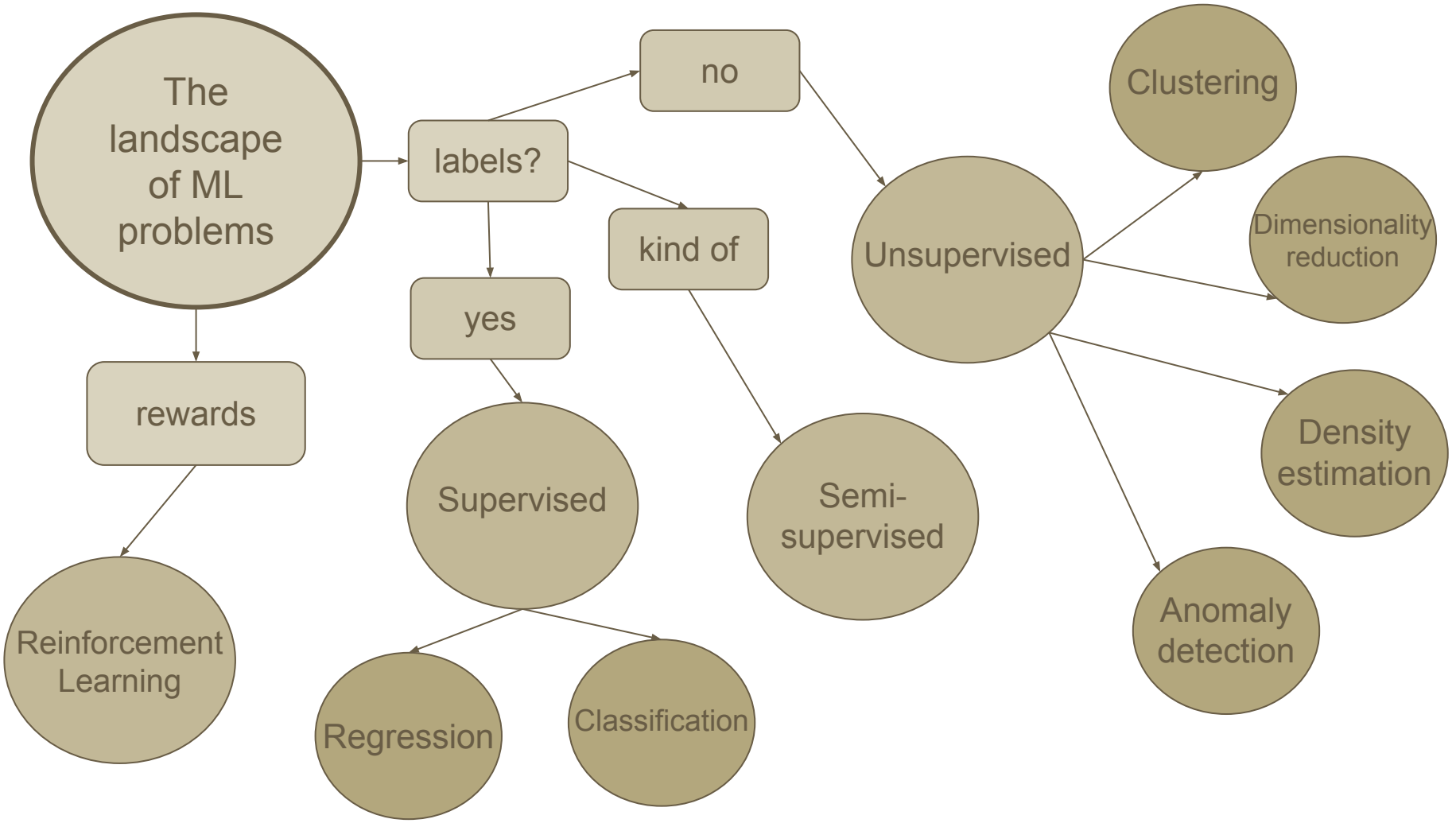
Machine Learning for us

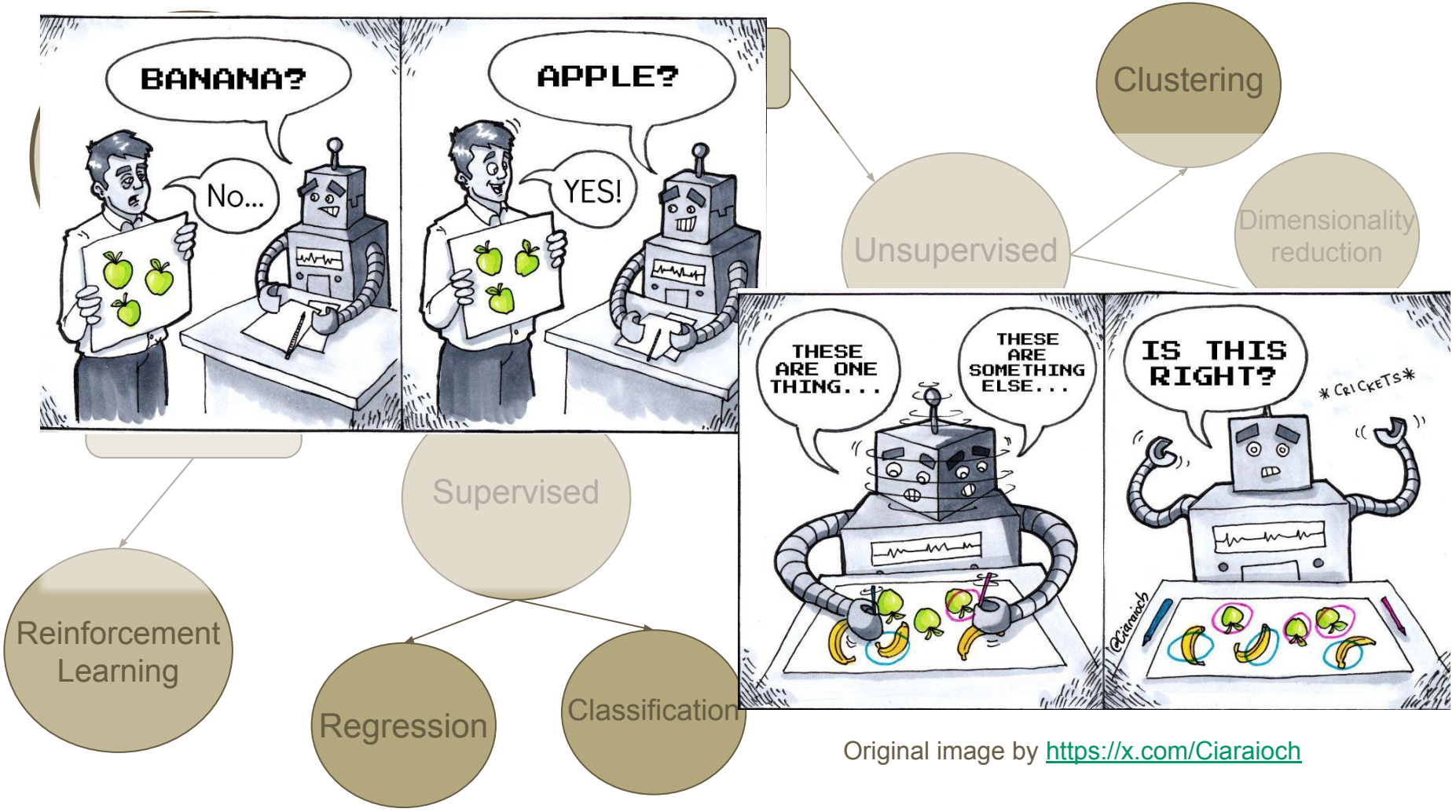


The landscape of ML problems

Things you need to decide:

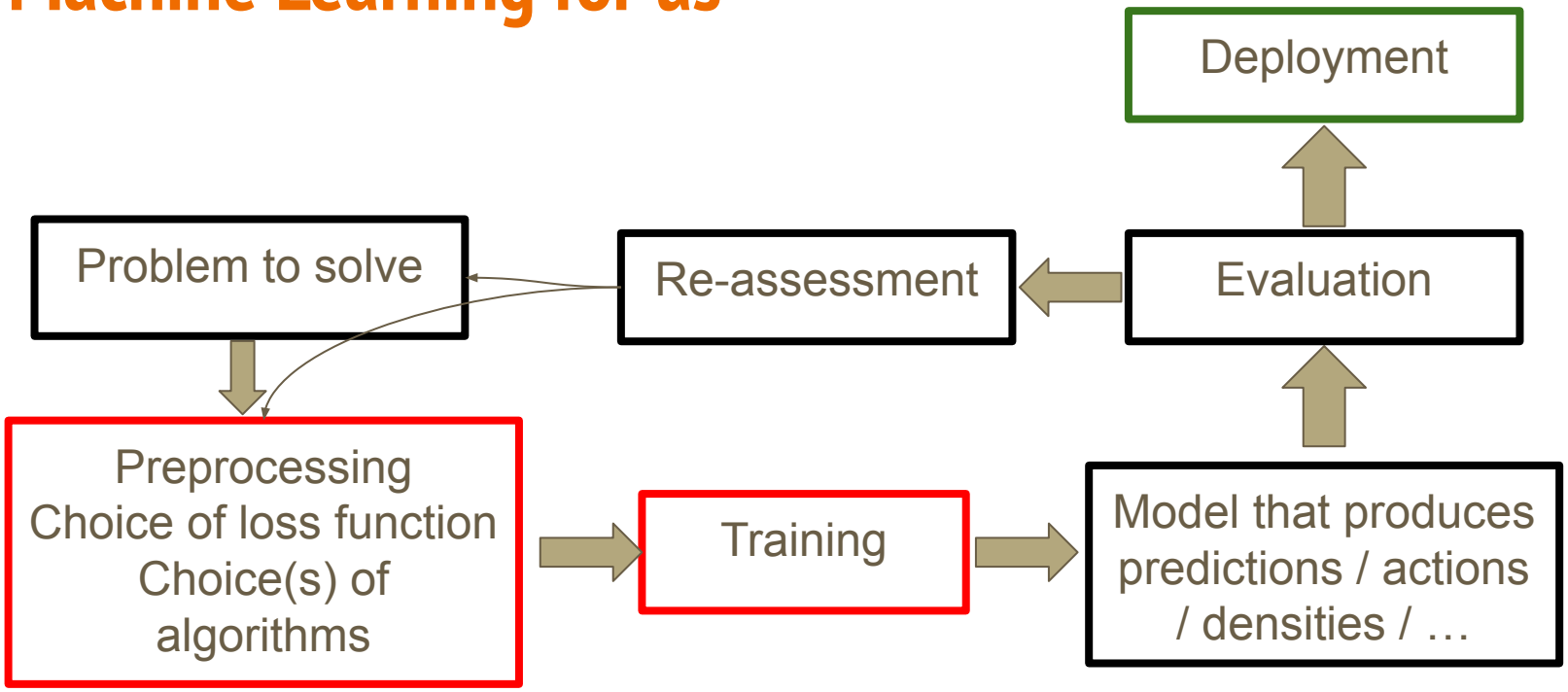
- How much information do you have? **Supervised, unsupervised, semi-supervised, Reinforcement learning**
- Do you want to train once and be done with it (**offline learning**) or be able to update your model with new data **(online learning**)
- **Non-parametric** vs **parametric** models.


