Overview

YuHua Li(李玉华)

Intelligent and Distributed Computing Lab,
Huazhong University of Science & Technology

idcliyuhua@hust.edu.cn

2019年04月09日

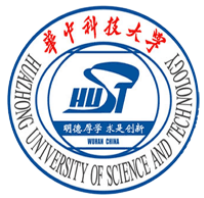
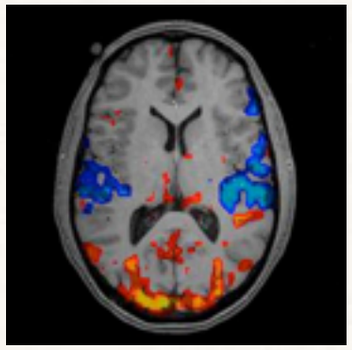


Table of contents

Table of Contents

Traditional Computer Science

Traditional CS:



Data

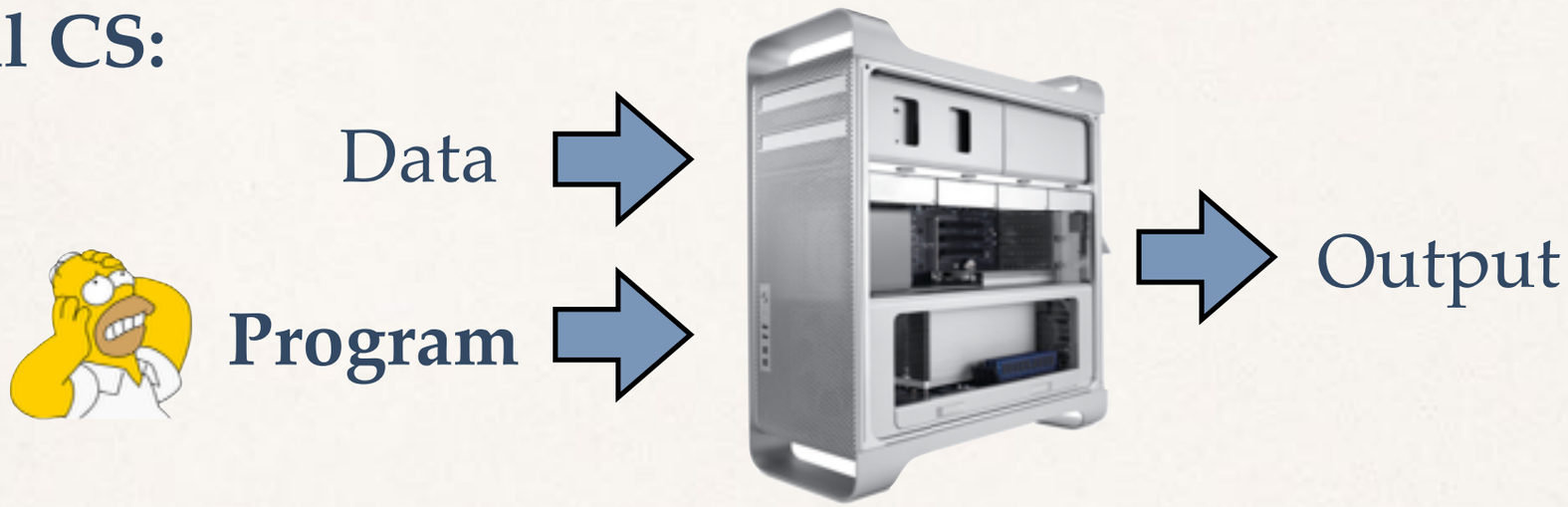
Program

Computer

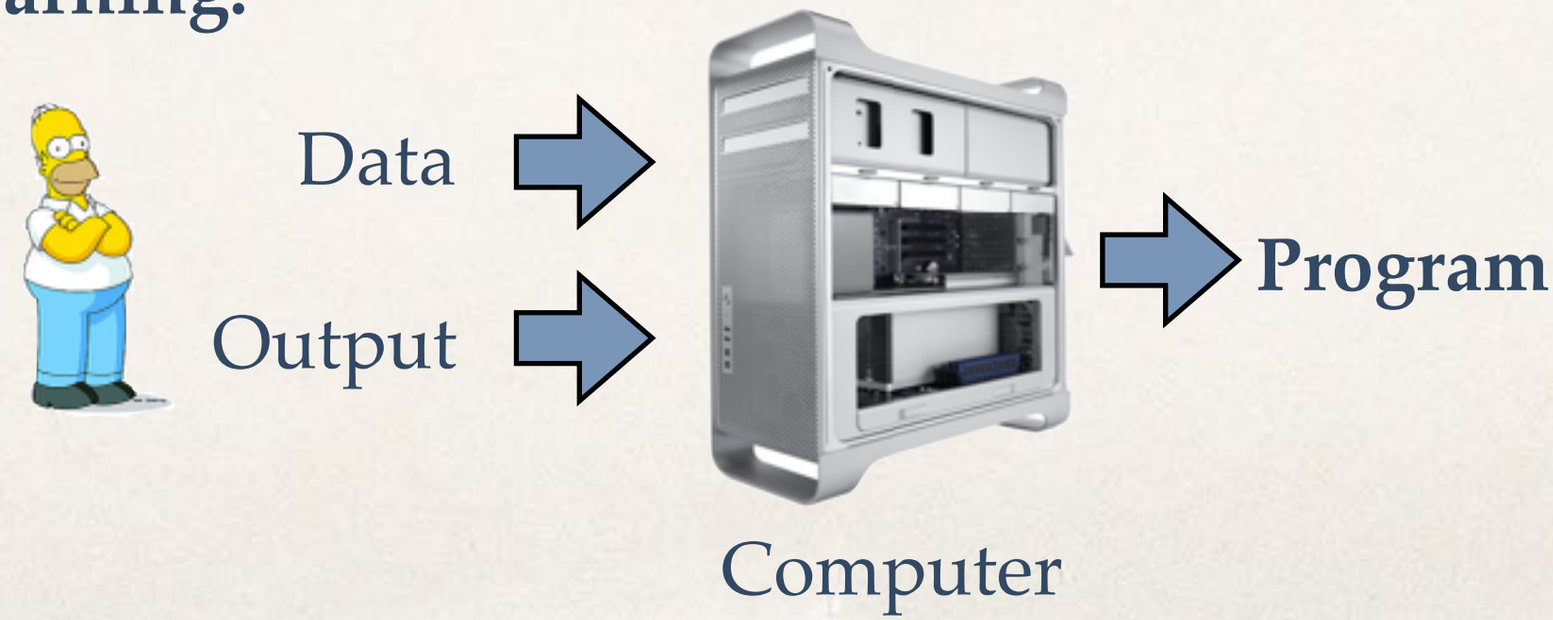
Output

Machine Learning

Traditional CS:

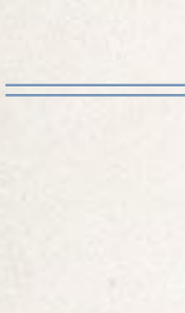


Machine Learning:



Machine Learning

Machine Learning:



Output

Data

Computer

Program

Traditional CS:

