INFN Machine Learning course

Prof. Amir Farbin, Prof. Daniele Bonacorsi

20-22 May 2019

Camogli, Italy

Intro

Who am I?

I am a **physicist**.

My field is "experimental high-energy particle physics (HEP)" - or subnuclear physics - in particular with particle accelerators.

Research:

- subnuclear physics in OPAL at LEP, in CMS at LHC
- over last >10 years: focus on Software/Computing for the CMS experiment
  - CERN: https://home.cern/
  - CMS: https://cms.cern/

Teaching:

- Subjects: general physics, data analysis, physics laboratory, applied ML, software&computing for HEP
- "Scuole" (i.e. Faculties): Physics, Engineering, Natural Sciences
- Levels: Bachelor, Master, PhD

Contacts

✉ daniele.bonacorsi@unibo.it

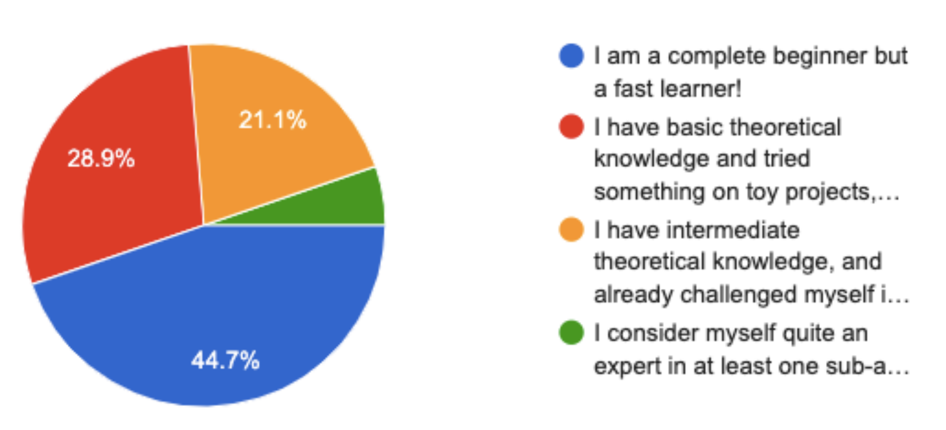
daniele.bonacorsi

@DBonacorsi

Who are **you**?

Highlights from the survey outcome:

- programming: mostly C/C++, ROOT, some Python - much less on all the rest
- ancillary tools: 100% replies expressed “Never used” as most frequent answer
- about 45% complete beginners
- all of you have high expectations though!



Quite heterogeneous, which is stimulating!

- I will try to be as explanatory as possible (I tried in material - slides and plenty of notebooks - I will try while talking)

ML landscape

Definition(s) of Machine Learning

“The capacity of a computer to learn from experience, i.e. to modify its processing on the basis of newly acquired information”

– The Oxford dictionary of statistics terms (today)

“ML is the field of study that gives computers the ability to learn without being explicitly programmed”

– Arthur Samuel (1959), author of the Samuel Checkers-playing Program (and some TeX..)

“A machine is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P** if its performance at tasks in T, as measured by P, improves with experience E.”

– Tom Mitchell (1997)

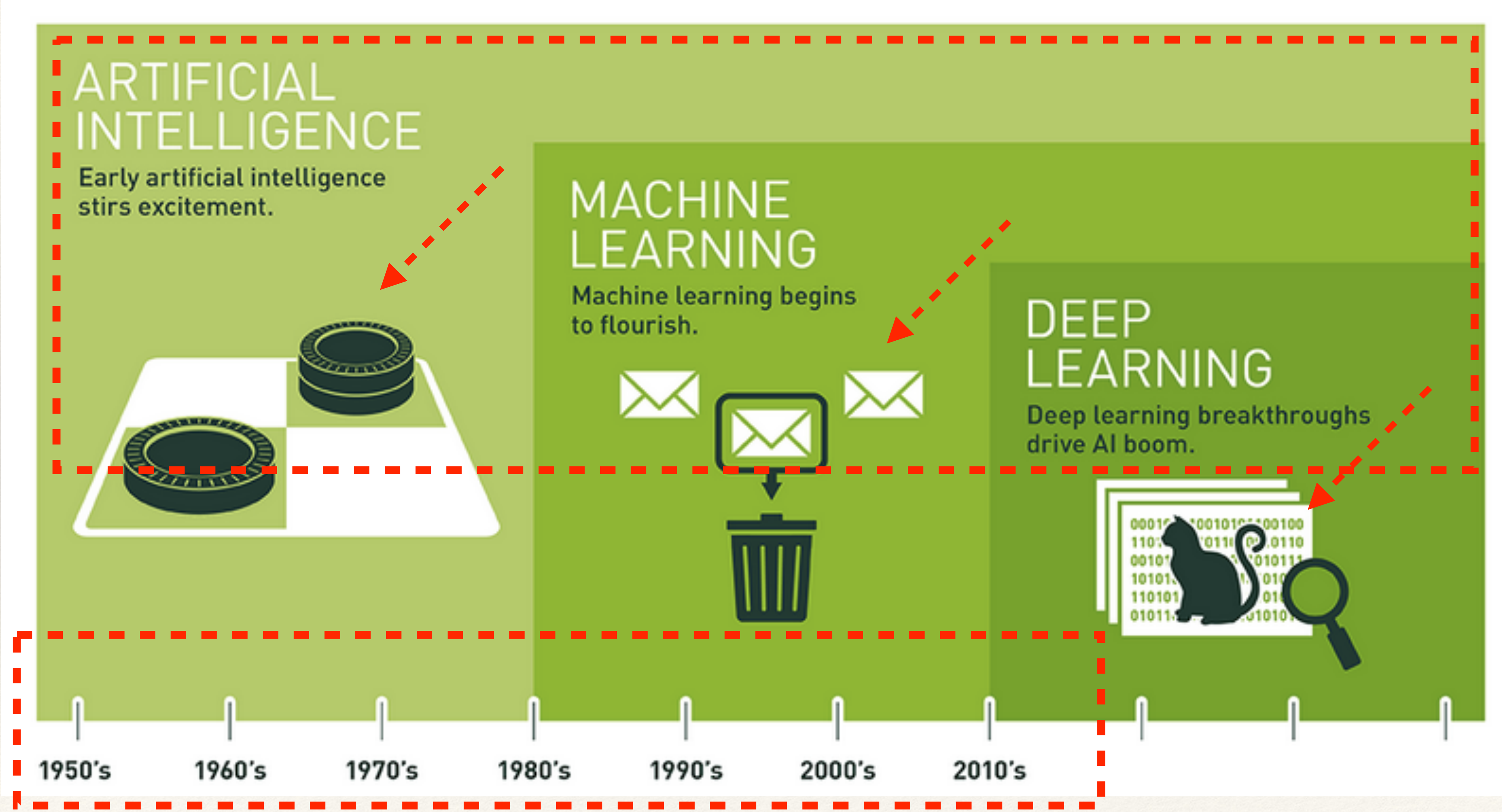
Definition(s) of Machine Learning

"A sub-domain of AI that provides software systems the ability to automatically learn and improve from experience without being explicitly programmed. It relies on an underlying hypothesis about a model one creates, and tries to improve such model by fitting more data into the model over time."

– *one possible elaboration of definitions from various ML practitioners*

Definition(s) of Machine Learning

A pictorial definition by a company (Nvidia)


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Learning algorithms are *everywhere* (HEP is last?)

Not exhaustive list of examples

- **spam** mail filters
- **web search engines** (ranking possible because of a learning algo)
- **clickstream data** collection and consumers' profiling via ML
- **self-customizing programs** (Amazon, Spotify, etc), ML-based recommendation systems, as there is no way to write different programs for million of users
- **images/patterns recognition**. Handwriting recognition. Natural language processing (NLP). Computer vision. All fields of AI pertaining to understanding language or images today is applied ML.
- **Image/patterns recognition**, e.g. medical applications, photo tagging, autonomous driving, ..
- many segments of **industry** and **engineering** (e.g. predictive maintenance)
- **Computational biology,** gene/DNA sequences, ..
- more **science** domains
- …

We will quote and discuss more some of these later on,
and we will also quote more..

“everywhere” → AI vs HI ?

Will AI replace human intelligence (HI)?

- the infamous (and debatably clever) SkyNet quotes on mass media on AI..

No. At least, not in the foreseeable future

- admittedly AI experts think the best approach to implement the ML definition of learning is to try to mimic how the human brain learns, but that’s a different point
- *Example*: where is intelligence (HI) located, at the best of today’s knowledge?
- *Example*: a baby learning car shapes over time, is this “intelligence”?

Before continuing, we had better reshape our terminology to assess proper content.

“AI”, really?

“AI” terminology perhaps misleading in most practical discussions

Most of AI research today is actually not trying to recreate intelligence in any shape or form, at all

~~Artificial Intelligence~~ → **Automation (of decisions)**

It is aiming at collecting data around how humans make decisions, to perform the same tasks at a scale and latency that are not humanly possible

- *Example*: facial recognition. The best of us can link thousands of faces to names, without cheating. Machines can do this for hundreds of millions of faces, and in less time. Are such machines smarter than us in a general sense? not even close! But for that specific domain, once trained with large amounts of data (and money..), they perform that task at a speed and scale that is beyond what any human may ever hope for.