The landscape of ML problems
rewards
Reinforcement Learning
labels?
no
yes
kind of
Supervised
Regression
Classification
Semi-supervised
Unsupervised
Clustering
Dimensionality reduction
Density estimation
Anomaly detection



Original image by https://x.com/Ciaraioch

Machine Learning for us



Machine Learning and statistics

$$p(\theta|X,y) = \frac{p(X,y|\theta)p(\theta)}{p(X,y)}$$

Our knowledge is usually encoded in a likelihood function. We want to model the data with parameters θ (over which we can define a prior distribution).

We can obtain likelihood-driven estimates of the parameters (and sometimes associated uncertainties!) or a distribution over parameter space through the posterior

Not all models work this way, most notably tree-based models. But also using non-probabilistic **losses** can be useful!

Loss (or cost) functions for point estimates

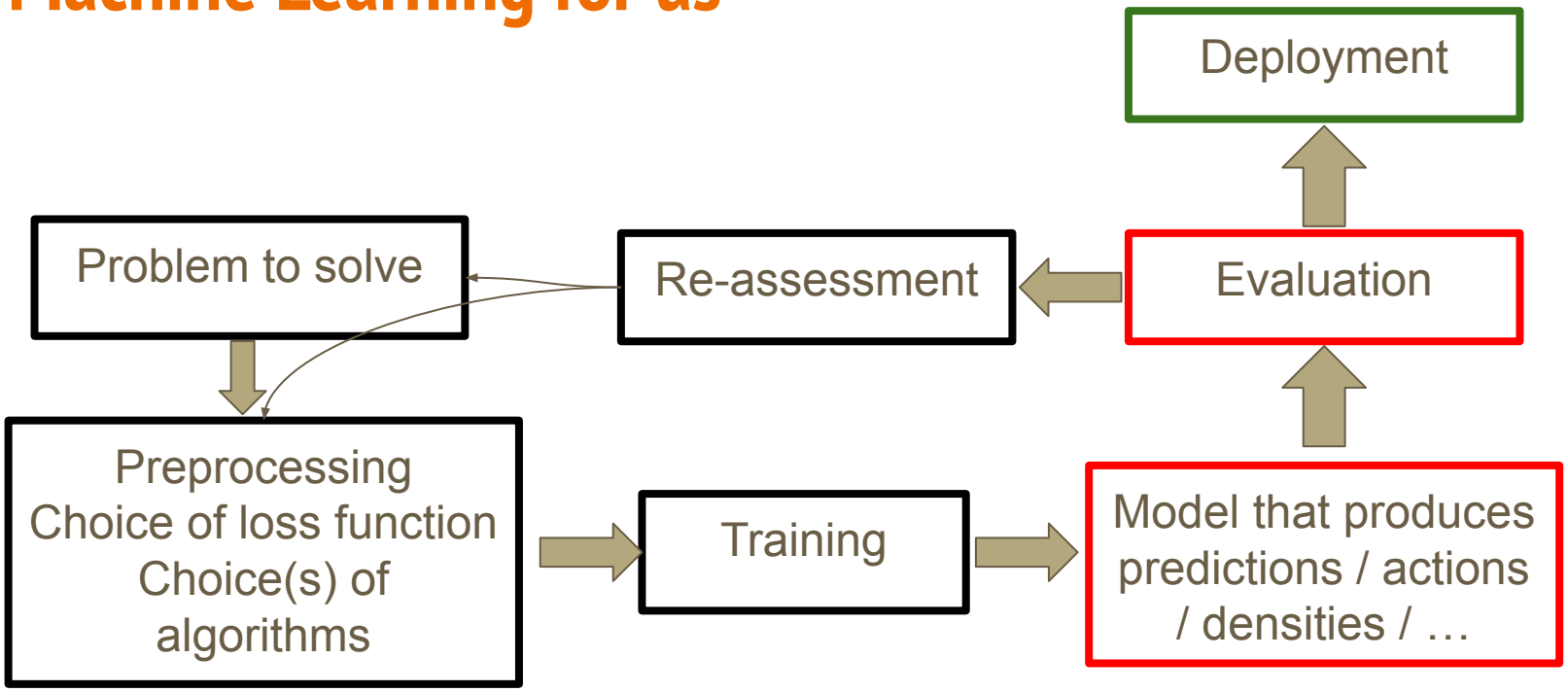
$$L(\theta) = -\ln \mathcal{L}(\theta) = -\prod_{n=1}^{N} \ln p(y_n|x_n, \theta)$$

To find best estimates of our parameters, we need **to decide what's best**.

A key part of the problem is defining a loss function that we can use. We usually like to minimize, and a natural choice (but not the only one!) is the negative log likelihood.

Function minimization is an art of its own, and we rely on a lot of algorithms to do so (e.g. Stochastic Gradient Descent, ADAM, etc. for Neural Networks)

Machine Learning for us



Metrics

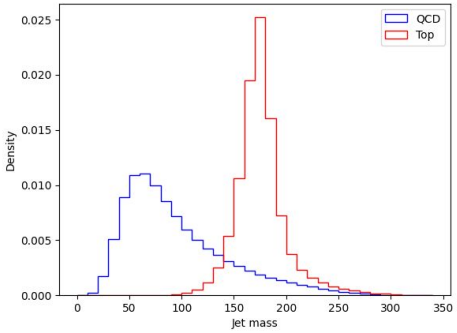
To evaluate, we define what we're interested in. Examples: fidelity of target reproduction, number of games won, etc.