Data

Output

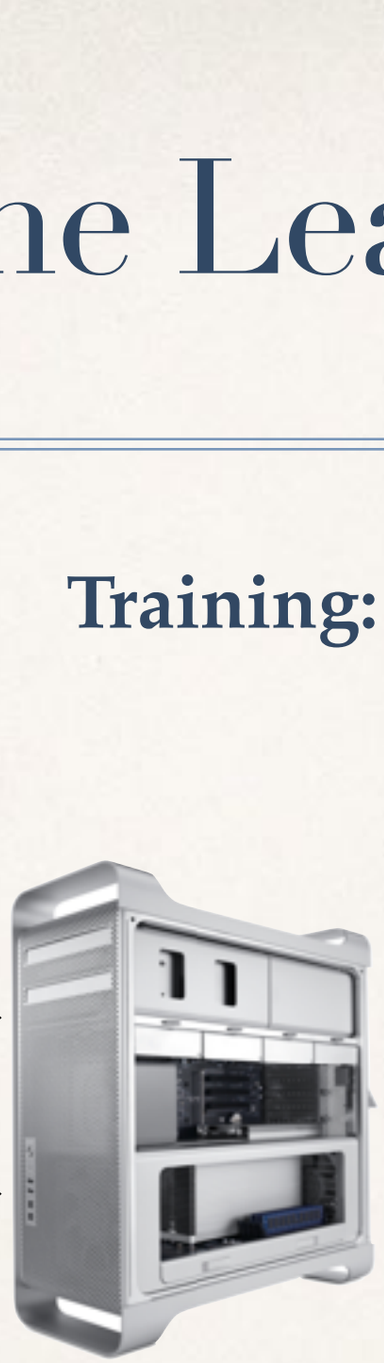
Computer

Machine Learning

Training:

Data

Output



Computer

Program

Testing:

Data

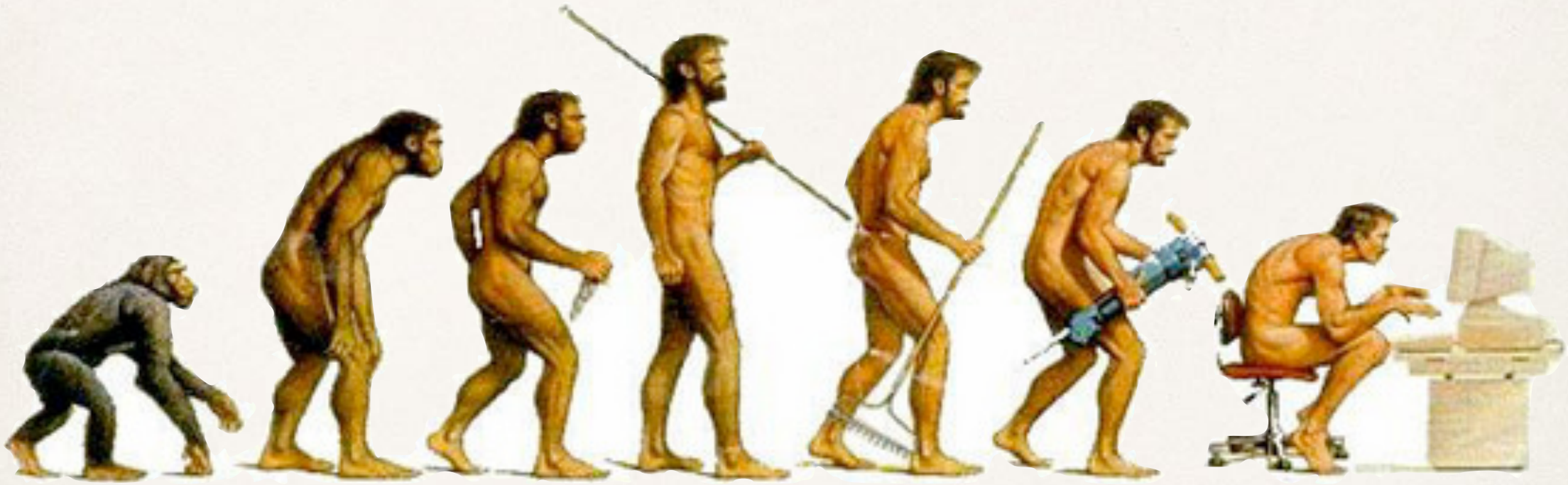
Output

Computer

What is Machine Learning?