This is the “artificial intelligence” we are talking about.

Automation of decision and integration

Areas that may benefit from this "automation of decisions" are areas where such advancement would eventually help humans.

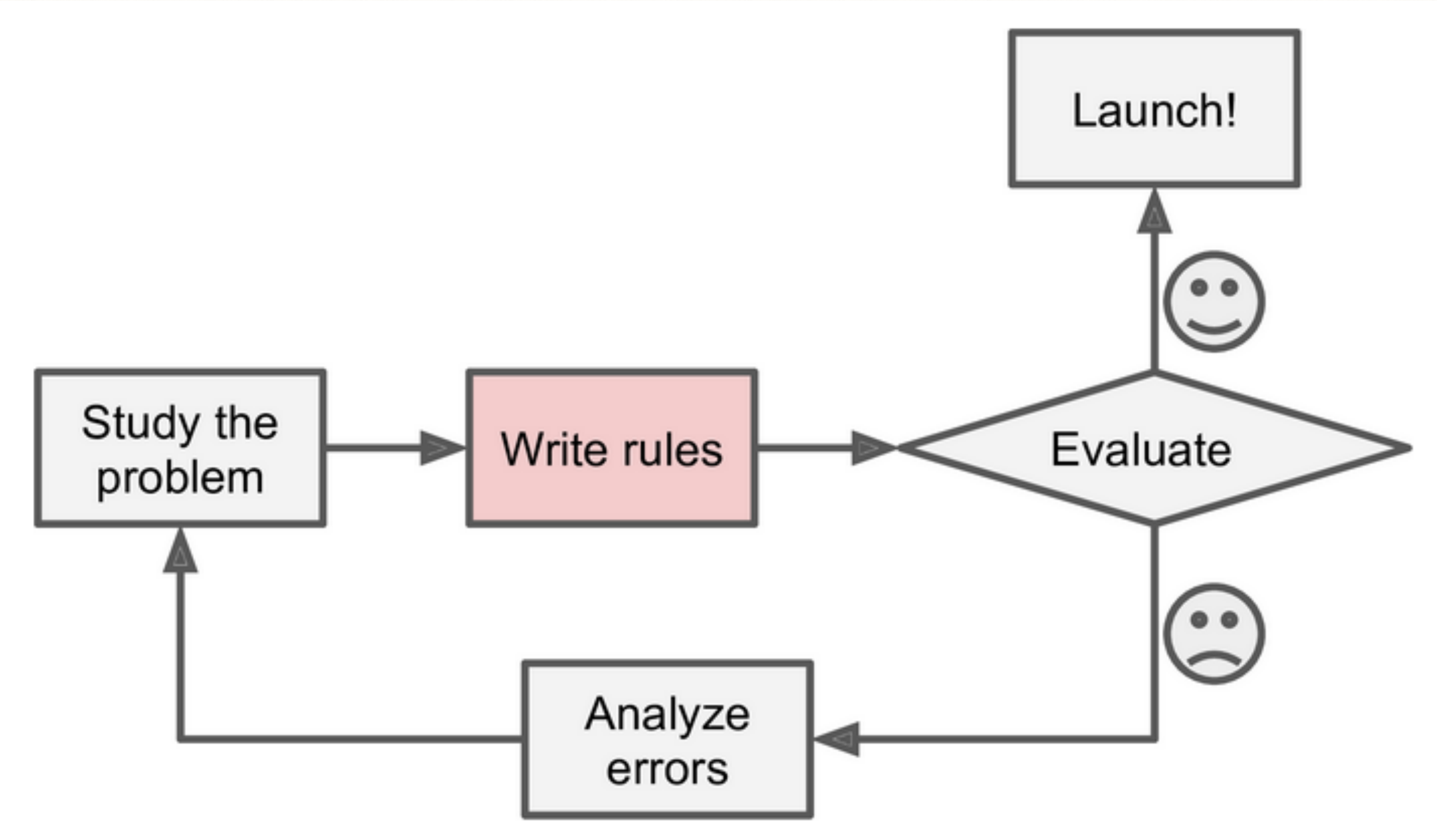
In this scope, it is beneficial to use machines instead of humans in tasks that either humans <u>do not</u> want to do or <u>cannot</u> do as well

- *<u>Example</u>*: AI-based systems able to retain high-performances also in high-stress conditions or environmental dangers unbearable by humans

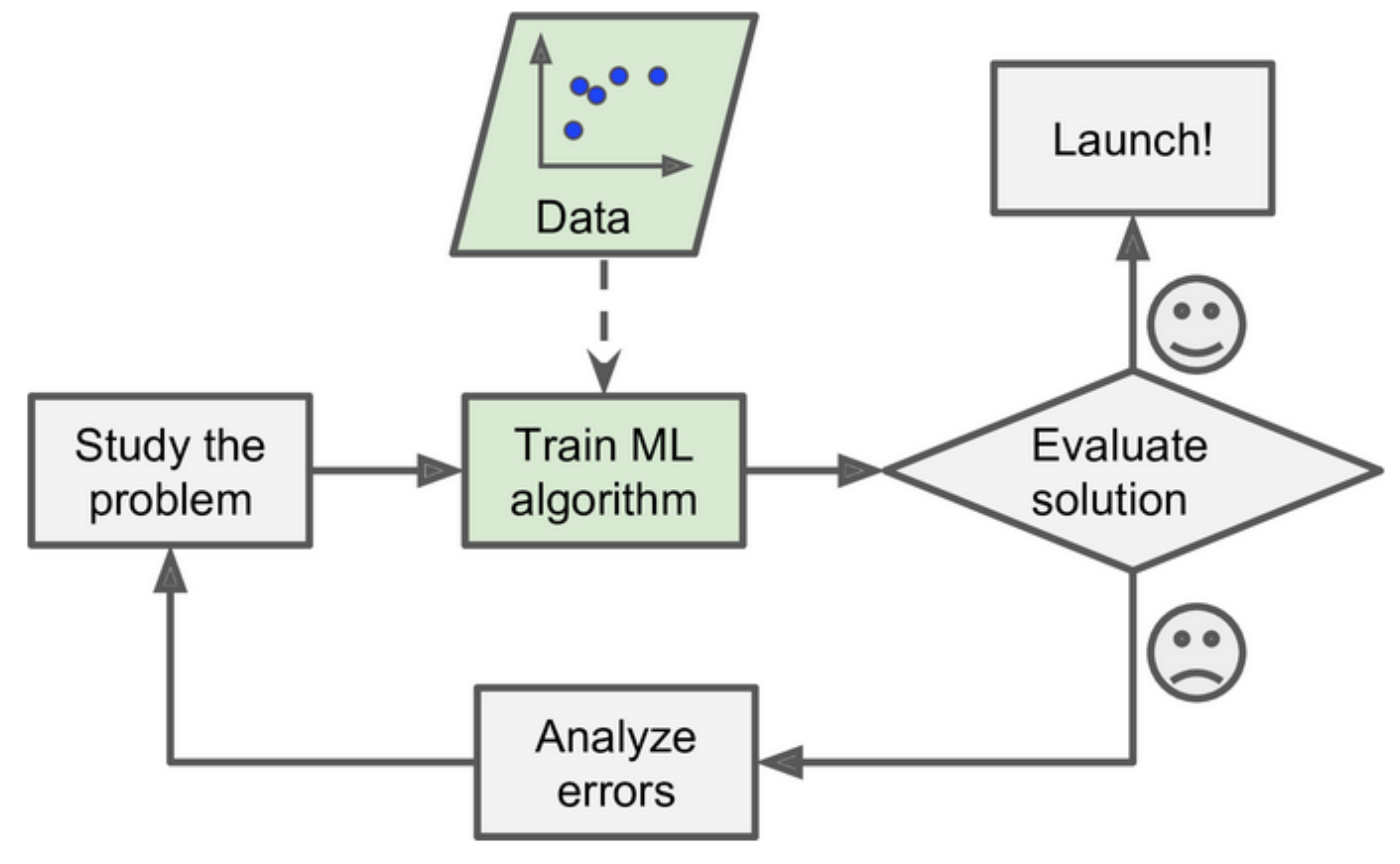
The keyword might be integration. I.e. AI "**together with**" HI, and not "**instead of**".

- *<u>Example</u>*: recent progresses in ML-based radiology, yielding unprecedented and unexpected ethical implications in <u>not</u> using AI..