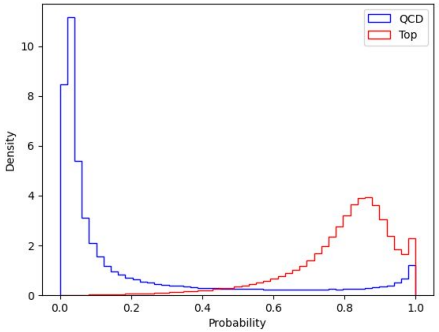
Metrics should be evaluated over **unseen** data (this is usually called a validation set)

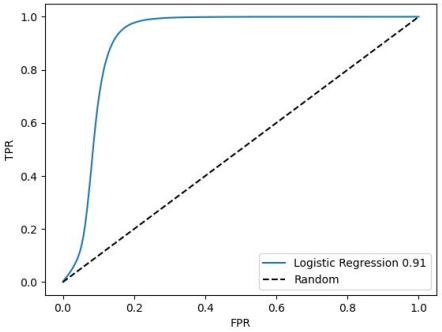
Model selection can be performed through these metrics. To take more advantage of the data we can use **K-Fold cross-validation**.

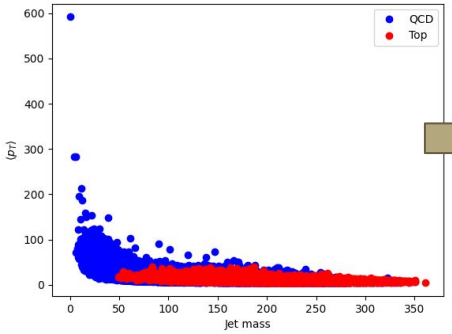
An example of ML-based tagger as mentioned by Reinhard

Example: Top Tagger









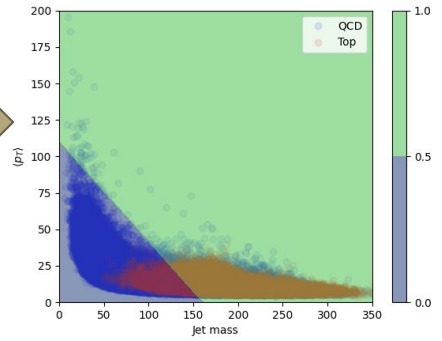

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

```
X_train = np.vstack([total_mass,av_pT]).T
print(X_train.shape)
lr.fit(X_train,train_labels)
```

```
(1211000, 2)
```

```
▾  LogisticRegression
LogisticRegression()
```

```
proba_lr = lr.predict_proba(X_train)[:,1]
assignment_lr = lr.predict(X_train) # it's just 0 if proba < 0.5 and 1 otherwise
```



Regression

We want to predict a continuous target $y_n \in \mathbb{R}$

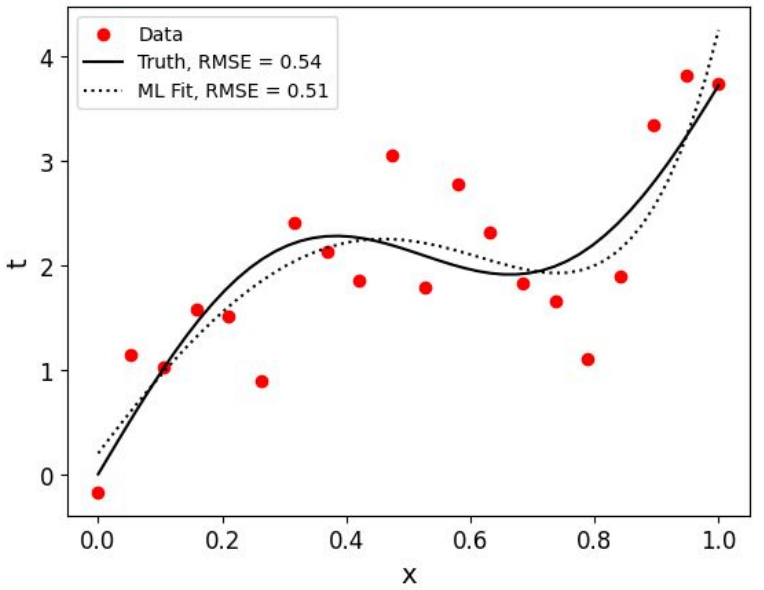
Suppose we have a model, then we describe the target as a random variable depending on the model and associated random noise

$$y_n = f(x_n; \theta) + \epsilon_n$$

If we assume i.i.d

$$p(Y|\theta, X) = \prod_{n=1}^{N} \mathcal{N}(y_n; f(x_n; \theta), \epsilon_n)$$

(usually we also assume homoscedasticity, but not necessary)



A loss function for regression

A gaussian likelihood gives us the usual Mean Squared Error loss

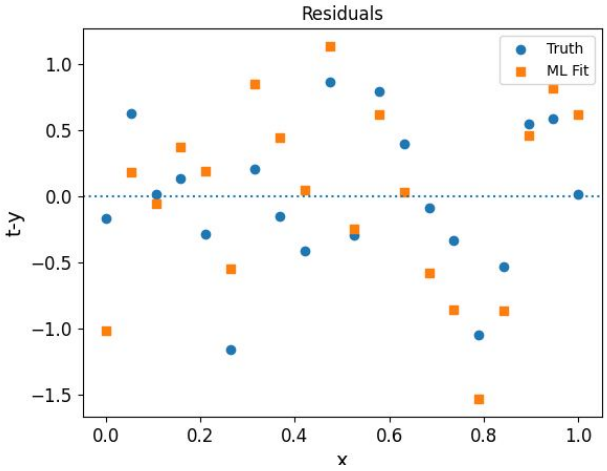
$$\ln \mathcal{L}(\theta) = \sum_{n=1}^{N} -\frac{(y_n - f(x_n, \theta))^2}{2\sigma^2} + \text{terms independent of } \theta$$