- **Formally:** (Mitchell 1997): A computer program **A** 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**.

- **Informally:** *Algorithms* that *improve* on some *task* with *experience*.



A (very brief) History of ML

Samuel's Checker Player (1952)

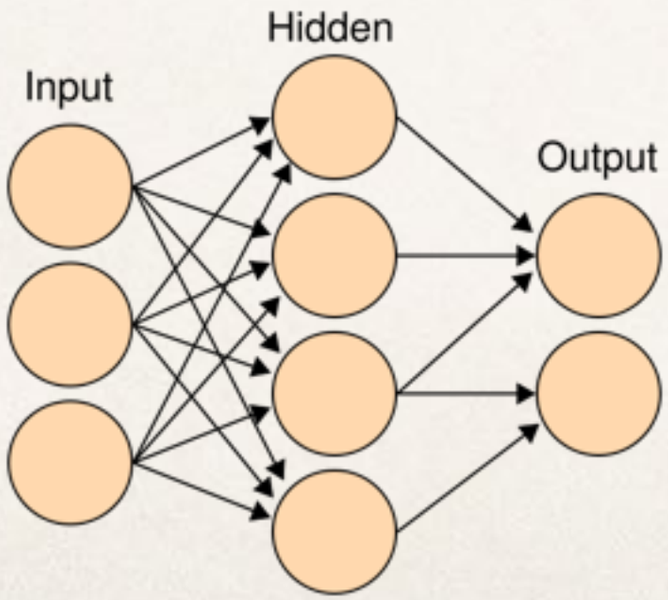
- Basically Shannon's Minimax Algorithm (traditional AI)

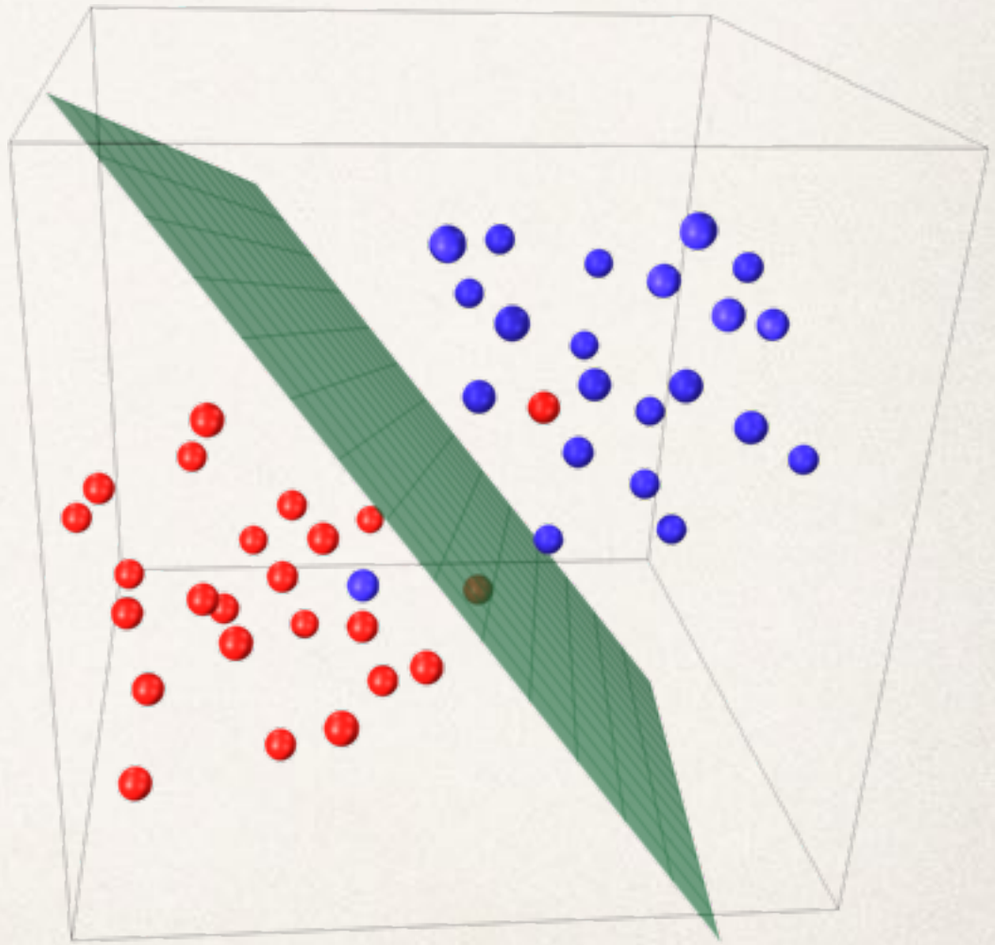
- Included simple learning algorithm to improve board evaluation.