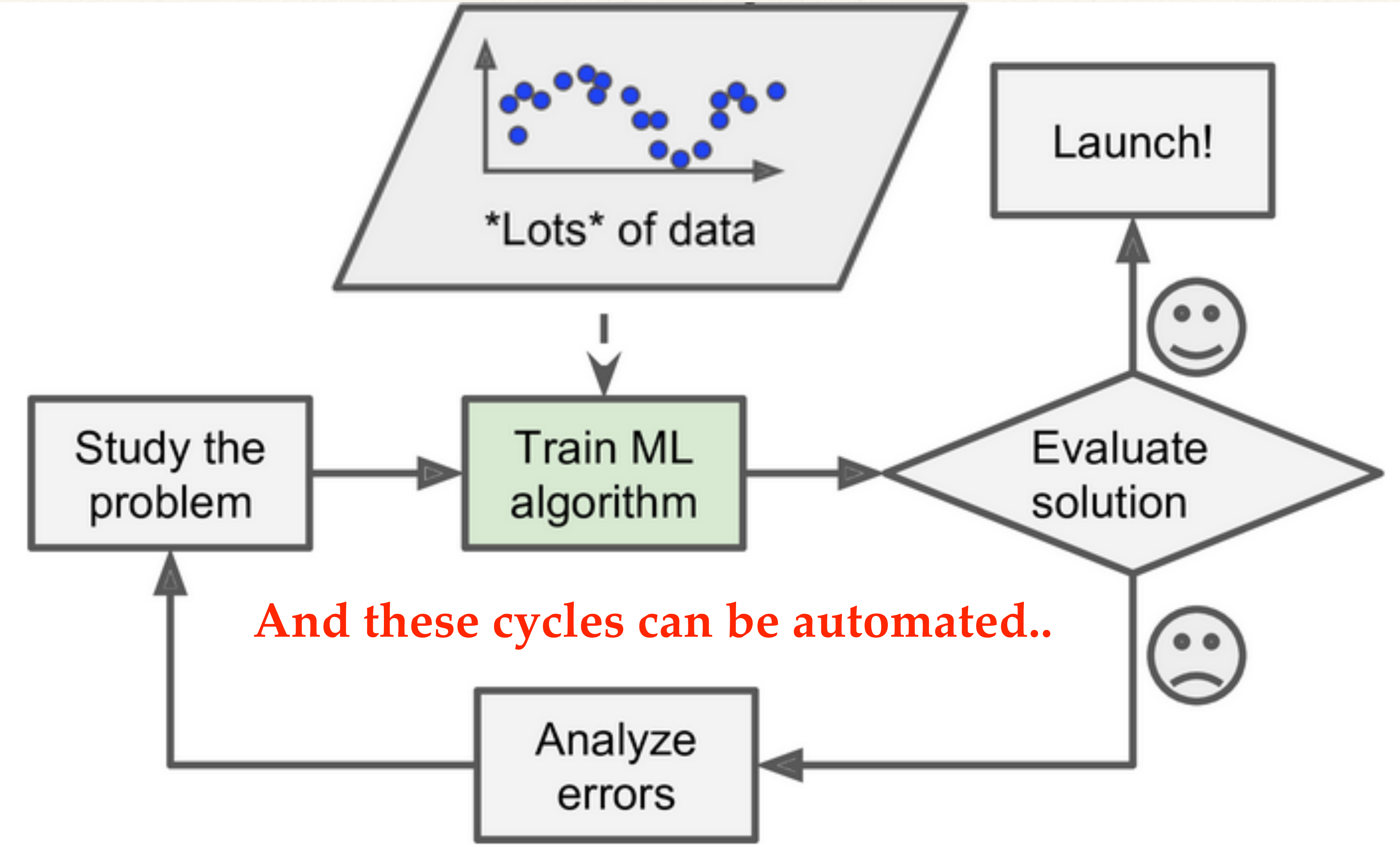
Traditional approach



ML approach: data-driven modelling



ML approach: **data**-driven modelling



And these cycles can be automated..

Types of ML

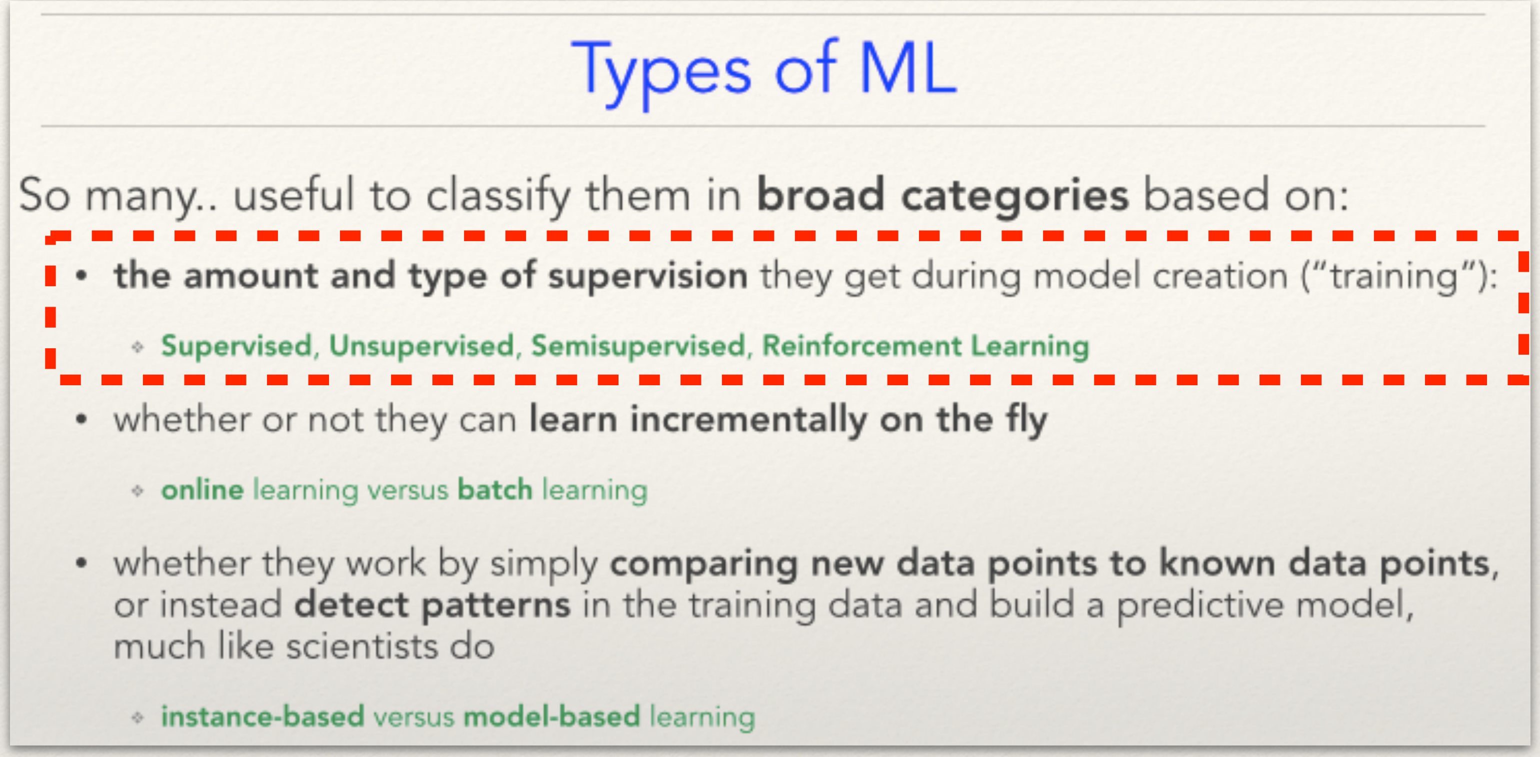
So many.. useful to classify them in **broad categories** based on:

- **the amount and type of supervision** they get during model creation (“training”):
  - **Supervised**, **Unsupervised**, **Semisupervised**, **Reinforcement Learning**
- whether or not they can **learn incrementally on the fly**
  - **online** learning versus **batch** learning
- whether they work by simply **comparing new data points to known data points**, or instead **detect patterns** in the training data and build a predictive model, much like scientists do
  - **instance-based** versus **model-based** learning

These criteria are not exclusive; you can combine them in any way

- e.g. a state-of-the-art spam filter may learn on the fly using a deep neural network model trained using examples of spam and good mails, which makes it an *online, model-based, supervised learning system*

Let’s look at key concepts of each.



