PhD in Computer Science and Engineering
Bologna, April 2016

Machine Learning

Marco Lippi

marco.lippi3@unibo.it

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

What is machine learning ?

An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside [...] The learning process may be regarded as a search for a form of behaviour which will satisfy the teacher (or some other criterion).

Turing, A. (1950), Computing machinery and intelligence

A computer program 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.

Mitchell, T. (1997), Machine Learning

What is machine learning ?

Machine learning is currently **one of the hottest topic** within Artificial Intelligence and Computer Science in general.

All the most important ICT companies all over the world are **investing money** and **hiring people** with a strong background in machine learning: Google, Facebook, IBM, Baidu, Disney, ...

Why ?

Because they have **tons of data**...

...and they need **some way to make sense of it** !

An overwhelming amount of data

Data never sleeps (2014)

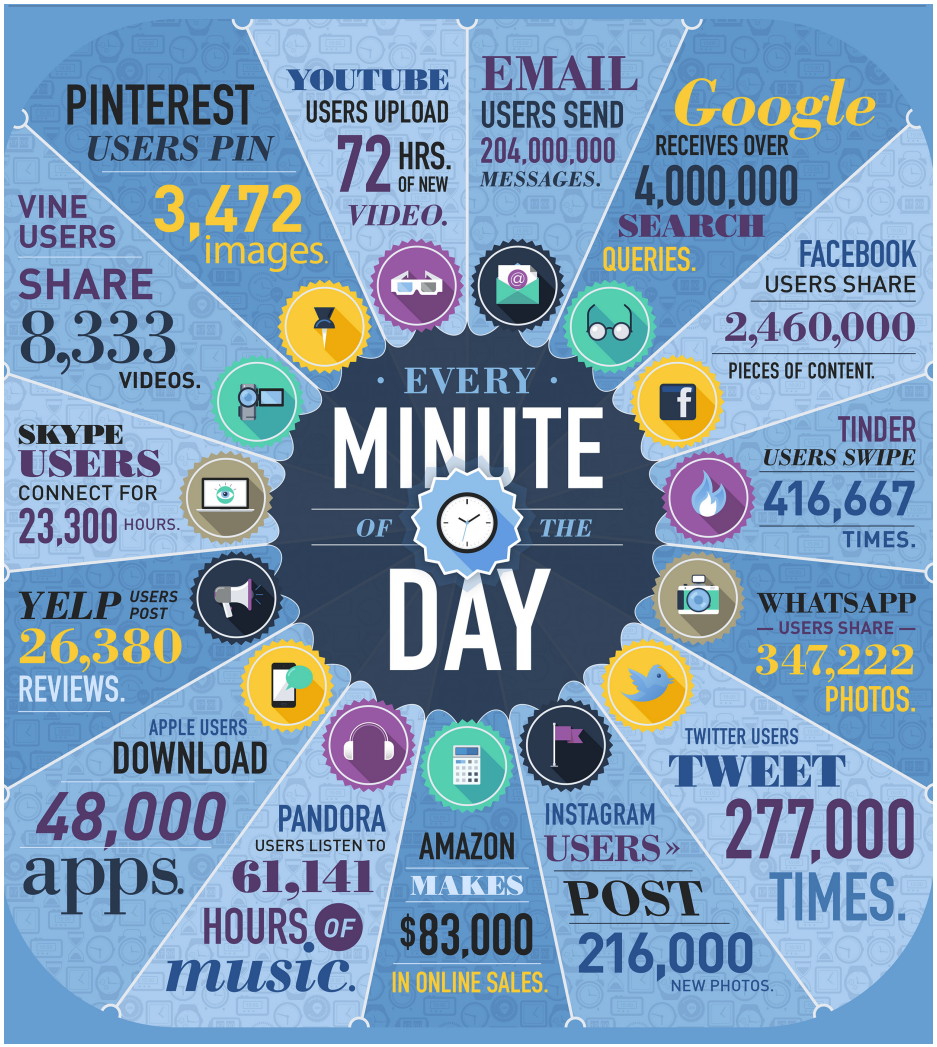


Figure source: http://www.domo.com/

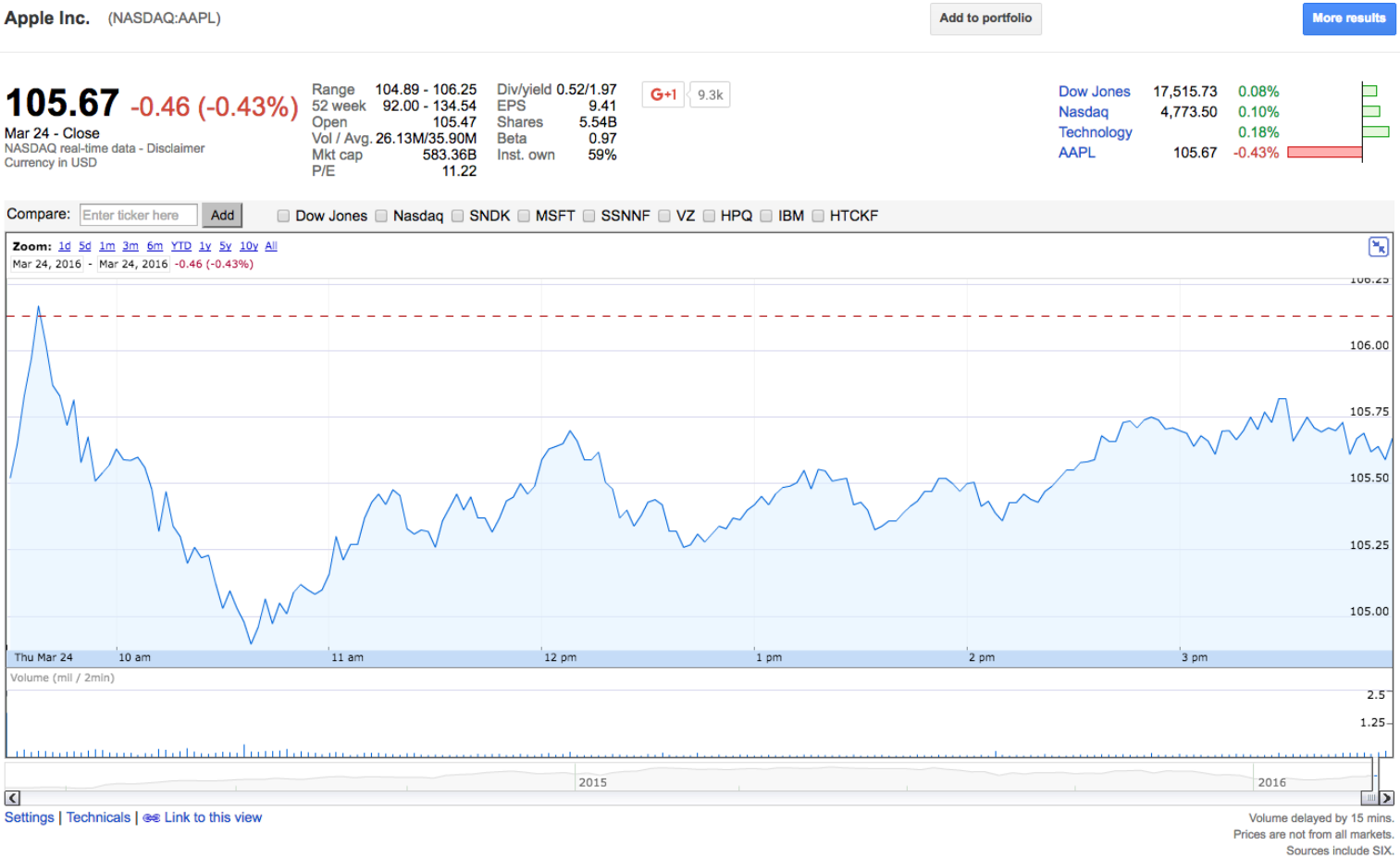
What can machine learning do ?