  - Player improved over time!!

Perceptron 1957 (Frank Rosenblatt @ Cornell)

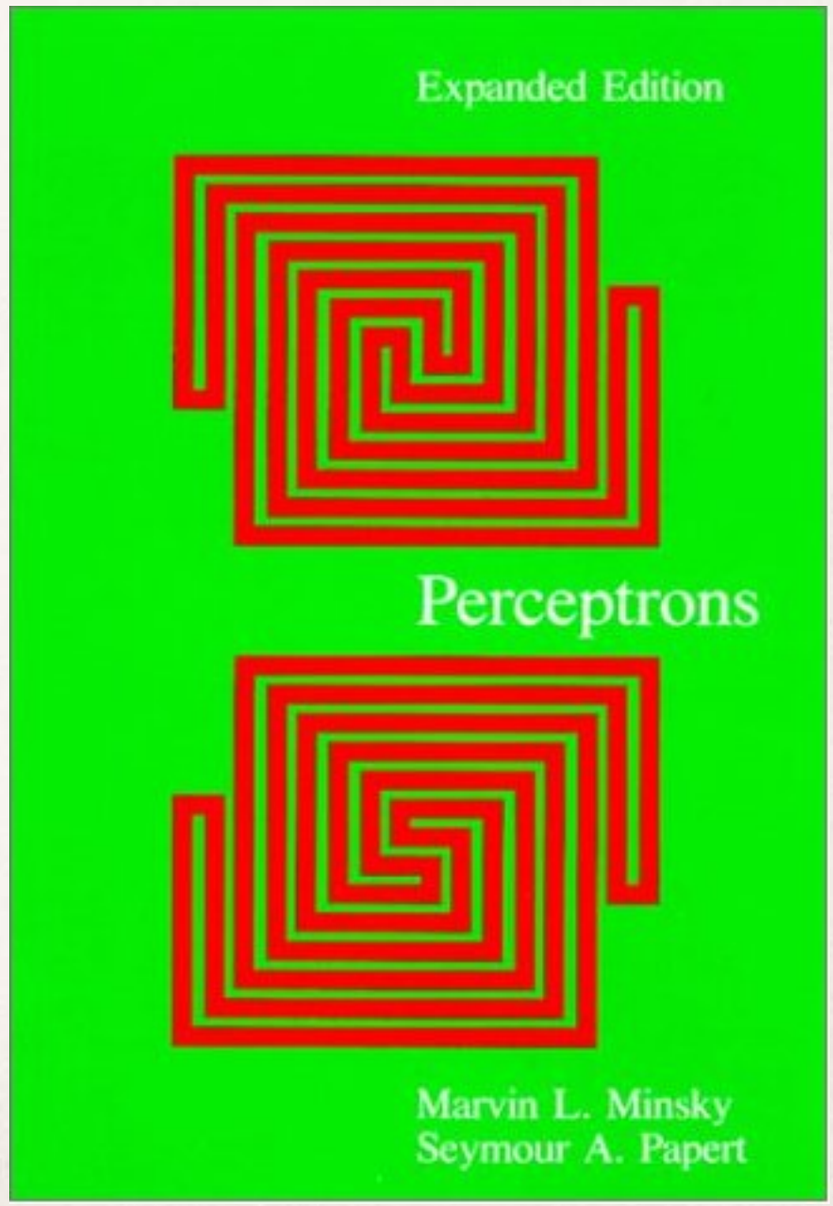
- Provable convergence properties
- Eventually leads to **multilayer perceptron = Artificial Neural Networks = Deep Learning**