Types of ML

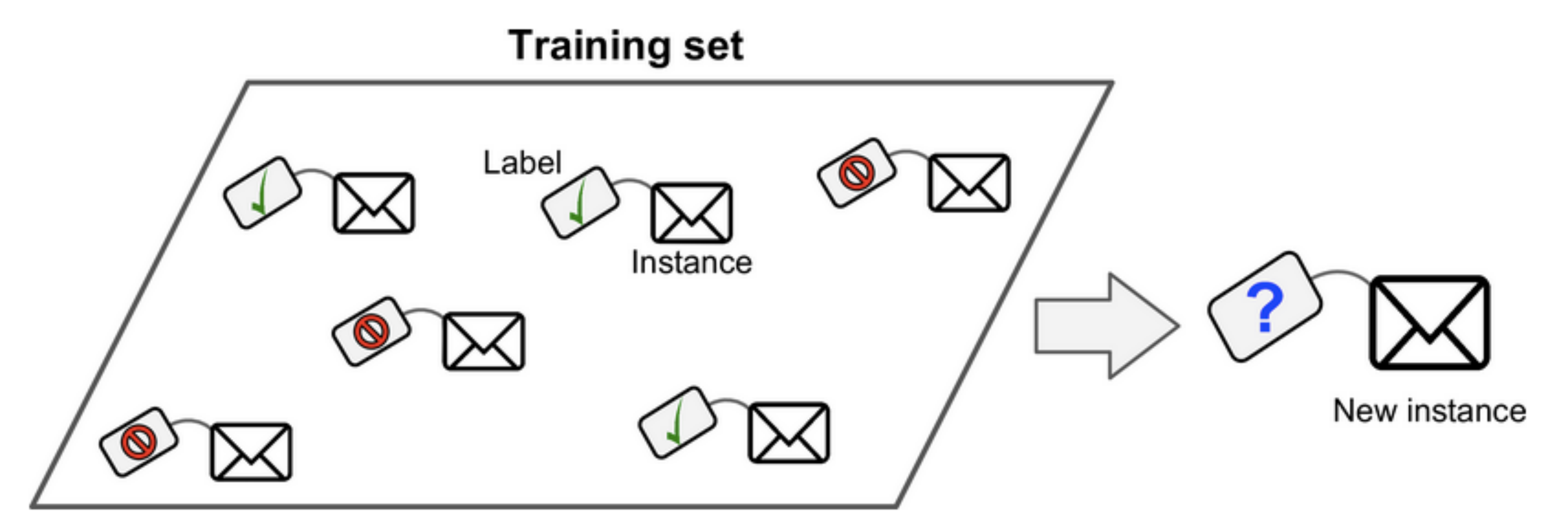
So many.. useful to classify them in **broad categories** based on:

- **the amount and type of supervision** they get during model creation (“training”):
  - **Supervised**, **Unsupervised**, **Semisupervised**, **Reinforcement Learning**
- whether or not they can **learn incrementally on the fly**
  - **online** learning versus **batch** learning
- whether they work by simply **comparing new data points to known data points**, or instead **detect patterns** in the training data and build a predictive model, much like scientists do
  - **instance-based** versus **model-based** learning

Supervised ML

In **supervised learning**, the data entries (“**instances**” or “examples”) you feed to the algo for it to learn (through its “**attributes**” - instantiated in“**features**” - in a process called “**training**”) includes the truth info, i.e. the so-called “**labels**”

- e.g. for a spam detection problem:



Another (one of my favourite) example is a child learning to recognise car vs bus vs bicycle in traffic with a parent guiding her/him

Supervised ML: **Classification** vs **Regression**

A typical supervised learning task is **classification**

- predict “**classes**”: **binary** (0/1, yes/no) or **multi-class** (A/B/C/D)
- *e.g. spam filter: trained with many example emails along with their class labels (“spam” or “not-spam”), it learns how to classify new coming emails*

Another typical supervised learning task is **regression**

- predict “**target numeric values**” (in a continuum of values)
- *e.g. a price of a house, knowing its attributes, and being given plenty of instances of other houses (both their features and price)*

Note: regression algorithms exist that can be used for classification as well, and vice versa

- e.g. Logistic Regression is commonly used for classification, as it can output a value that corresponds to the probability of belonging to a given class (e.g., 20% chance of being spam)

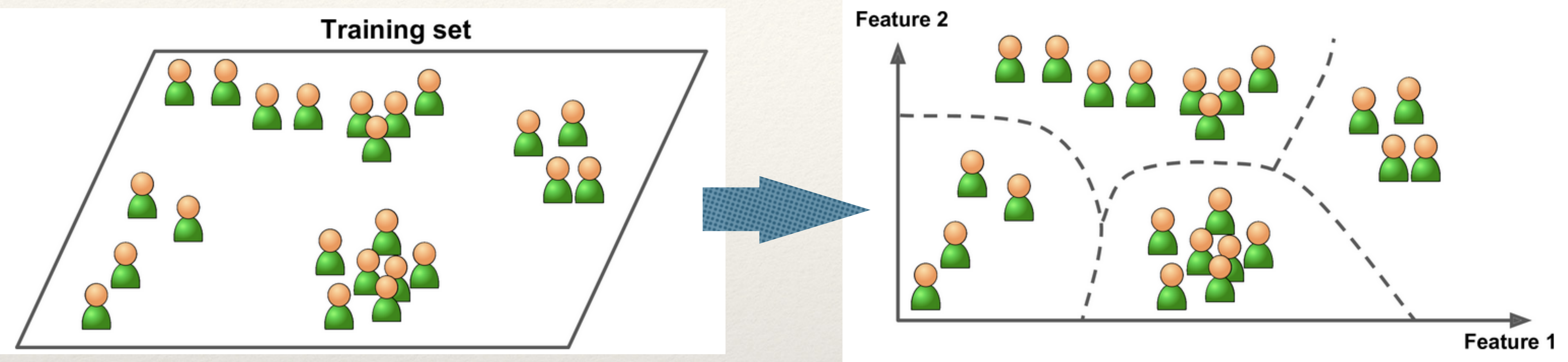
Supervised ML **algos**

Most commonly used / important:

- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- k-Nearest Neighbours
- Decision Trees and Random Forests
- Neural Networks
  - some neural network architectures can be unsupervised, such as autoencoders and restricted Boltzmann machines. They can also be semisupervised, such as in deep belief networks and unsupervised pretraining
- …

Unsupervised ML

In **unsupervised learning** the training data is **unlabeled**, so the system tries to learn without a teacher guiding it



<u>Example</u>: data about blog readers

- run an unsupervised (e.g. clustering) algo to detect "groups of similar visitors"
- at no point you tell the algo which group a visitor belongs to
- but it finds it out: e.g., it might detect that 30% are females who comment on your posts on topic X, and usually read the blog in the evening, etc.
- with unsupervised (hierarchical clustering) algos, you may detect subgroups, etc

Unsupervised ML **algos**

Most commonly used / important:

Clustering → try to detect groups

- K-Means
- DBSCAN
- Hierarchical Cluster Analysis (HCA)
- Anomaly detection and novelty detection
- One-class SVM
- Isolation Forest

Visualisation and **Dimensionality Reduction** → display / simplify data w/o losing too much info

- Principal Component Analysis (PCA)
- Kernel PCA
- Locally-Linear Embedding (LLE)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

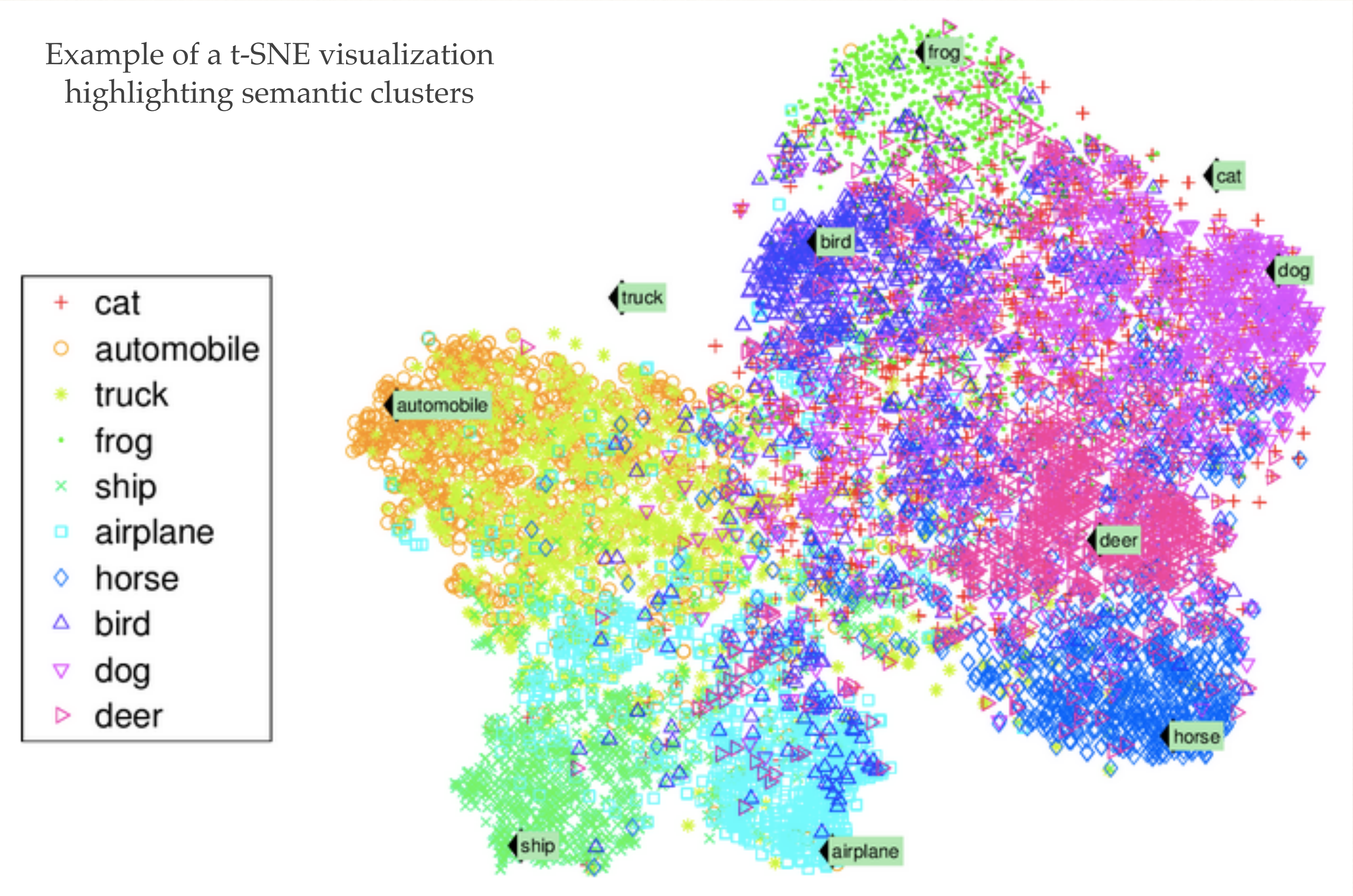
Association rule learning → discover interesting relations between attributes in large datasets