- Classify incoming e-mails as spam or not

Figure source: http://www.resilientsystems.co.uk/

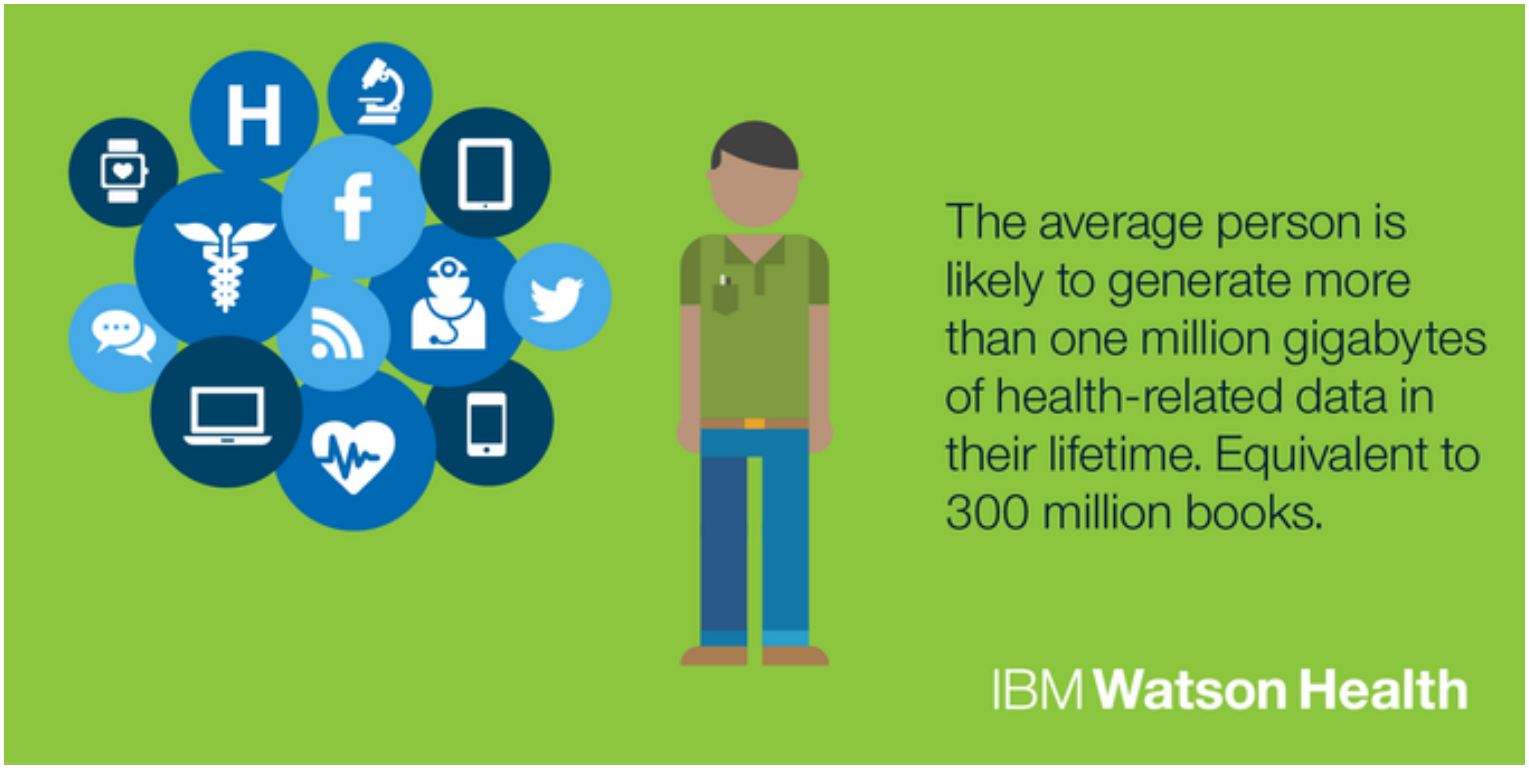
What can machine learning do ?

- Forecast Apple stock price in the next three hours



What can machine learning do ?

- Make a diagnosis based on patient examinations



What can machine learning do ?

- Recognize digits in zipcode forms

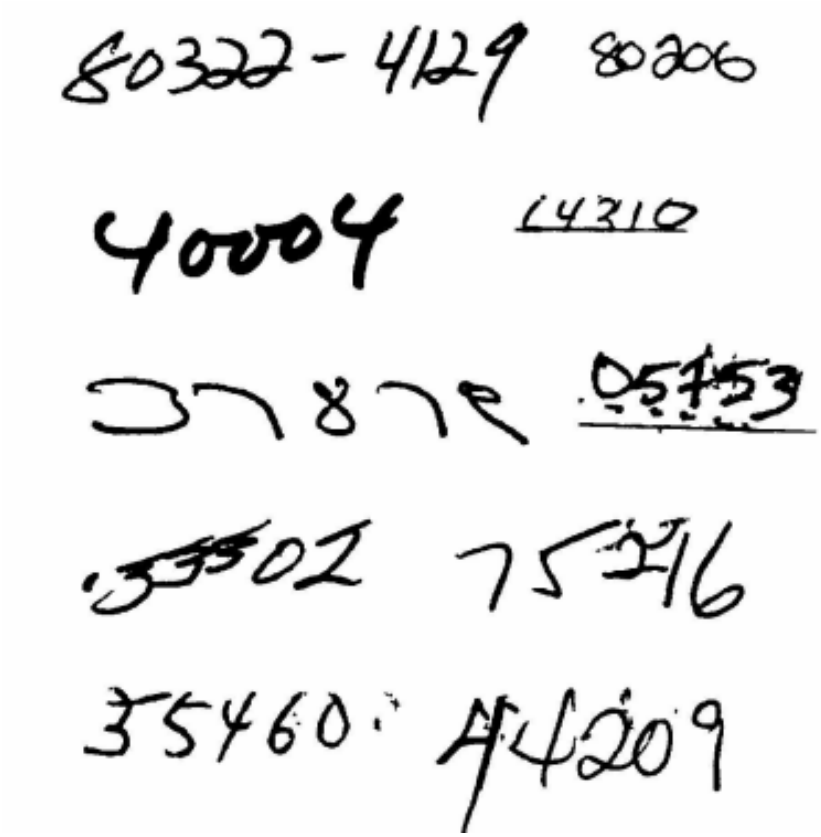


Figure source: [LeCun et al., 1989]

What can machine learning do ?

- Teach a robot to grasp a mug

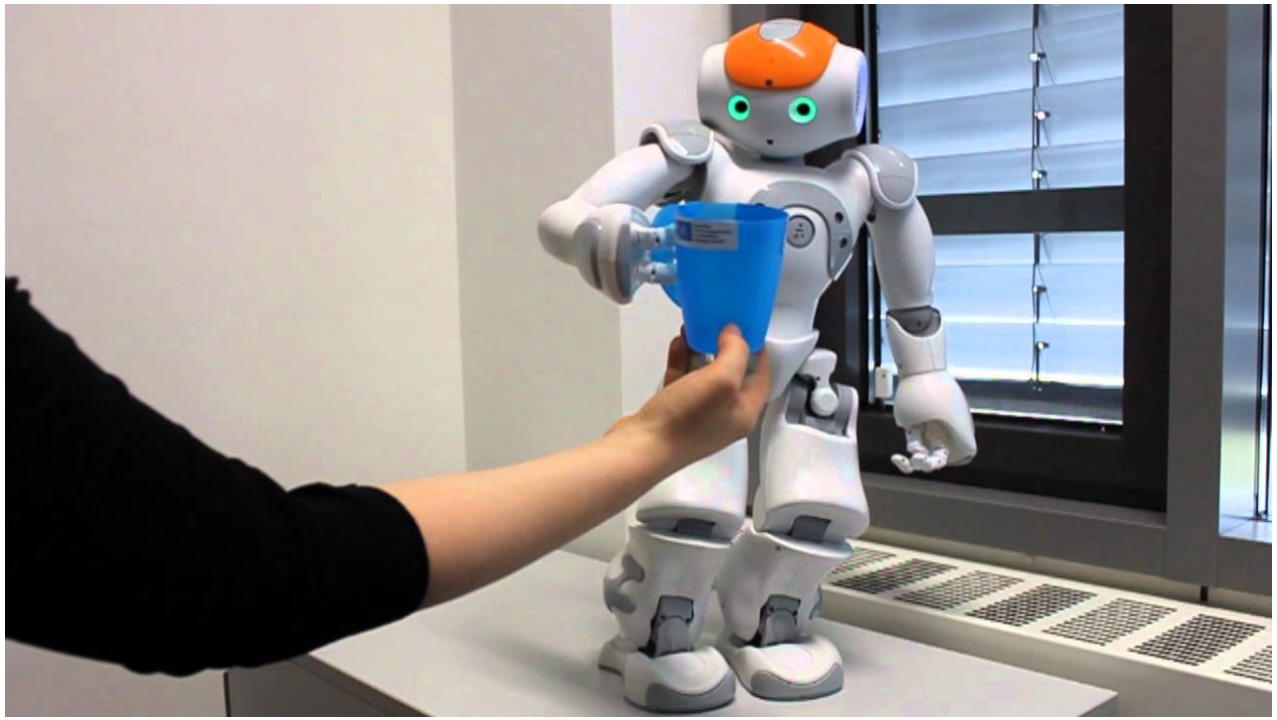


Figure source: http://www.informatik.uni-bremen.de/

What can machine learning do ?