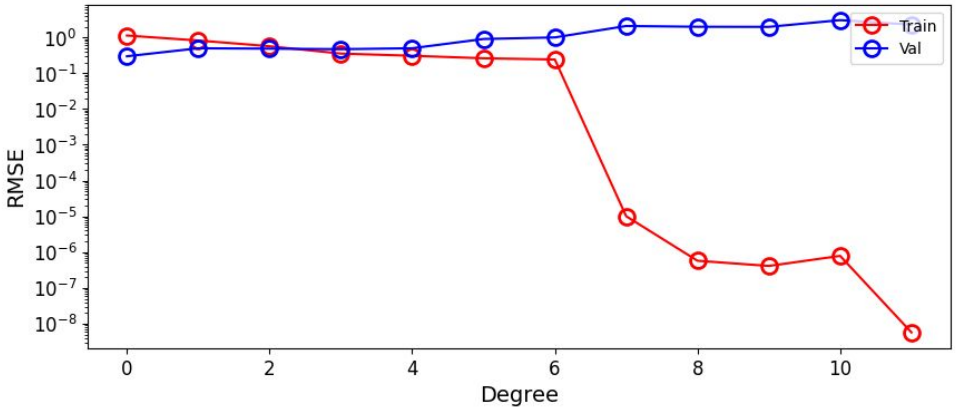
Other choices are possible (MAE, hinge, etc) + regularizations which can be prior-motivated (LASSO, Ridge). MSE is usually a good start due to nice properties both for minimization and for the properties of the obtained estimators (UNOVA)

Metrics

Possible metrics for regression are very similar to the possible losses: the (root) MSE, the MAE, arbitrary exponents.

We also can explore the residuals, which should be normally distributed





Some regression models

Linear Regression (+ with arbitrary basis functions)

Support Vector Machines

Tree-based models

Perceptron

Neural Networks

Classification

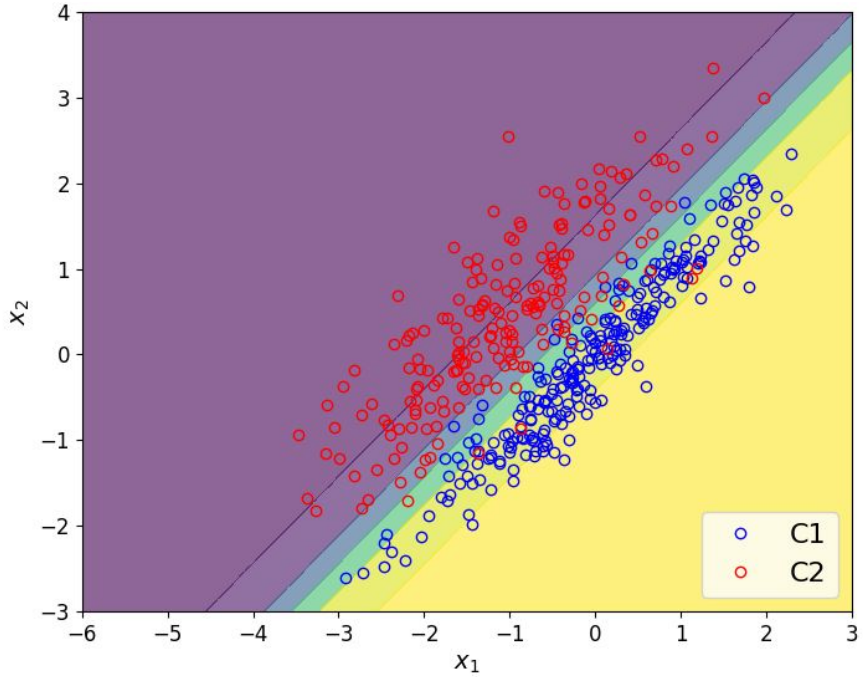
My labels are discrete. For the binary case $y_n \in \{0, 1\}$

If again we assume i.i.d. data, we can think of everything in terms of coin tosses

$$p(Y|\theta, X) = \prod_{n=1}^{N} \text{Bernoulli}(y_n; f(x_n, \theta))$$

f usually converges to a monotonic function of the likelihood ratio, useful for statistical inference (Neyman-Pearson lemma)

$$f(x, \theta) \rightarrow \frac{p(x|\theta, y = 1)}{p(x|\theta, y = 1) + p(x|\theta, y = 0)}$$



The loss function for Classification

The bernoulli likelihood leads to the **Binary Cross-Entropy** loss function, which is the usual choice (but not unique) for classification

$$\ln \mathcal{L}(\theta) = \sum_{n=1}^{N} (t_n \ln y_n(x_n, \theta) + (1 - t_n) \ln(1 - y_n(x_n, \theta)))$$