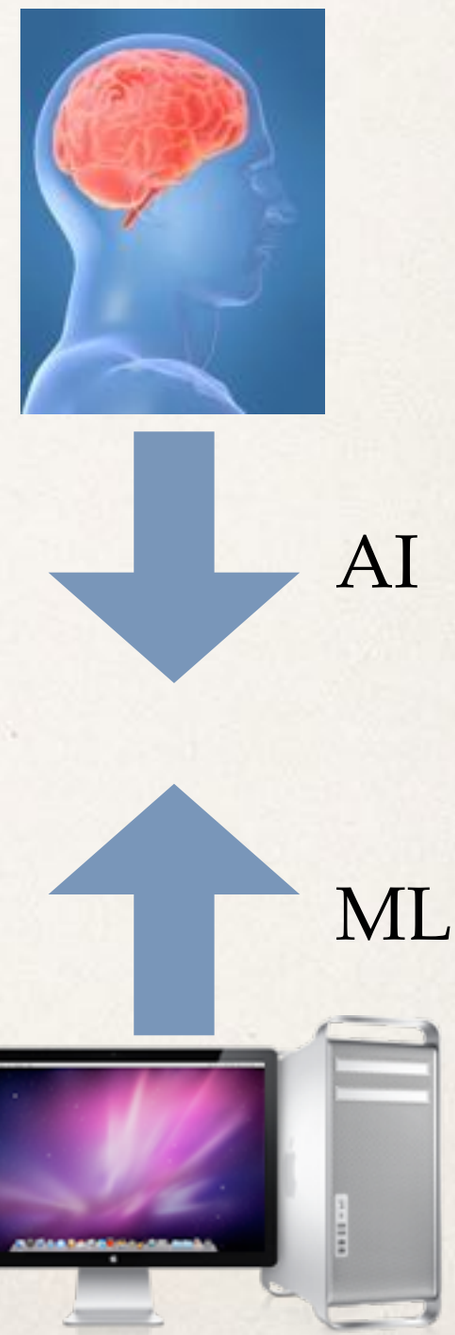
AI Winter

- (1969) Minsky & Papert “killed” AI
- Burst huge expectation bubble
- Funding for AI research collapsed for decades



Rebirth as **Machine Learning**

- Machine Learning:
  - Originally: Mostly a name game to get funding.
- Profound difference:
  - ML: Bottom up, AI: Top down
  - ML: More practical smaller goals
  - Based on **Statistics and Optimization, not Logic**



TD-Gammon (1994)