- Design a drug/molecule with certain properties

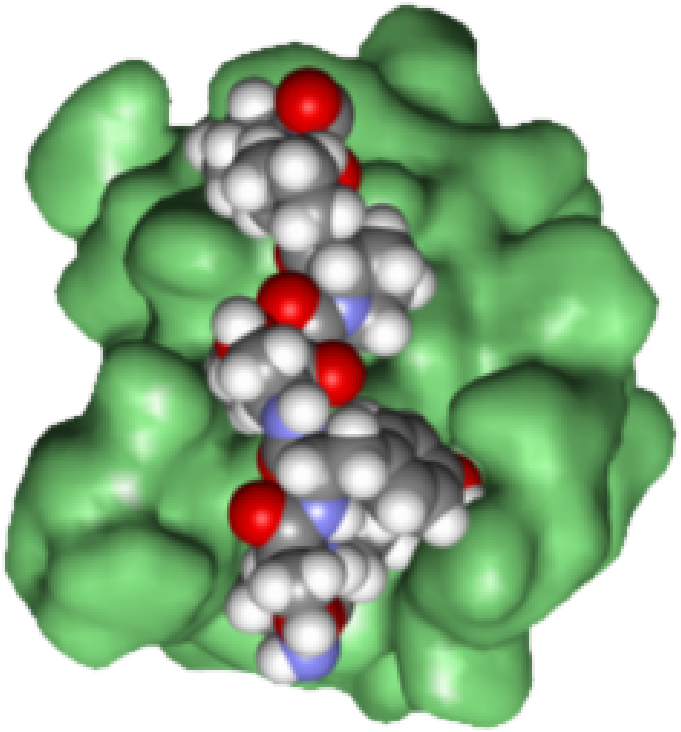


Figure source: http://pande.stanford.edu/

What can machine learning do ?

- Translate text from one language into another

Google Translate

What can machine learning do ?

- Convert a speech into text

What can machine learning do ?

- Automatically write the caption of an image

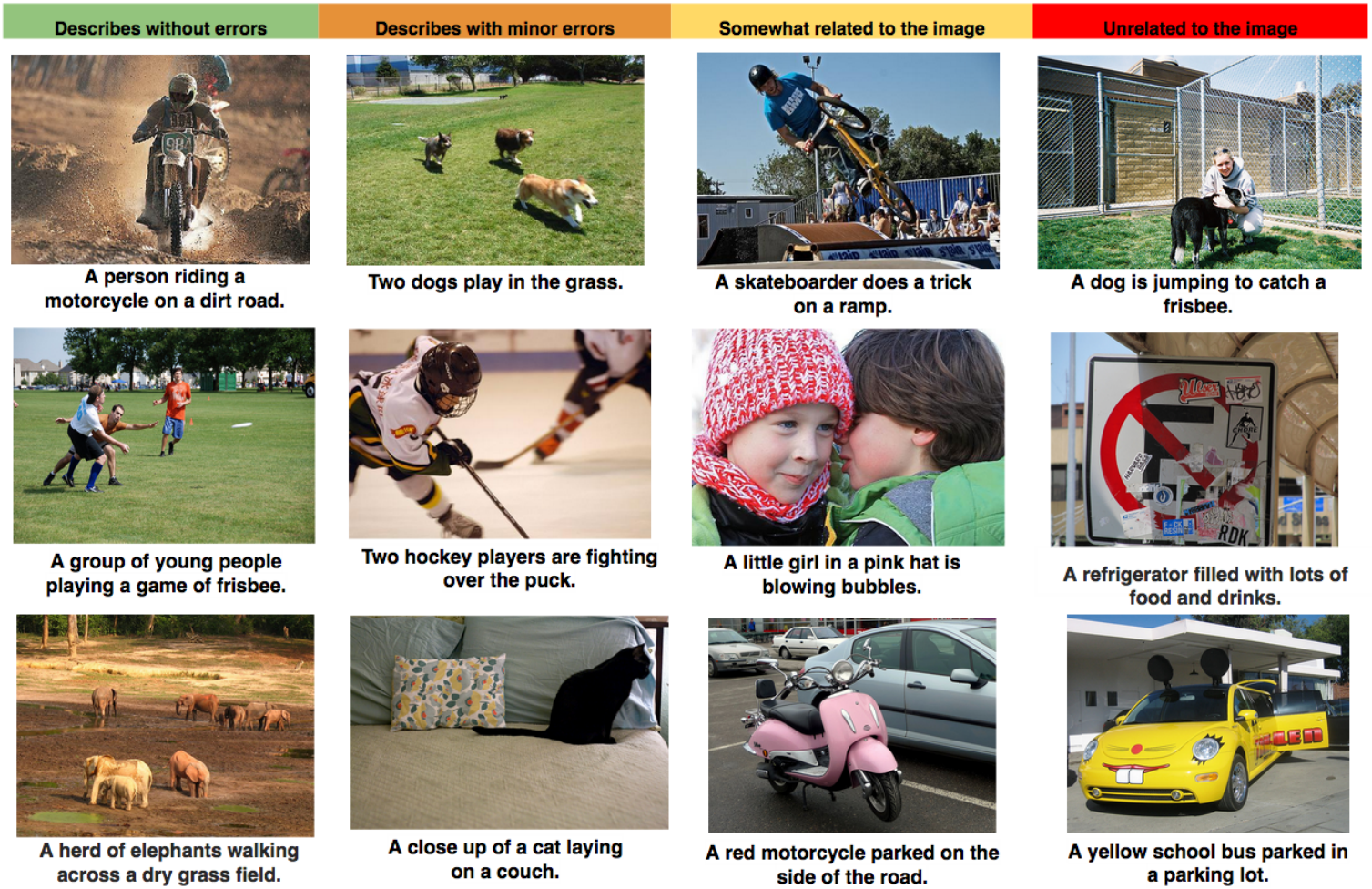


Figure source: Google Research

What can machine learning do ?

- Beat the world's top Go player (March 2016 !)

Figure source: Google Research

Basics

Basic concepts

Learning **paradigms**:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

How to **assess the quality** of a machine learning system ?

- Performance measurements
- Overfitting and cross-validation

General **categories** of machine learning models

- Symbolic approaches
- Sub-symbolic (connectionist) approaches

Supervised learning

Supervised machine learning problem:

- find a mapping between input/output variables, directly **from data observations**

Main framework:

- given **observation** $x \in \mathcal{X}$
- **target** of prediction $y \in \mathcal{Y}$
- **data set** $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$
- build a **function** $f : \mathcal{X} \to \mathcal{Y}$
- f must be able to predict the value of $\hat{y}$, given some $\hat{x}$

Supervised learning

Different algorithms, methods and problems according to:

- the kind of **input** space $\mathcal{X}$
- the kind of **output** space $\mathcal{Y}$
- the choices made for the **definition of function** f
- the choices made for the **computation of function** f

Classic problems

Possible input spaces:

- $\mathcal{X}$ = numerical and/or attributes
- $\mathcal{X}$ = logic representations
- $\mathcal{X}$ = some structured space (tree, graph, etc.)

Possible output spaces:

- $\mathcal{Y} = \{\pm 1\} \rightarrow$ binary classification
- $\mathcal{Y} = \{1, \ldots, M\} \rightarrow$ multi-class classification
- $\mathcal{Y} = \Re \rightarrow$ regression
- $\mathcal{Y}$ = some structured space $\rightarrow$ structured output

Classic problems

A very classic example: **least-squares linear regression**

- $\mathcal{X} = \Re^d$
- $\mathcal{Y} = \Re$
- $\hat{y} = f(x) = \sum_{k=1}^{d} \alpha_k x_k$
- $\alpha = \arg\min \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$

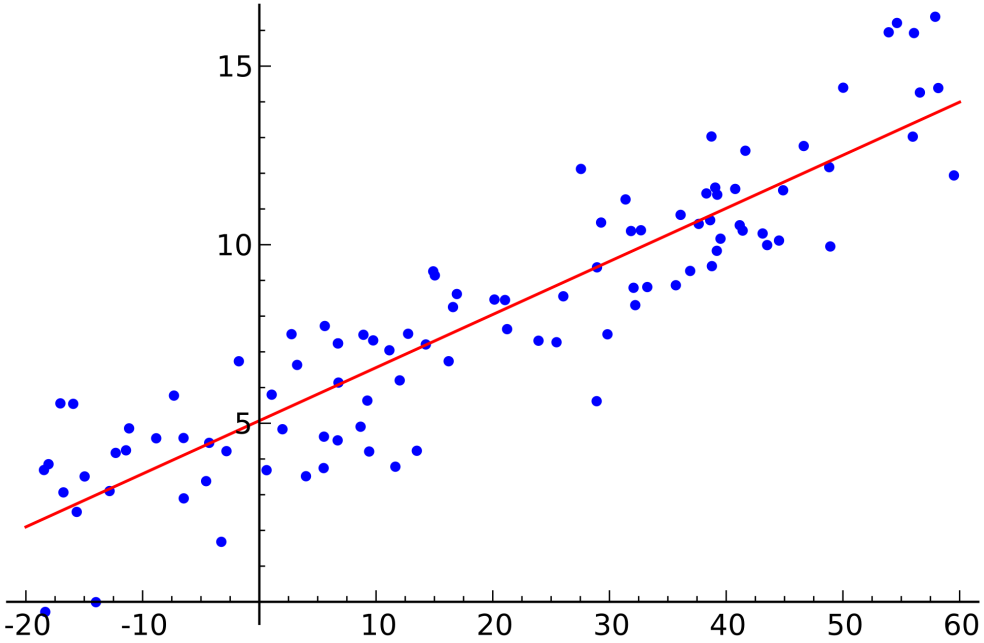


Figure source: Wikipedia

Unsupervised learning

Setting:

- we are given **no supervision**, just data
- we may **not even know the task** to perform on such data

So. . . Why is it useful. . . ?