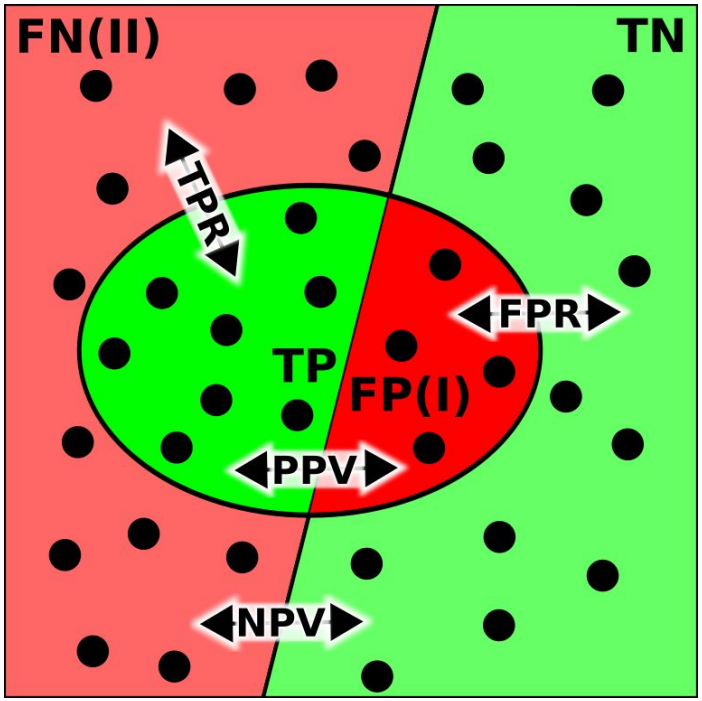
You can check that for t = 1, y should be close to 1 and for t = 0, y should be close to 0.

Metrics

The BCE, although useful, is not so easily interpretable (but it should be monitored!)

Let's say we assign a class to each feature based on a cut (default one is usually $p_1 > 0.5$), we have correctly and incorrectly classified instances.

(image from https://en.wikipedia.org/wiki/File:Binary-classification-labeled.svg)



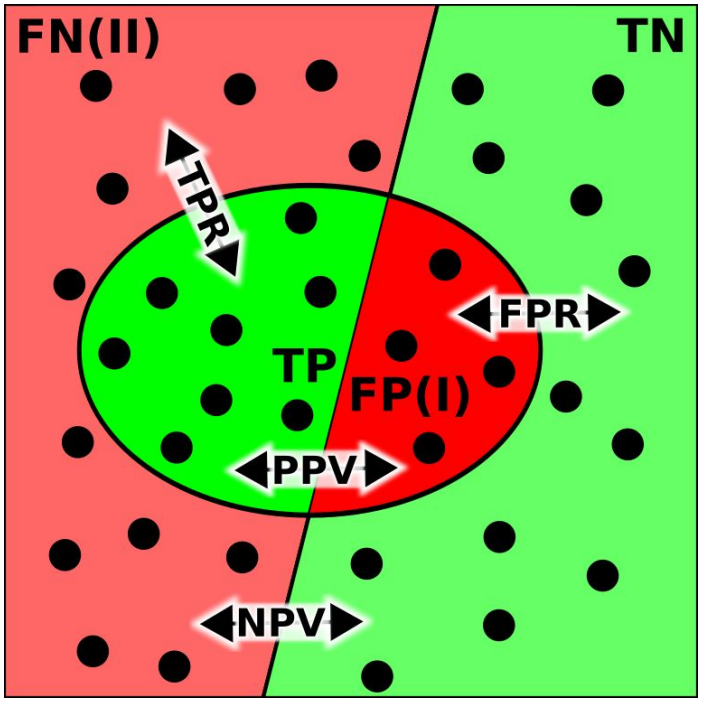
Metrics

From these, we build some nice metrics:

Accuracy = (TP + TN)/(TP+TN+FP+FN)

Precision = TP / (TP+FP)

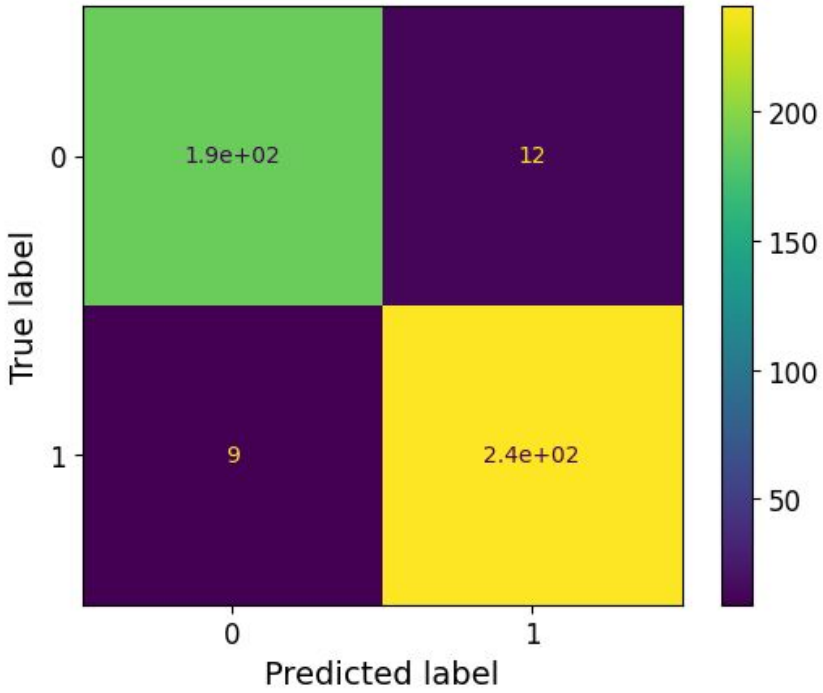
Recall = TP / (TP+FN)



Metrics

From these, we build some nice metrics, encapsulated in **the confusion matrix**

Rows are true labels, columns predicted labels.