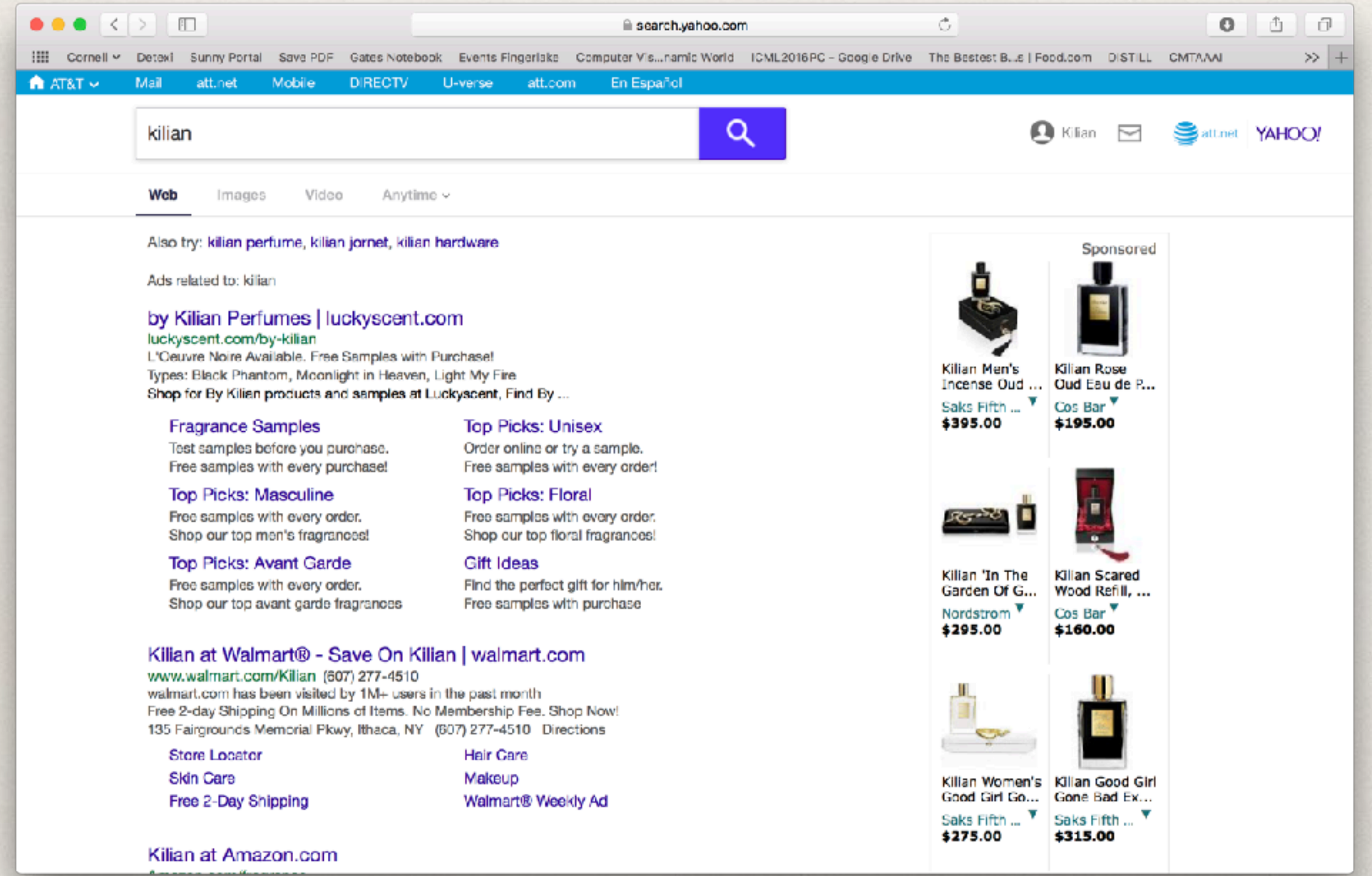
- Gerry Tesauro (IBM) teaches a neural network to play Backgammon. The net plays 100K+ games **against itself** and beats world champion [Neurocomputation 1994]

- Algorithm teaches **itself** how to play so well!!!