- **Cluster** data, find (frequent) **patterns**
- Identify **most important features**
- Reduce **data dimensionality** (encoding/compression)

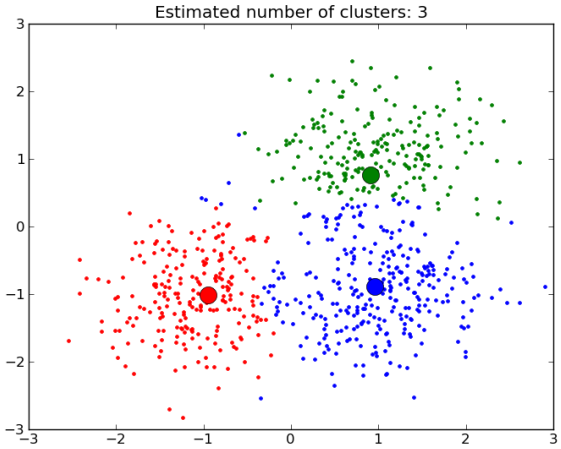


Figure source: Wikipedia

Supervised vs. unsupervised learning

Supervised learning

- ☺ labels are very useful
- ☺ function to be learned clearly defined
- ☹ labels can be very costly

Unsupervised learning

- ☺ unsupervised data are everywhere
- ☺ can extract the most useful information from data
- ☹ worse performance for a highly specific task

Semi-supervised learning ?

Reinforcement learning

A different paradigm: teacher giving **rewards/penalties**

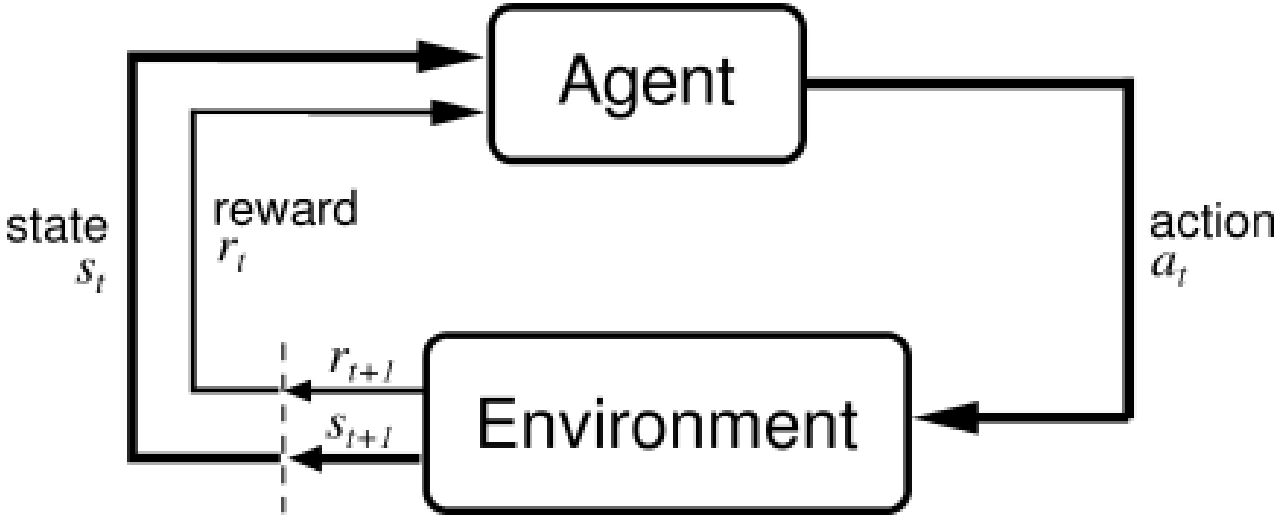


Figure source: web.stanford.edu

Largely used in **many contexts**: robotics, agents, control, ...

Reinforcement learning

A great example: computers playing Atari (by Google DeepMind)

Figure source: Wikipedia

Performance measures

When we are given a problem and we build a ML system to solve it, it is crucial to know **how to assess its performance**.

We basically need to **count the errors** of our system...
...But this is not so trivial as one could expect !

Example

- We have to build a diagnosis system for a very rare pathology
- Our system correctly classifies examples 99.9% of the times
- Can we be happy with such a classifier ?

Performance measures (binary classification)

		True Class	
		0	1
Pred Class	0	TN True Negative	FN False Negative
	1	FP False Positive	TP True Positive

- Accuracy $A = \frac{TP+TN}{TP+TN+FP+FN}$
- Precision $P = \frac{TP}{TP+FP}$
- Recall $R = \frac{TP}{TP+FN}$
- F-Measure $F_1 = \frac{2PR}{P+R}$

Performance measures (multi-class classification)

		True Class		
		A	B	C
Pred Class	A	TP (A)	FP (A) FN (B)	FP (A) FN (C)
	B	FP (B) FN (A)	TP (B)	FP (B) FN (C)
	C	FP (C) FN (A)	FP (C) FN (B)	TP (C)

- Accuracy $A = \frac{\sum_i TP_i}{\sum_i TP_i + FN_i} = \frac{\sum_i TP_i}{\sum_i TP_i + FP_i}$
- Precision $P^i = \frac{TP_i}{TP_i + FP_i}$
- Recall $R^i = \frac{TP_i}{TP_i + FN_i}$
- F-Measure $F_1^i = \frac{2 P_i R_i}{P_i + R_i}$

Performance measures (binary classification)

If a classifier produces a **score** (or a **probability**) for each example, rather than just the class, one can compute:

- Recall-Precision Curve
- ROC (Receiver Operating Characteristic) Curve

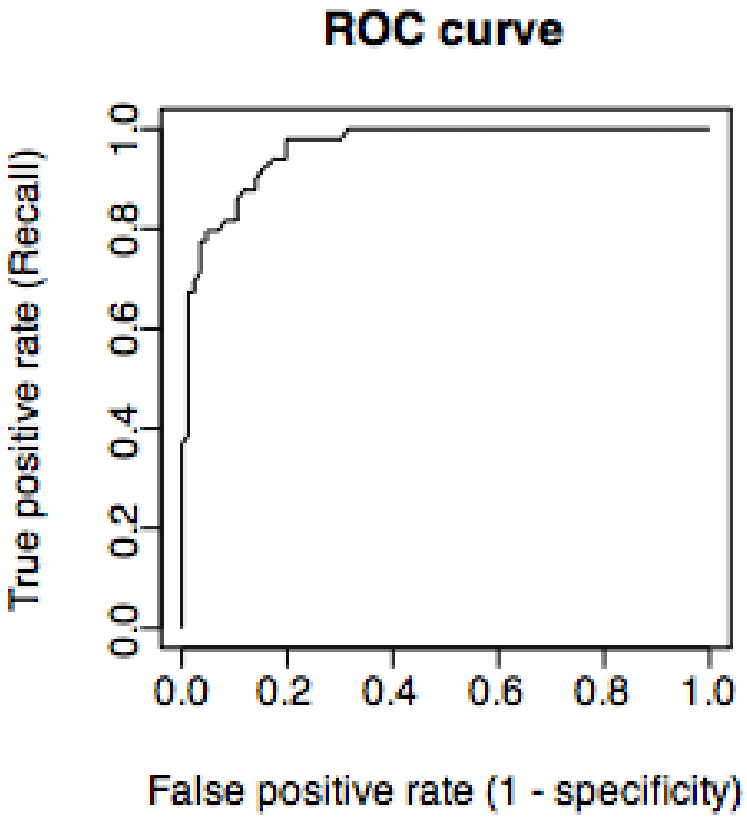
The curves are obtained by varying the threshold for the positive class, by using the scores produced in output by the classifier.

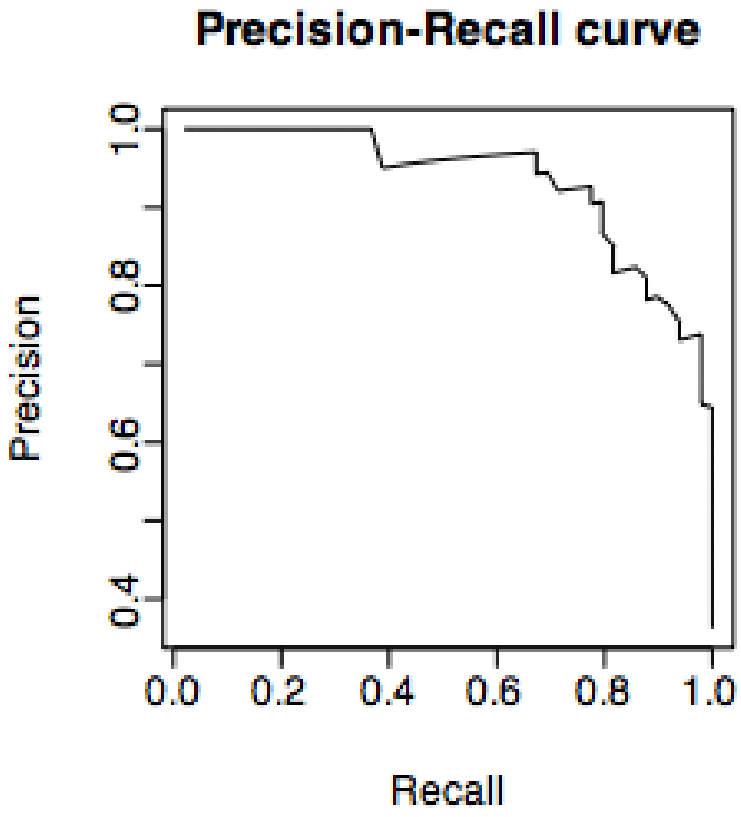
For **Recall-Precision** curves, Precision (y-axis) is reported as function of Recall (x-axis).