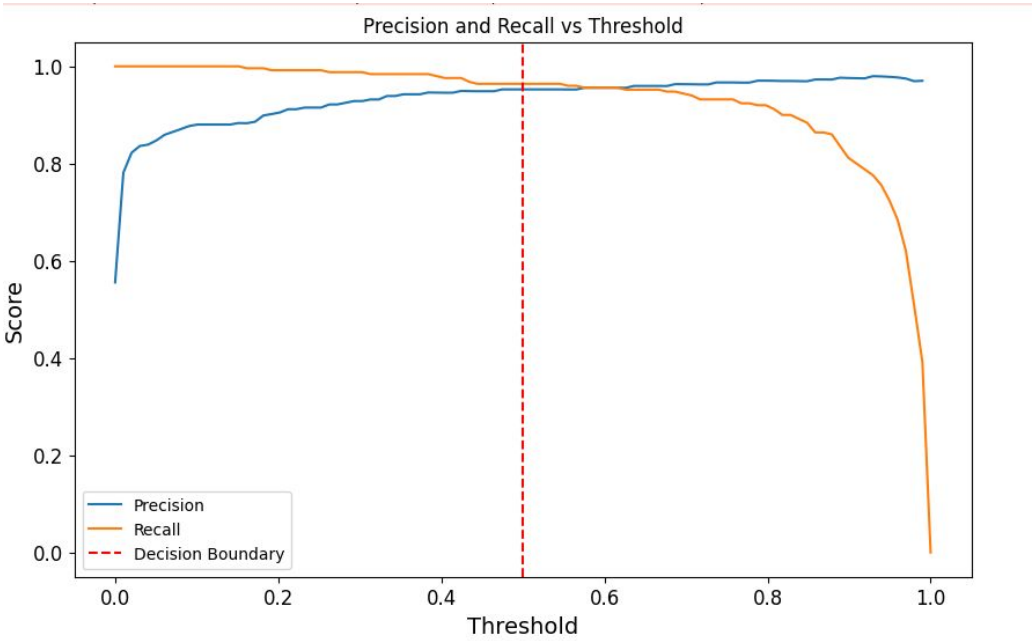


Metrics as a function of selection criteria

For a non-perfect classifier, there is a trade-off between metrics.

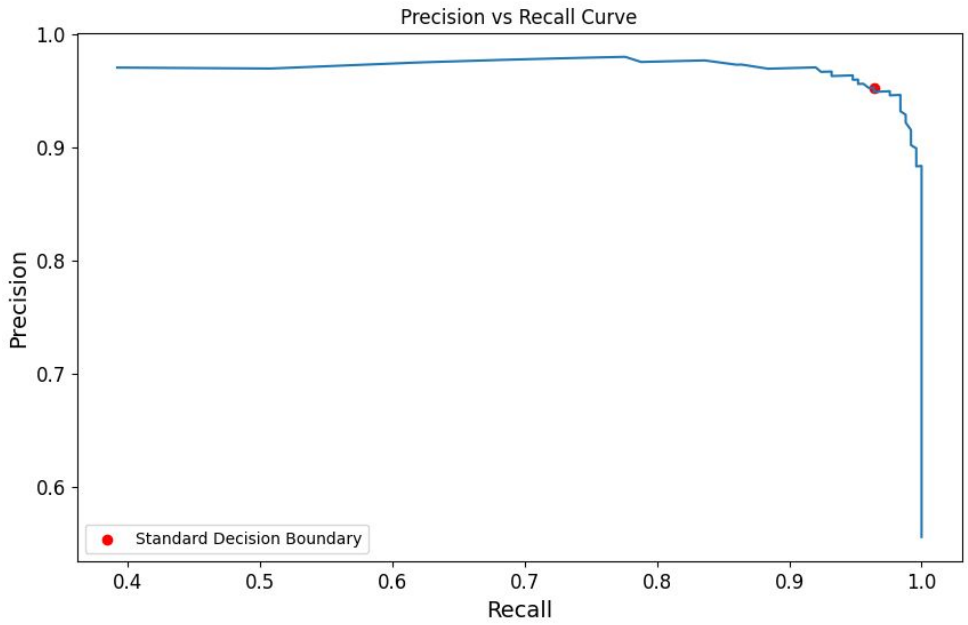
The usual $p_1 > 0.5$ maximizes accuracy (reduces errors), which is good!

But we may prefer other compromises depending on what the problem (think of a spam filter for example)