- Apriori
- Eclat

Unsupervised ML: **visualisation**

Data visualisation is also a customer of unsupervised learning algos

- feed visualisation algos with a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted



Example of a t-SNE visualization highlighting semantic clusters

Semi-supervised ML

Semi-supervised learning algos deal with **partially-labeled training data**

- usually: a lot of unlabeled data plus a little bit of labeled data

Example: Google Photos

- **clusters** of photos with same persons (cluster with A and cluster with B)
- then, **labelling** some faces with names help the system to clean-up clusters, kill look-alikes, etc

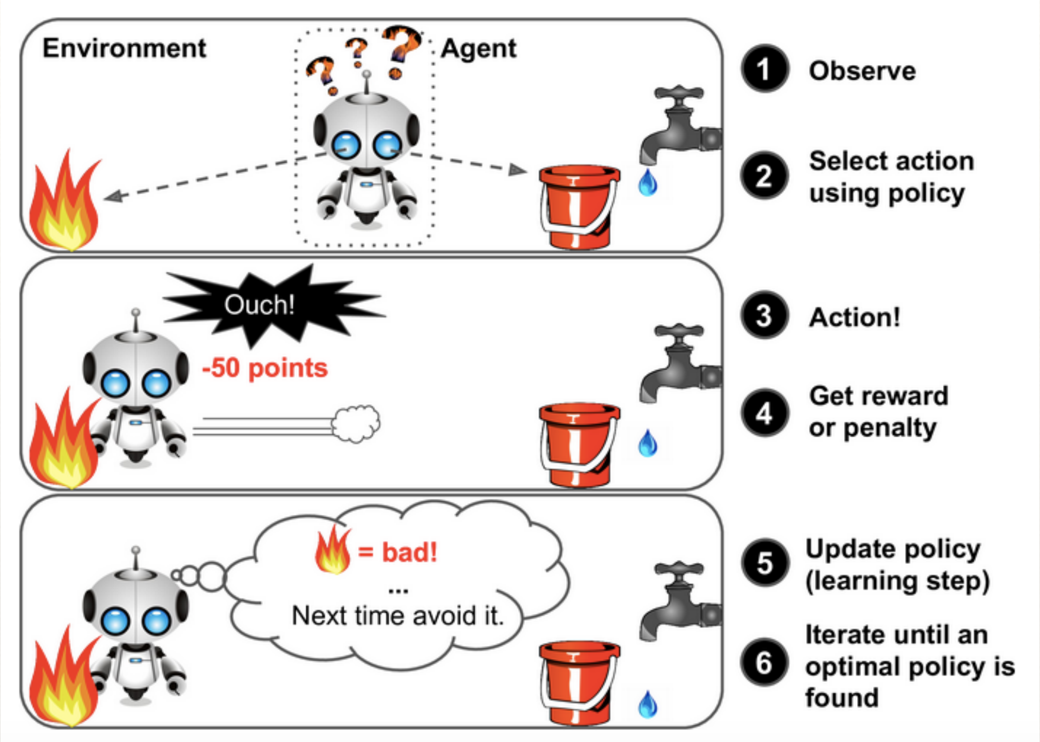
Most semisupervised learning algos are **combinations of unsupervised and supervised algos**

- e.g. **Deep Belief Networks** (**DBNs**) are based on unsupervised components called **Restricted Boltzmann Machines** (**RBMs**) stacked on top of one another. RBMs are trained sequentially in an unsupervised manner, and then the whole system is fine-tuned using supervised learning techniques.

Reinforcement Learning (RL)

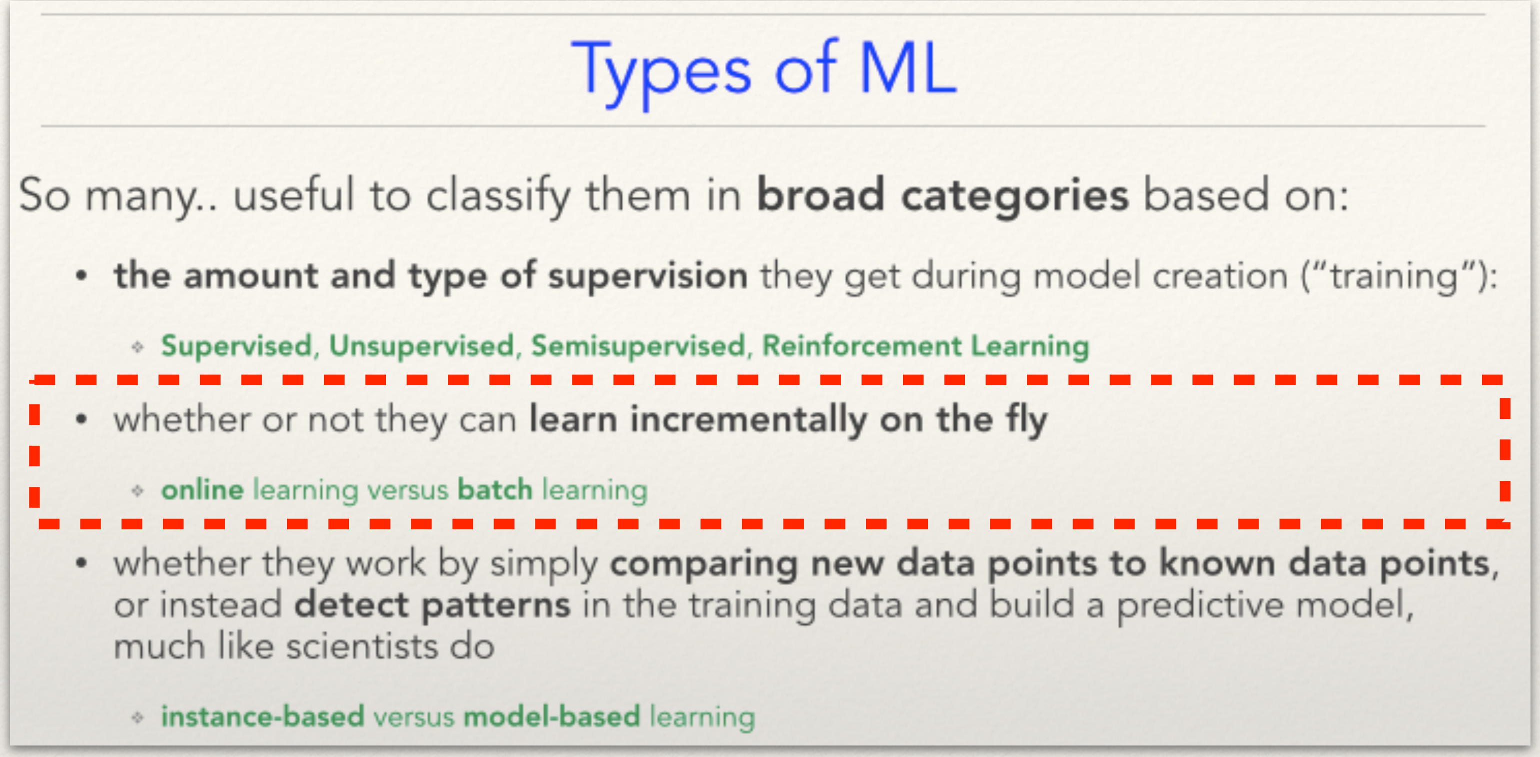
Reinforcement Learning is completely different.