For **ROC** curves, *FPR* is reported on x-axis, and *TPR* on y-axis:

$$TPR = \text{True Positive Rate} = \text{Recall}$$
$$FPR = \text{False Positive Rate} = 1 \text{ - Precision}$$

Performance measures (classification)





Computing area under the curves (software):
`http://mark.goadrich.com/programs/AUC/`

Performance measures (regression)

- Mean squared error (MSE)
- Root mean squared error (RMSE)
- Mean absolute deviation (MAD)

$$MSE = \sum_{i=1}^{N} \frac{(\hat{y}_i - y_i)^2}{N}$$

$$RMSE = \sqrt{MSE}$$

$$MAD = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i|$$

Model selection

The performance of any machine learning system strongly depends on a set of **parameters**, whose values have to be chosen during the training phase.

The procedure through which we choose the best parameters for a machine learning system is called **model selection**.

Model selection

A typical approach to model selection splits the data into:

- training set (or learning set)
- validation set
- test set

The **training set** is used to learn function f.
The **validation set** is used to measure the performance of the model during paramater selection.
The **test set** is used to measure the effective performance of the predictor on unseed data.

The data split **should maintain** the proportion among classes.

K-fold cross validation

Split the data into K sets, named **folds**

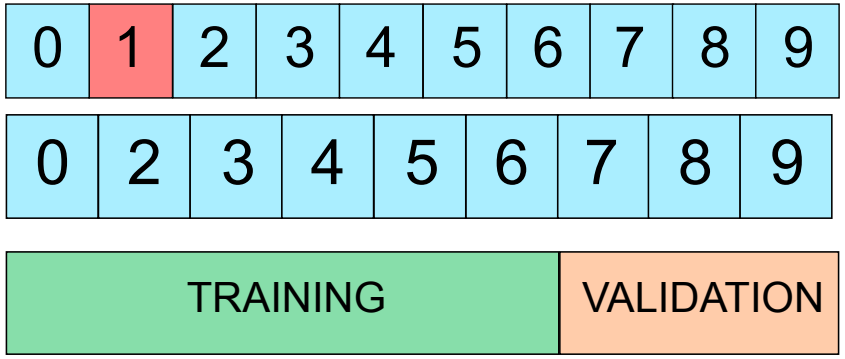
0	1	2	3	4	5	6	7	8	9

In turn, each fold will act as **test set**, whereas the remaining folds will form the **training and validation sets**

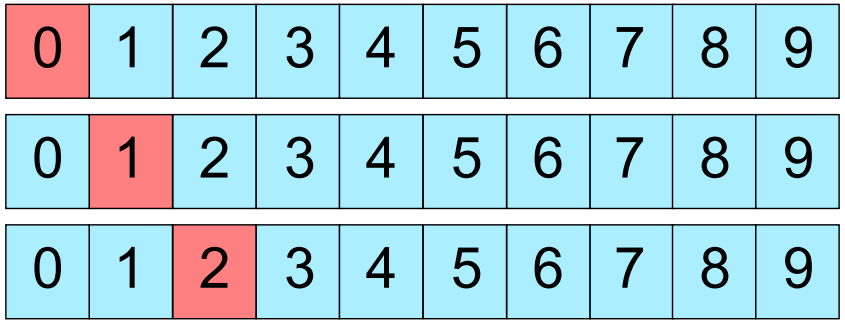
0	1	2	3	4	5	6	7	8	9

K-fold cross validation

Training and validation sets are employed to select the model, whereas the test set is used to compute the performance



The procedure is repeated K times for all the folds



Macro-average vs. micro-average

Statistics computed over the K folds are reported.
Suppose to have the following results from a 3-fold cv:

- TP=5, TN=10, FP=2, FN=3
- TP=3, TN=12, FP=4, FN=1
- TP=8, TN=11, FP=0, FN=1

We have two possibilities:

1. compute P, R, F_1 for each fold and then average (**macro-average**)
2. sum up all TP, TN, FP, FN and compute a single P, R, F_1 (**micro-average**)

Why do we need cross-validation ?

A machine learning system is usually trained to **minimize some cost (loss) function**, thus it should not be surprising that it is capable to have a low error on the examples seen during training !

Main drawback

One could learn **perfectly** the training set, while being completely **unable to generalize** across different examples !

This is called **overfitting** $\rightarrow$ that is why we need cross-validation !

A first classifer: k-Nearest Neighbors

A very simple supervised classifier:

- use some **distance** to measure distance between examples
- predict the class of a given example as the **most frequent class** among the k nearest examples in the training set
- possibly compute a **weighted** voting (using distances)

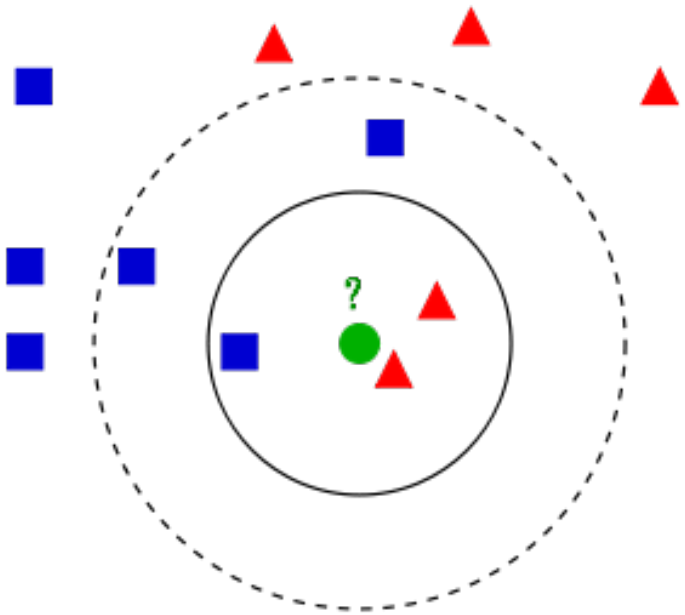


Figure source: Wikipedia

A first classifer: k-Nearest Neighbors

Despite simplicity (no learning !), it may work well in practice. . .

- when dealing with a **large training set**
- when a **distance measure** is employed

. . . but beware of the following drawbacks:

- when dealing with a **large training set** ☺
- when dealing with **high dimensional spaces**

No free lunch theorem

Wolpert & MacReady, 1997

Any two learning algorithms are equivalent when their performance is averaged across all possible problems

No single model that works best for every problem

- looking only at data is **not enough**
- a model needs some assumption to **generalize**
- all we know about data is what we observe (Hume)

Example

- two Boolean input features, one binary output class
- two examples are given: ((0,0),0) and ((1,1),1)
- how to predict output for the two missing cases ?

Bias-variance tradeoff

The **expected generalization error** of a learning algorithm can be **decomposed** into **bias**/**variance** terms:

$$Error = (E[f(x)] - y)^2 + Var[f(x)] + \sigma^2$$

Bias error due to:

- wrong assumptions of the learning algorithm (underfitting)

Variance error due to:

- large variations with small perturbations (overfitting)

Low bias $\rightarrow$ accurate esitmate on average
Low variance $\rightarrow$ estimate not changing much w.r.t. training set

Bias-variance tradeoff

Bias-variance **trade-off** (or **dilemma**)

Procedures with increased flexibility to adapt to the training data (e.g., have more free parameters) tend to have **lower bias** but **higher variance**, and the other way round