Precision-recall curve

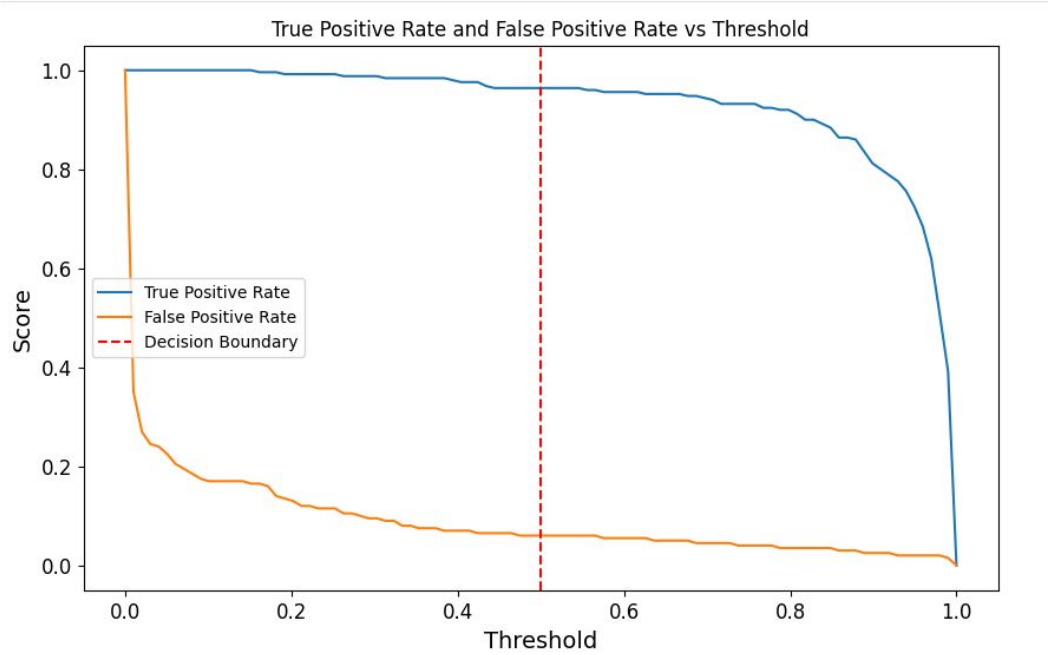
This behavior is usually shown in term of metric vs metric curves, with the score threshold implicit in the cut.



ROC curve

A very common metric is the **Receiver Operating Curve**, which compares how the TPR (=TP/(TP+FN)) and the FPR (=FP/(FP+TN))

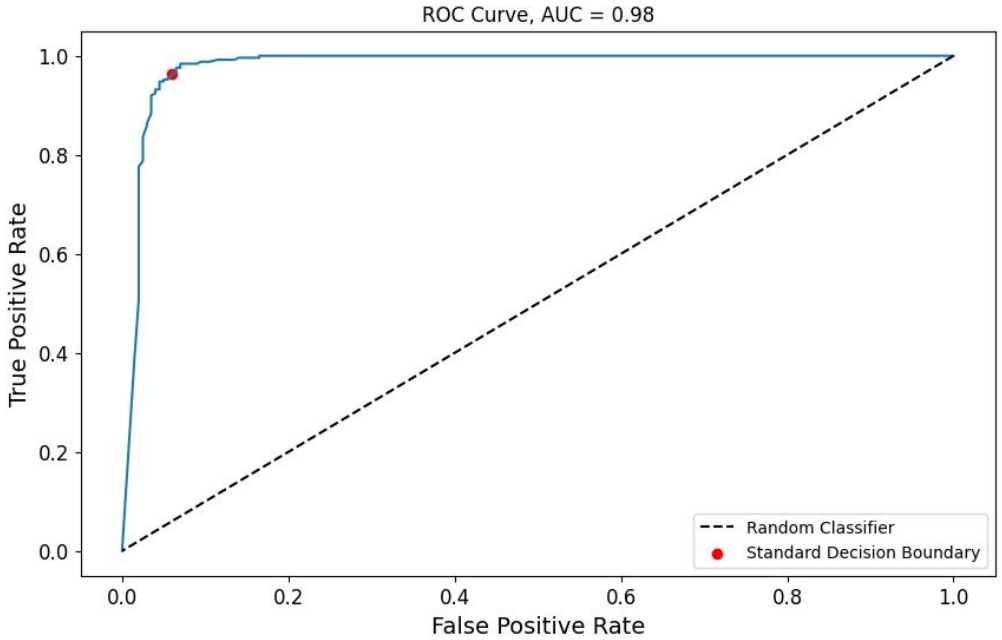
It captures the compromise between recall and leakage



Precision-recall and ROC curves

This is how it's usually shown. The quantitative metric is the **area under the curve** (AUC). The closer to 1.0, the better (and the more upper left the plot is).

A random classifier has AUC = 0.5



Some typical classifiers

Logistic Regression

Support Vector Machines

Naive Bayes

Tree-based models

Perceptron

Neural Networks

The landscape of ML losses!

The training loss depends on the problem. The most common are:

Regression:

- Mean Squared error: $(y_{meas}-y_{pred})^2$
- Mean Absolute error: $|y_{meas}-y_{pred}|$

Classification:

- Binary classes: binary cross entropy $-(y_{meas}Ln(y_{pred})+(1-y_{meas})Ln(1-y_{pred}))$
- K classes : categorical cross-entropy $\Sigma_k\, y_{meas}(k)Ln(y_{pred}(k))$

Usually based on statistical grounds but **not necessarily**

The landscape of ML metrics!

So, evaluating our models is hugely important. Some task-agnostic metrics are

Supervised:

- Regression: Measures of similarity between target and prediction → same as losses!
- Classification: Accuracy, precision, Area Under the Curve, confusion matrix

Unsupervised are usually more task specific but some fairly general are:

- Dimensionality reduction: Explained Variance Ratio for PCA,
- Clustering: the Silhouette coefficient for K-Means
- Density learning: BIC or AIC for generative models

(Some) useful libraries + algorithms

- **scikit-learn**: For most of the “basic” algorithms like Linear Regression and Boosted Decision Trees. Also useful for preprocessing, model combination, etc.
- **XGBoost, LightGBM, CatBoost**: For optimized tree-based models
- **Tensorflow, Pytorch, Jax**: For deep learning models → See tutorial

These are used in academia and industry.

So far...

A big picture talk about Machine Learning, what it is and what types of problem we apply.

Slightly more involved discussion of Regression and Classification.

Now we can do the Regression, Classification and Unsupervised tutorials.

Next...

Here and in the tutorials we've seen "small" models. This is not only pedagogical: **baseline models are vital**. I don't need to overdo things!

But now, let's check the big guns: tree-based models and neural networks.