- The learning system is called "**agent**" in this context
- it **observes** the environment
- select and perform **actions**
- get positive **rewards** or negative rewards (i.e. **penalties**) in return
- learning step is define the best **policy** to get the most reward over time


Environment
Agent
Ouch!
-50 points
= bad!
...
Next time avoid it.
1 Observe
2 Select action using policy
3 Action!
4 Get reward or penalty
5 Update policy (learning step)
6 Iterate until an optimal policy is found

<u>Examples</u>:

- robots implement RL to teach themselves learn how to walk
- DeepMind's AlphaGo program beat Ke Jie at the game of Go (May 2017)



Types of ML

So many.. useful to classify them in **broad categories** based on:

- **the amount and type of supervision** they get during model creation ("training"):
  - **Supervised**, **Unsupervised**, **Semisupervised**, **Reinforcement Learning**
- whether or not they can **learn incrementally on the fly**
  - **online** learning versus **batch** learning
- whether they work by simply **comparing new data points to known data points**, or instead **detect patterns** in the training data and build a predictive model, much like scientists do
  - **instance-based** versus **model-based** learning

Batch learning vs **Online** learning

Classification of ML algos based on **whether or not the system can learn incrementally from a stream of incoming data**

Batch learning

In **batch learning**, the system is incapable of learning incrementally

- it must be trained **using all the available data** (i.e. **offline learning**, in contrast with "online")
- **more data?** stop, re-train (on old+new data), create new model, refine it, switch to it and abandon the old one, launch the new in production
    - "new data weekly" mode vs "rapidly changing data" (e.g. aiming to predict stock markets?)
- it can be resource hungry (i.e. time and computing)
    - CPU, memory space, disk space, disk I/O, network I/O, etc.

When do you hit an unsurmountable limit with batch learning?

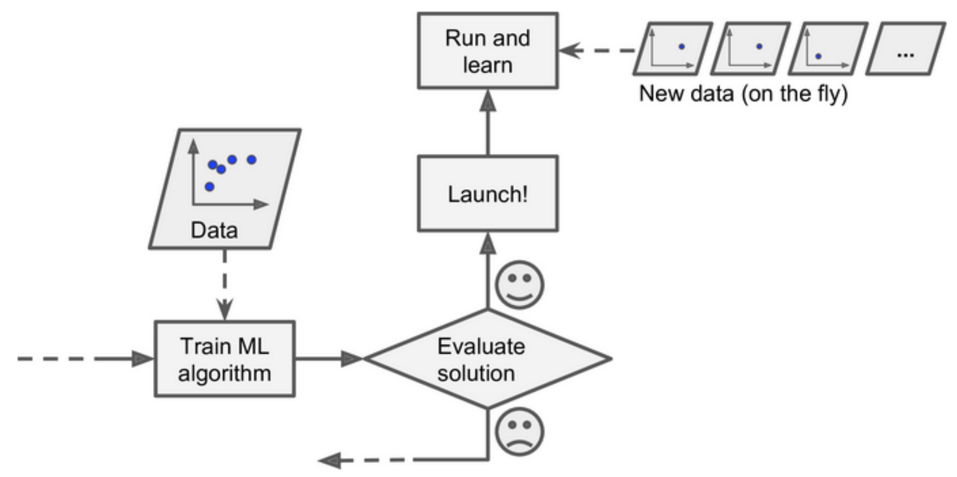
- plenty of data + automation requirements → plenty of money!
- need to learn autonomously (based on large volumes of data) + limited resources (e.g. smartphone app, or rover on Mars..) → even impossible!

Solutions here might come from using different algos, that are capable of learning incrementally..

Online learning

In **online learning**, you **train the system incrementally**

- by feeding it data instances sequentially
  - either individually or by small groups called "mini-batches"
- **each learning step is fast and cheap**, so the system can learn about new data on the fly, as it arrives



Points of strength:

- perfect for systems that receive **data as a continuous flow**
  - and need to adapt to change rapidly or autonomously
- good option if you have **limited computing resources**
  - once an online learning system has learned about new data instances, you can discard them
- Good choice to train systems on huge datasets that cannot fit in one machine's main memory (aka "**out-of-core learning**"): the algo loads part of the data, runs a training step on that data, and repeats the process until it has run on all of the data

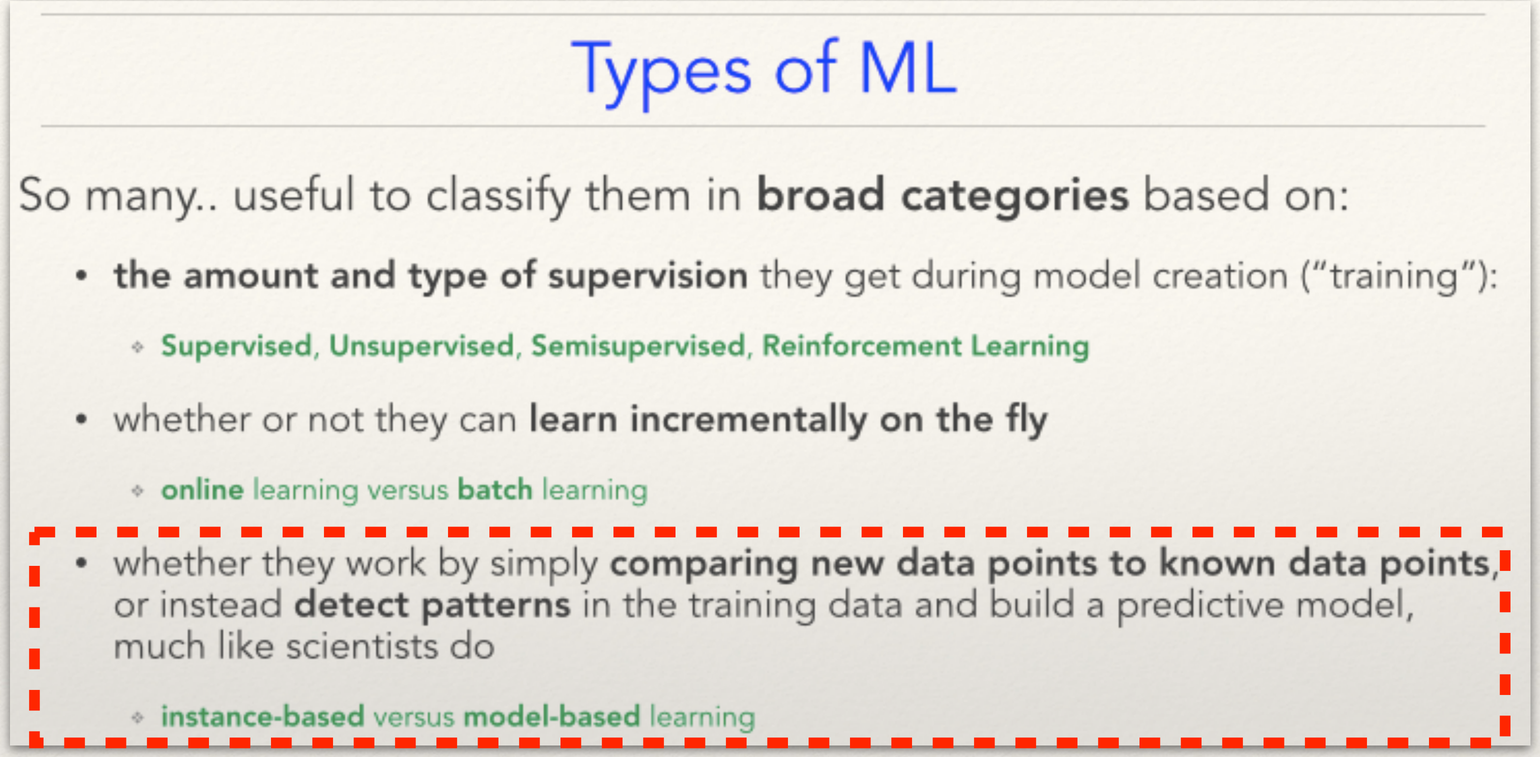
Online learning: **challenges**

One crucial aspect for online learning systems is how fast they should adapt to changing data: this is controlled via the **learning rate**

- set a high learning rate → the system will rapidly adapt to new data (but it will also tend to quickly forget the old data - e.g. not good for spam filters..)
- set a low learning rate → the system will have more inertia; the con is that it will learn more slowly, but the pro is that it will also be less sensitive to noise in the new data or to sequences of non-representative data points ("outliers")

Another big challenge with online learning regards **bad data fed to the system**: the system's performance will gradually decline

- To reduce this risk, one needs to monitor your system closely and promptly switch learning off (and possibly revert to a previously working state) if you detect a drop in performance
- One may also want to monitor the input data and react to abnormal data (e.g. using an anomaly detection algo).



Types of ML

So many.. useful to classify them in **broad categories** based on:

- **the amount and type of supervision** they get during model creation ("training"):
  - **Supervised**, **Unsupervised**, **Semisupervised**, **Reinforcement Learning**
- whether or not they can **learn incrementally on the fly**
  - **online** learning versus **batch** learning
- whether they work by simply **comparing new data points to known data points**, or instead **detect patterns** in the training data and build a predictive model, much like scientists do
  - **instance-based** versus **model-based** learning

Instance-based vs **Model-based** Learning

Classification of ML algos based on **how they generalise**.

Note that "**generalisation**" is key to success of a ML system

- data → training → ability to make predictions on previously unseen data
- key of applied ML (and all its art!) is to perform well on new instances!