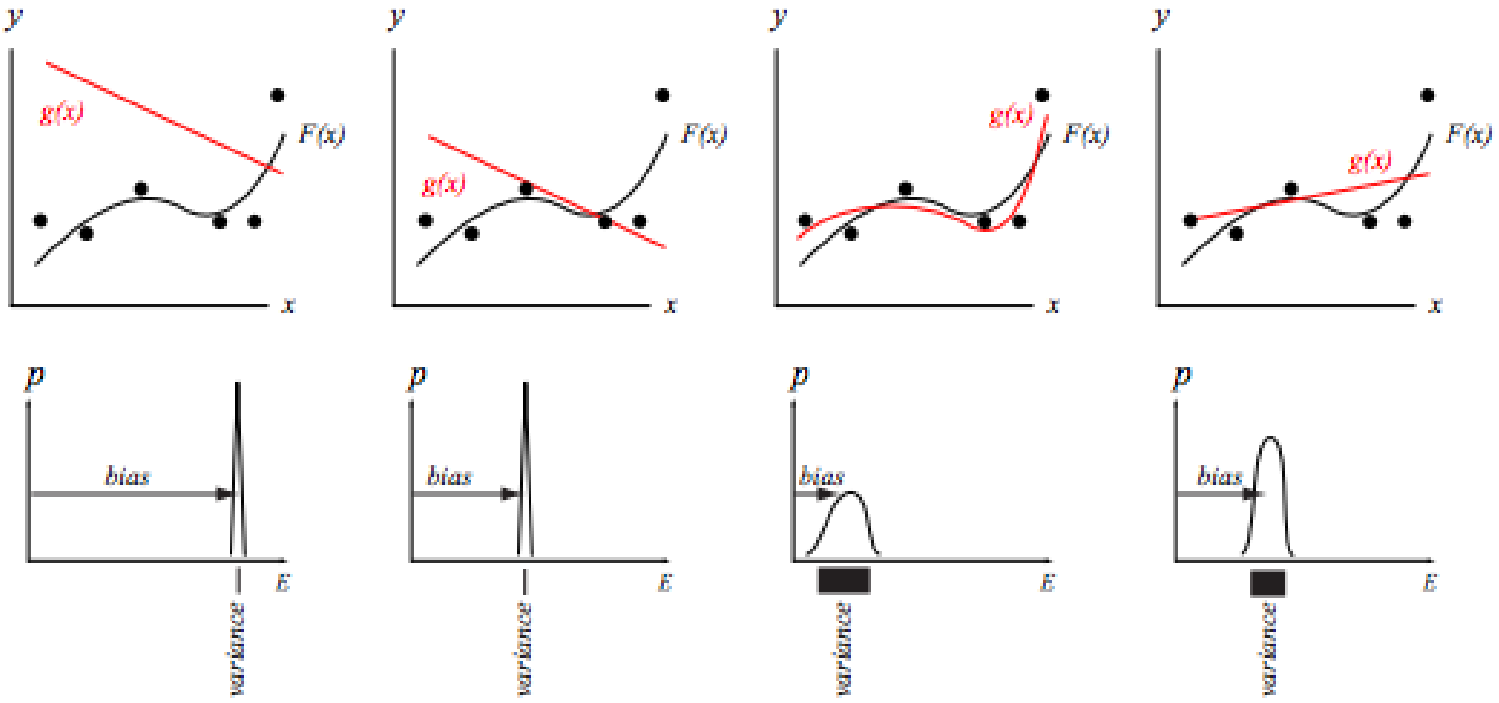


Figure source: [Duda et al., 2000]

Regularization

Which function is to be preferred ? Green or blue ?

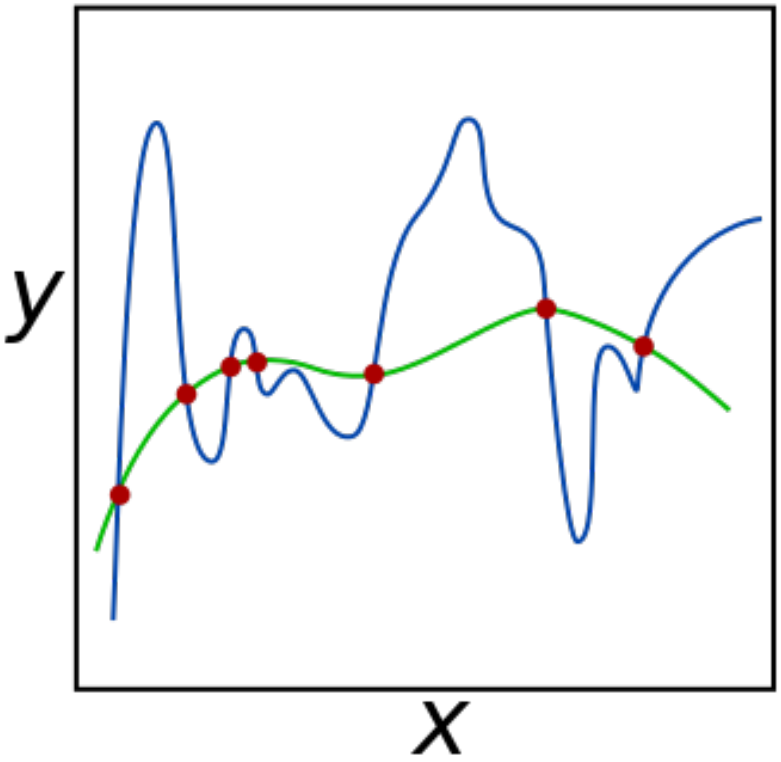


Figure source: Wikipedia

Regularization

Many approaches minimize a function composed of two terms:

$$\min_f \sum_{i=1}^{N} V(f(\hat{x}_i), \hat{y}_i) + \lambda R(f)$$

- $V(f(x), y)$ is a **loss function** measuring how much f **misclassifies** the training examples
- $R(f)$ is a **regularization term** that typically **penalizes the complexity** of f

Examples of regularization in case of $f(x) = w^T x$:

- $\|w\|_2^2 \rightarrow$ Tikhonov regularization
- $\|w\|_1 \rightarrow$ sparsity regularization
- ...

The key problem: feature engineering

Probably **the most important factor** for a machine learning application to succeed or fail:

- features correlating well with the class $\rightarrow$ easy ☺
- class as a complex function of features $\rightarrow$ tricky 😐
- raw data has no evidence of correlation with class $\rightarrow$ hard ☹

In the last case, **constructing** good features as a pre-processing step becomes the **crucial ingredient** of the application

One major aim of machine learning is to **feature learning** (a.k.a. **representation learning**) directly from data

Connectionist vs. symbolic approaches

A central question in AI

How do knowledge and information is represented in our mind ?

Symbolic approaches:

- reasoning as the result of formal manipulation of symbols

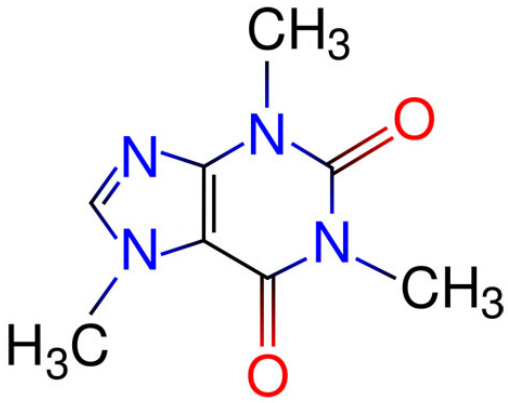
Connectionist (sub-symbolic) approaches:

- reasoning as the result of processing of interconnected (networks of) simple units

What is the solution ?

Connectionist vs. symbolic approaches

- Symbolic approaches have a **high interpretability**



toxic(m) :- doublebond(m,c1,c2), hydroxyl(c2), methyl(m)

- Connectionist approaches can be easily **distributed**, decisions are **smooth**, undertake a **graceful degradation**, and can easily handle **uncertainty and incompleteness** in data

What else ?

What we will **not** cover in this course:

- Clustering
- Reinforcement learning
- Computational learning theory
- Genetic algorithms
- Bayesian learning
- Graphical models
- Active learning
- ...

Introduction to Artificial Neural Networks

Artificial Neural Networks

Artificial Neural Networks (ANNs) are one of the **most famous** and **long-standing** learning paradigms developed in AI:

- inspired by the architecture of **human brain**
- highly **parallel** and **distributed** computable units
- the archetype of **connectionist** model

Throughout the years, research in ANNs has alternated periods of great **excitement** and **delusion**. . .

. . . Currently, there is a huge excitement in the area ! ☺

Biological Neural Networks

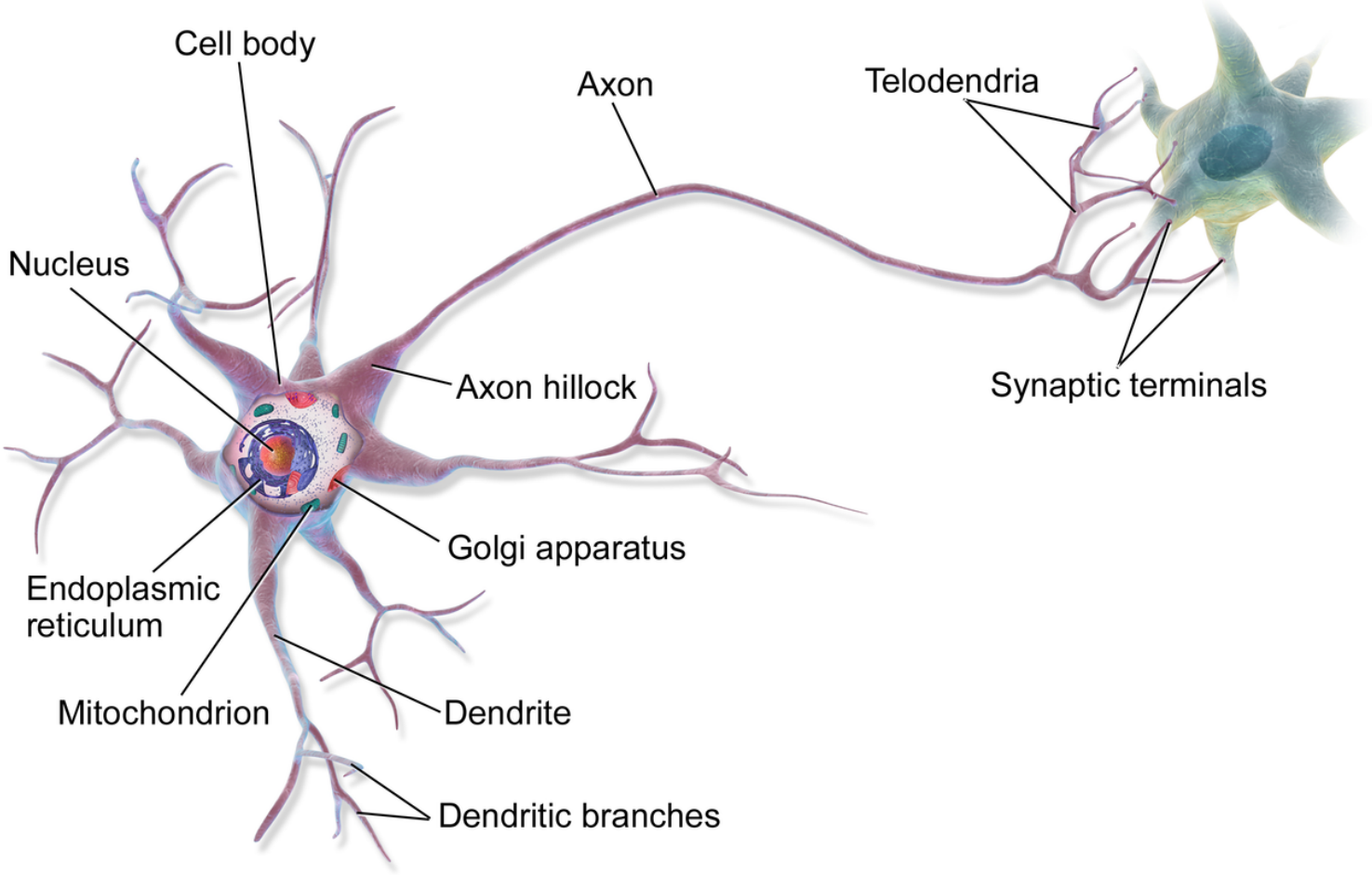


Figure source: Wikipedia

1943 — **McCulloch and Pitts**: the artificial neuron

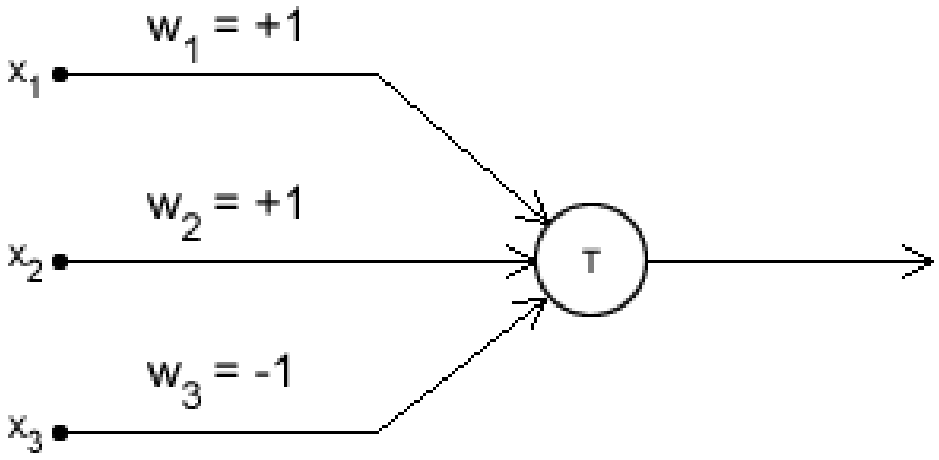


Figure source: http://aishack.in/

Fixed weights, binary input/output !

1958 — **Rosenblatt**: perceptron

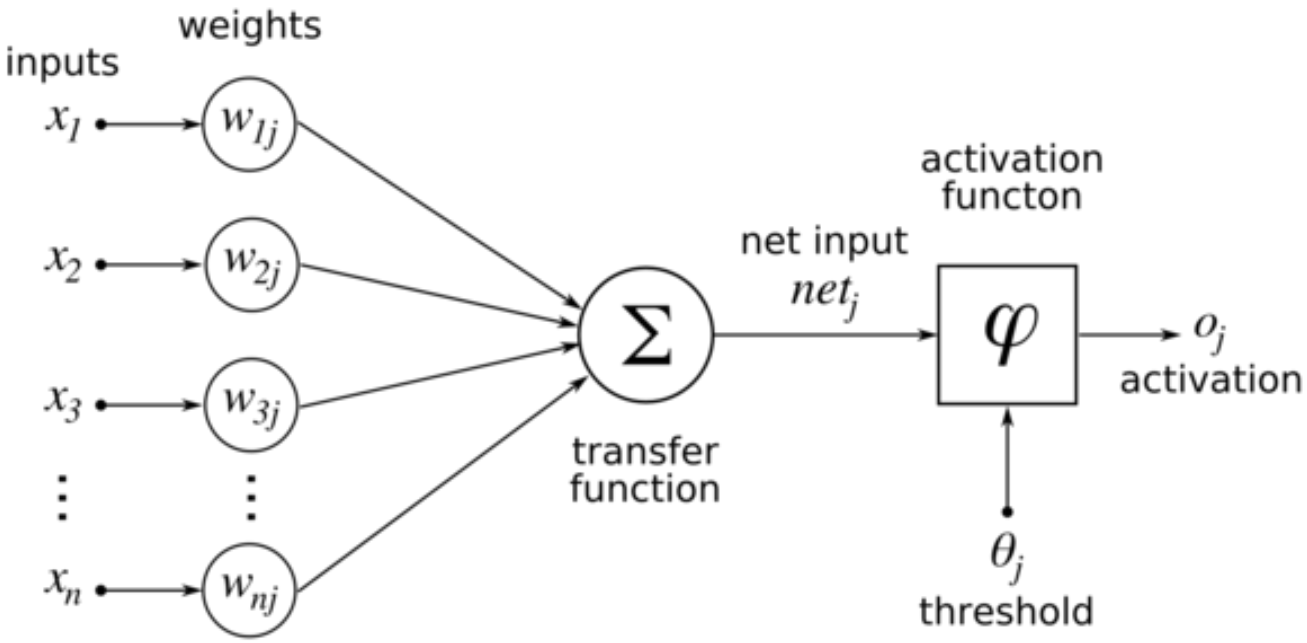


Figure source: Wikipedia

Weights can be learned, input/output not binary !

A short history of Artificial Neural Networks...

1969 — **Minsky and Papert**: limits of the perceptron

Even simple functions (e.g., XOR) cannot be learned...
Only **linearly separable** problems can be solved !