The 2 main approaches to generalisation are:

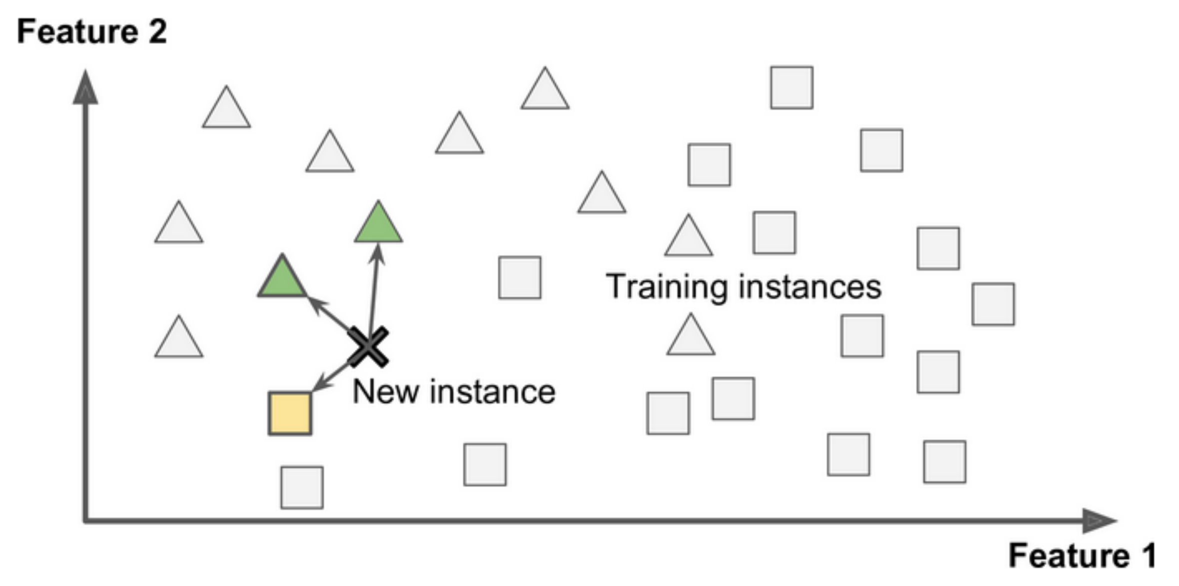
- **Instance-based Learning**
- **Model-based Learning**

Instance-based Learning

The most trivial form of learning is simply to learn by similarity

- with [A,B] as options, classify an instance as [A] if that instance is similar to a previous one in the training sample that had an [A] label
- of course, this requires a "measure of similarity" between instances

This is **instance-based learning**: the system learns, then generalises to new cases by comparing them to the learned examples (or a subset of them), using a similarity measure.



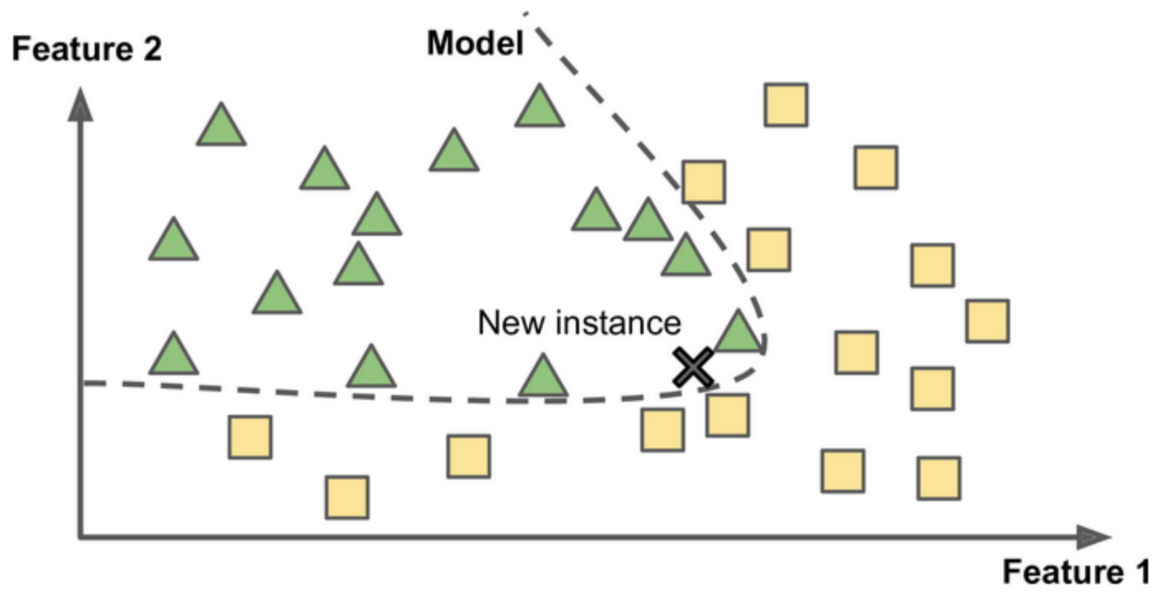
The new instance will be classified as a triangle **because most of most similar** (close-by in the features space) **training instances belong to that class**

Model-based Learning

In **model-based learning**, instead, a "model" for the training examples is built, then it is used to make predictions

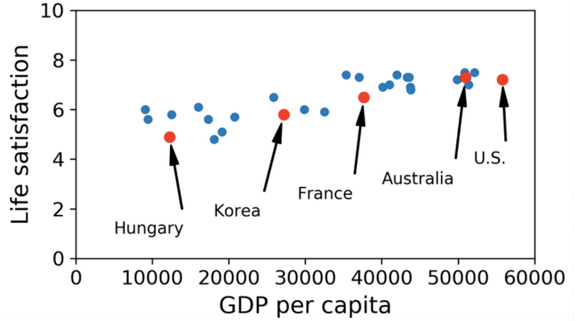
- a completely different way to generalise from a set of examples

You rely on a **model selection**, i.e. you reason on your data and make an **hypothesis** as of how each data **features** contribute to its **label**, and apply such model to make predictions



The new instance will be classified as a triangle **because a model exists that explain why triangles do populate that portion of the features space**

Model-based Learning



The basic steps

- study problem, define goals
- collect and plot data
- make an hypothesis
- write the mathematical relation
- specify a performance measure, either a **utility function** or a **cost function** (how good/bad my model is?)
- fit
- **inference**

Not good predictions? Use more attributes, get more or better quality training data, perhaps select a more powerful model (e.g. a Polynomial Regression model)…

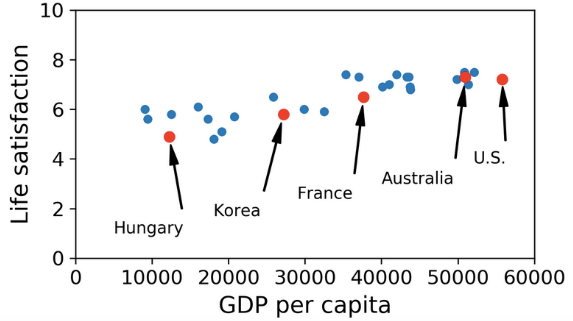
Example

- Are salary and happiness related?
- get salary data, get happiness rankings
- they look linearly correlated
- equation of a line: I need (θ_0, θ_1)
- cost function = distance between the linear model's predictions and the training examples. I need to minimize this.
- make a fit = train a model → (θ_0, θ_1)
- give me a country's average salary, **I apply my model** and I predict you the happiness ranking

This also is well known and called Linear Regression

~~Model~~-based Learning

Instance



The basic steps

- study problem, define goals
- collect and plot data
- make an hypothesis
- write the mathematical relation
- specify a performance measure, either a **utility function** or a **cost function** (how good/bad my model is?)
- fit
- **inference**

Example

- Are salary and happiness related?
- get salary data, get happiness rankings
- if I am asked happiness of country X, I check its average salary and look for the k countries with most similar salaries (Y_1, Y_2, .., Y_k), then I take their happiness rankings, I average them, and assign that average to country X as its estimated happiness rankings

This is a less well known algo called
k-Nearest Neighbours Regression

Breathing time: recap

With Amir earlier and myself now, we have covered a lot of ground so far, on:

- what Machine Learning is really about
- why it is useful
- what some of the most common categories of ML systems are
- what options are there in terms of ML types

Let's look at what can go wrong in learning and thus prevent you from making accurate predictions.

Main **challenges** in ML

Basically, related to:

- bad data
- bad algos

Main challenges in ML: **Insufficient Quantity of Training Data**

Bad data

Example: *how does a baby learn to spot car vs bus vs motorbikes?*

- try, dad gives labels, try, dad corrects, try, try, try, …
- you need traffic! quite examples, i.e. many cars, busses, motorbikes..

ML is like a baby learning as above, but way more stupid!