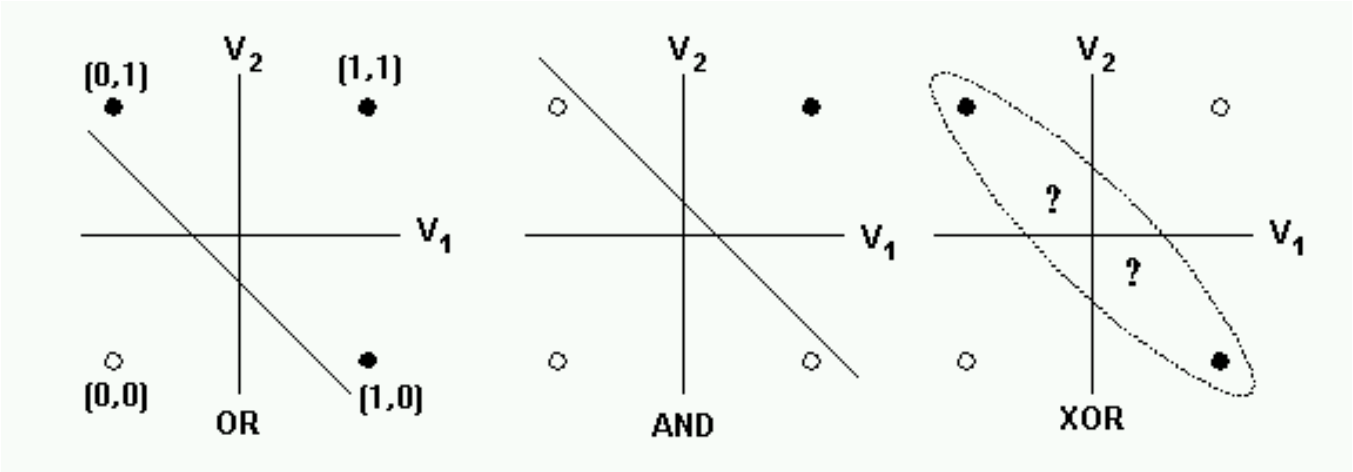


Figure source: http://colorado.edu

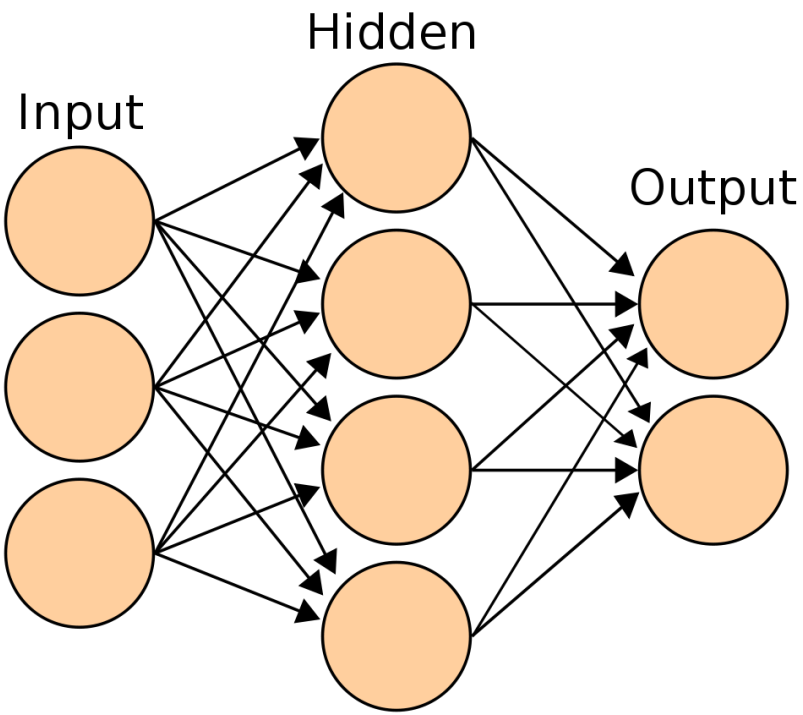
AI Winter ☹

A short history of Artificial Neural Networks...

1974 — **Werbos**: first ideas of backpropagation

1984 — **Hopfield**: Hopfield Networks

1986 — **Rumelhart, Hinton, Williams**: backpropagation ☺



The main idea of backpropagation

- Init the network with random weights
- For each example in a training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ repeat:
  1. compute a forward pass on the network
  2. propagate the errors backward from the output
  3. update the weights
- Compute the error on the validation set
- Stop training when some criterion is met

Backpropagation

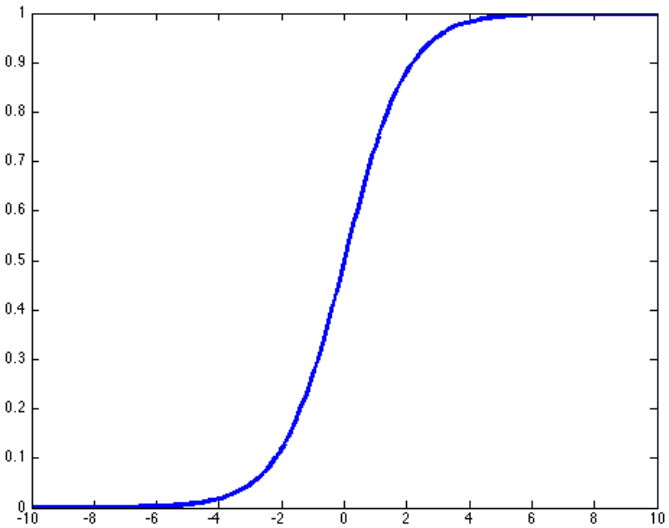
The error function is usually a **quadratic loss**:

$$E(w) = \sum_{i=1}^{N} (y_i - o_i)^2$$

The output of neuron j is activated by a **sigmoid function**:

$$o_j = \sigma(net_j) = \sigma\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Backpropagation

How to optimize the loss function ?

- **batch** optimization (update w_{ij} after each epoch)
- **stochastic gradient descent** (update w_{ij} after each example)
- **mini-batch** (update w_{ij} after a bunch of examples)

For each epoch (or example) t:

$$w_{ij}^{t} = w_{ij}^{t-1} - \eta \Delta w_{ij}$$

where:

$$\Delta w_{ij} = \frac{\partial E}{\partial w_{ij}}$$

and η is the **learning rate**

Backpropagation

How to compute $\frac{\partial E}{\partial w_{ij}}$?

Use the **chain rule**:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

(The activation function is required to be **differentiable**)

Example: for an output neuron, it is easy to compute:

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2}(y_j - o_j)^2 = o_j - y_j$$

Backpropagation

Main drawbacks:

- guarantee to converge only to a **local minimum**
- problems with **plateau regions** of the error function

Several tricks to circumvent the problem:

- **multiple restarts** with different initializations
- use a **momentum** term to move even when gradient is zero

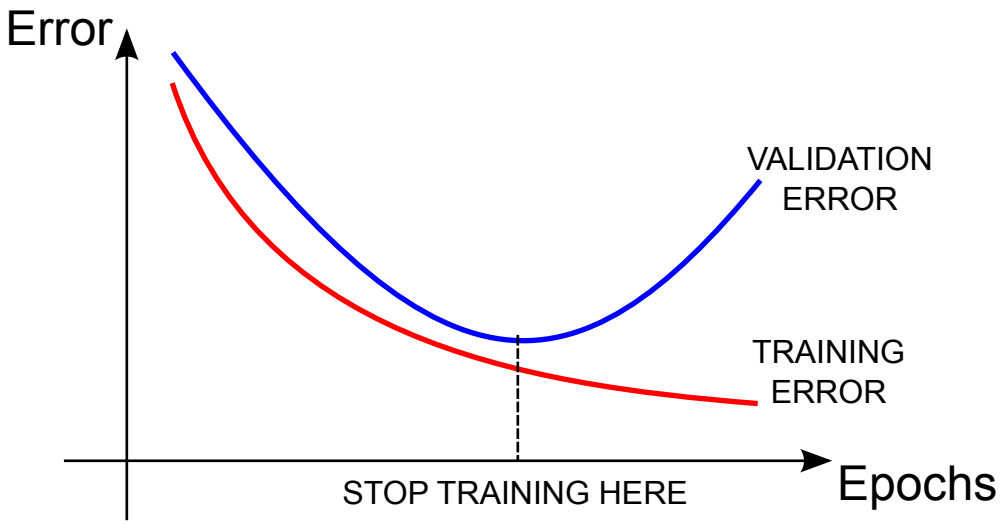
$$w_{ij}^{t} = w_{ij}^{t-1} - \eta \Delta w_{ij}^{t} - \mu \Delta w_{ij}^{t-1}$$

Training a neural network: tricks of the trade

How to setup:

1. the number of epochs
2. the number of layers
3. the number of hidden units in each layer
4. the learning rate

Given a network with a fixed architecture (parameters 2-3-4), the number of epochs is chosen following the **early stopping** principle



Training a neural network: tricks of the trade

- The **number of hidden layers**, until the recent "deep learning revolution", was typically 1 (2 at most)
- The **number of hidden neurons** should be chosen according to the **validation error**
- The **learning rate** has to be carefully chosen with a trial-and-error procedure (it is **problem-dependent** !)

Training a neural network: tricks of the trade

Choosing the learning rate is tricky !

- **too small**: slow convergence
- **too large**: divergence (might miss the optimum)

Many approaches employ an **adaptive** learning rate:

- if last step was good $\rightarrow$ **increase**
- if last step was poor $\rightarrow$ **decrease**

Multi-class classification

Network architecture:

- one output node per class
- **one-hot encoding** of classes

Target 3 out of 5 classes $\rightarrow$ 0 0 1 0 0
Target 5 out of 5 classes $\rightarrow$ 0 0 0 0 1

Categorical attributes

Consider for example some attributes of PhD students:

- Male/Female
- Nationality
- Name of M.Sc. University
- ...

Need to expand the number of features:

- **one-hot encoding** (as for multi-class)

Value 3 out of 5 outcomes $\rightarrow$ 0 0 1 0 0
Value 5 out of 5 outcomes $\rightarrow$ 0 0 0 0 1

Numerical attributes

Even numerical attributes need some **pre-processing**:

- different ranges
- different distributions

Need to encode and/or normalize:

- **uniform scaling**: $\hat{v} = \frac{v - v_{min}}{v_{max} - v_{min}} \rightarrow [0, 1]$
- **normalization**: $\hat{v} = \frac{v - \mu}{\sigma} \rightarrow$ zero mean and unit variance

To try some first experiments

A simple tool to start with ANNs:

- **pyBrain**: `pybrain.org`

A collection of datasets:

- **UCI Machine Learning Repository**: `http://archive.ics.uci.edu/ml/`