- well.. it can do in seconds what a baby takes months to learn..
- but needs **PLENTY** of examples to learn and perform in even simpler tasks

Needs for large volumes of data (**the Volume "V" of Big Data**..) is a need for advanced Machine/Deep Learning applications

- Reading material:
  - "Scaling to very very large corpora for natural language disambiguation" (Banko, Brill et al, 2001)
  - "The Unreasonable Effectiveness of Data" (Norvig et al, 2009)

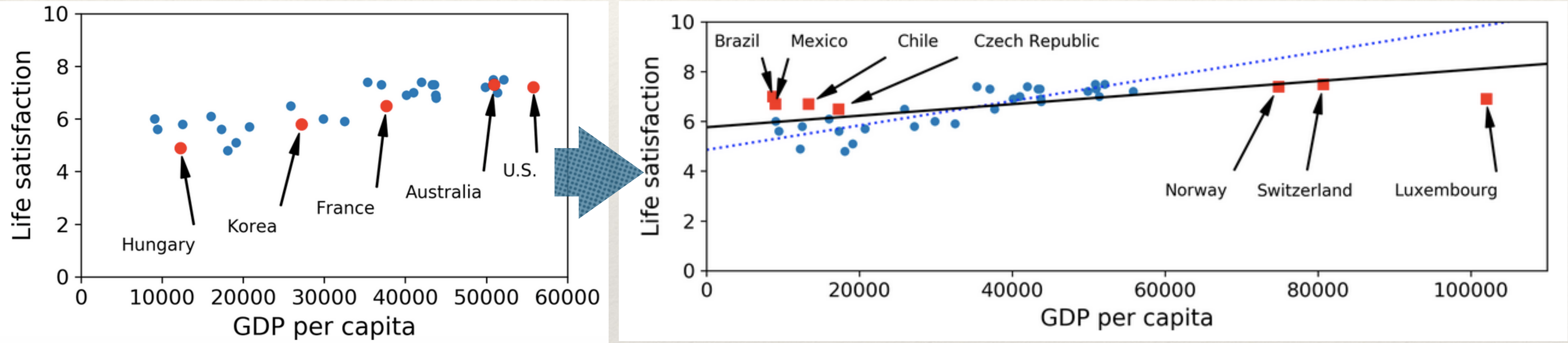
Main challenges in ML: **Non-representative Training Data**

Bad data

Careful: <u>not talking about the quality</u> of the data (that comes later)..

.. but about their **representativeness** of the problem such data is expected to describe

- e.g. are they complete? <u>Example</u>: YouTube search engine favours popularity..



Crucial to **use a training set that is representative of the cases you want to generalise to**

- if the sample is too small → "sampling noise" (i.e. non-representative data as a result of chance). Even if the sample is very large, if the sampling method is flawed it will still be non-representative. This is called "**sampling bias**"

Main challenges in ML: **Poor-Quality Data**

Bad data

ML systems are"**garbage in garbage out**"

If your training data is full of errors, outliers, noise (e.g. due to poor-quality measurements), the system will find it hard to detect any underlying patterns, so less likely to achieve decent performances

It is worth the effort to spend time cleaning up your training data. Best data science teams spend here a significant portion of their overall time on a ML project.

E.g. some instances missing a few features

- ignore the attribute altogether
- ignore the instances
- fill in the missing features (e.g. median): bias?
- train one model with the attribute and one model without it

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

[*credits: xkcd.com*]

Main challenges in ML: **Irrelevant Features**

Bad data

Again, ML systems are“**garbage in garbage out**”. Here, focus is on **features** and the learning process

Success in a ML project largely depends on your ability to feed a training process with features in your data that enable an effective learning process

This process is called “**feature engineering**“, and involves:

- **feature selection**: among existing and already collected features, select the most useful features to train on
- **feature extraction**: combining existing features to produce a (unreal?) more useful one
  - dimensionality reduction algos can help here
- **creating and adding new features** by gathering new data

Main challenges in ML: **Overfitting the Training Data**

Bad algos

Overfitting means that a model performs well on the training data, but it does not generalise well to new, previously unseen data. A serious issue.

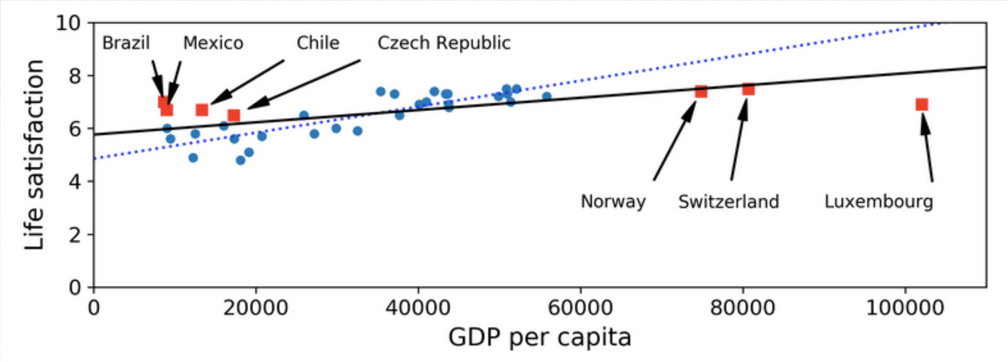
- unless cured, this issue makes a model completely useless

The source of the problem:

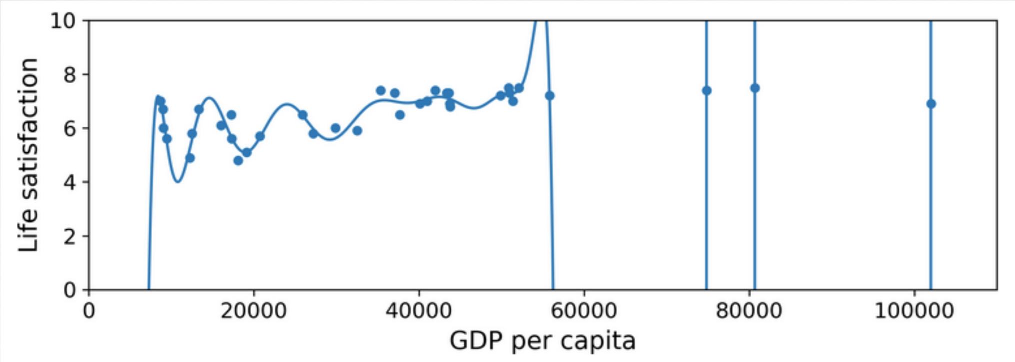
- The more complex a model is (DNN, or just high-degrees polynomials) the more it is able to detect and describe subtle patterns in the training data. But if the training set is noisy, or if it is too small (i.e. sampling noise), then the model is likely to detect patterns in the noise itself.

The cures include: **reducing the nb attributes**, **apply regularization**, **collect more training data** (ok here, but not effective - alone - in underfitting - see next)

- regularisation controlled via the λ hyperparameter (= parameter of the algo)



Linear model



High-degree polinomial model → **overfitting**!

Main challenges in ML: **Underfitting the Training Data**

Bad algos

Underfitting is the opposite of overfitting. Still a serious issue.

It occurs when your model is too simple to learn the underlying structure of your training data

The main options to cure this are:

- selecting a **more powerful model**, with more parameters
- **feature engineering**: feeding better features to the learning algo
- **reducing the constraints** on the model (e.g. reducing the regularization hyperparameter)
    - careful, as this may open doors to overfitting..

Training and **Testing** a model

The only way to know how well a model will generalise to new cases is to actually try it out on new data.

Basic choice is to **split your data** into two sets: the **training set** and the **test set**

- train your model using the training set, and test it using the test set
  - the error rate on new cases is called "generalisation error"
  - You can estimate this by evaluating your model on the test set
  - This value tells you how well your model will perform on instances it has never seen before
- If the training error is low but the generalisation error is high, it means that your model is overfitting the training data

More refined options do exist..

Model selection and Hyper-parameter tuning

You are hesitating between 2 ML models..

- train both and compare how well they generalise using the test set

Ok, this helped to pick one. Now, to fight overfitting, you apply regularisation. Which λ do you choose?

- you train 100 models with 100 values of λ, you find the λ that produces a model with the lowest generalisation error (e.g. 5%), and then you pick your model. You launch this to production and measure a 15% error. What?!
    - The problem is that you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that particular set. So presumably it will not perform well on new data

Solution: "holdout validation", with a **validation** set

- you train multiple models with various hyperparameters on the "reduced" training set (i.e. the full training set minus the validation set), and you select the model that performs best on the validation set.
- Then, the holdout validation process is over, you re-train the best model on the full training set (including the validation set), and this gives you the final model
- Lastly, you evaluate this final model on the test set to get an estimate of the generalisation error.

Cross-validation

This solution usually works quite well. However:

- if the validation set is too small, then model evaluations will be imprecise: you may end up selecting a suboptimal model by mistake
- if the validation set is too large, then the remaining training set will be much smaller than the full training set. This is bad because the final model will be trained on the full training set, it is not ideal to compare candidate models trained on a much smaller training set.
    - it is like selecting the best marathon runner from his performance on the first 500 m …

One way to solve this problem is to perform **repeated cross-validation**, using many small validation sets.

- Each model is evaluated once per validation set, after it is trained on the rest of the data. By averaging out all the evaluations of a model, we get a much more accurate measure of its performance.
- However, there is a drawback: the training time is multiplied by the number of validation sets.