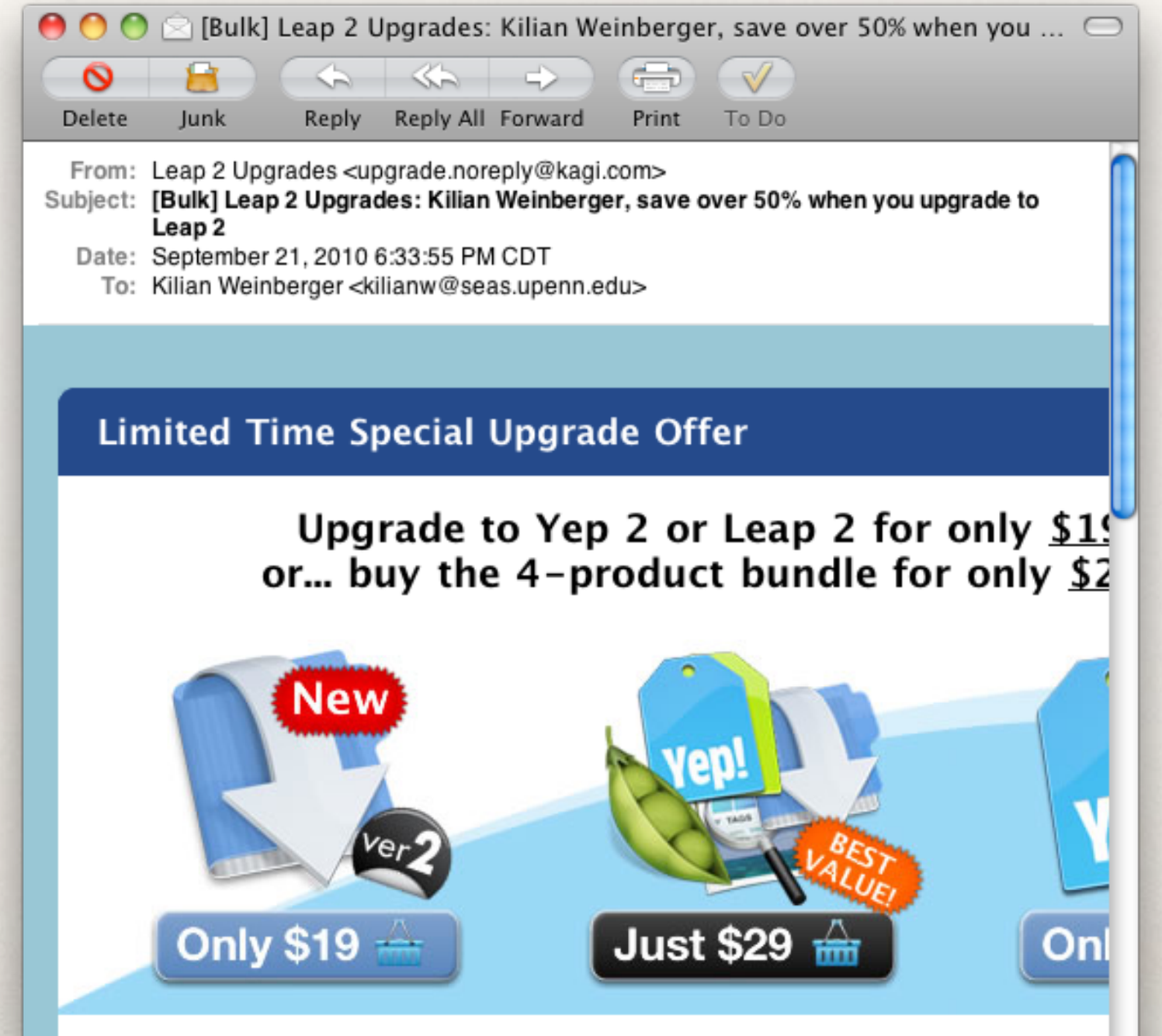
Deep Blue (1997)

- IBM's Deep Blue wins against Kasparov in chess. Crucial winning move is made due to Machine Learning (G. Tesauro).

Example: Websearch



Example: Spam Filter



Home assistant

Very soon: Autonomous Cars

When will it stop?

- **The human brain is one big learning machine**
  - We know that we can still do a **lot better!**
- However, it is hard. Very few people can design new ML algorithms.
- But many people can use them!

What types of ML are there?

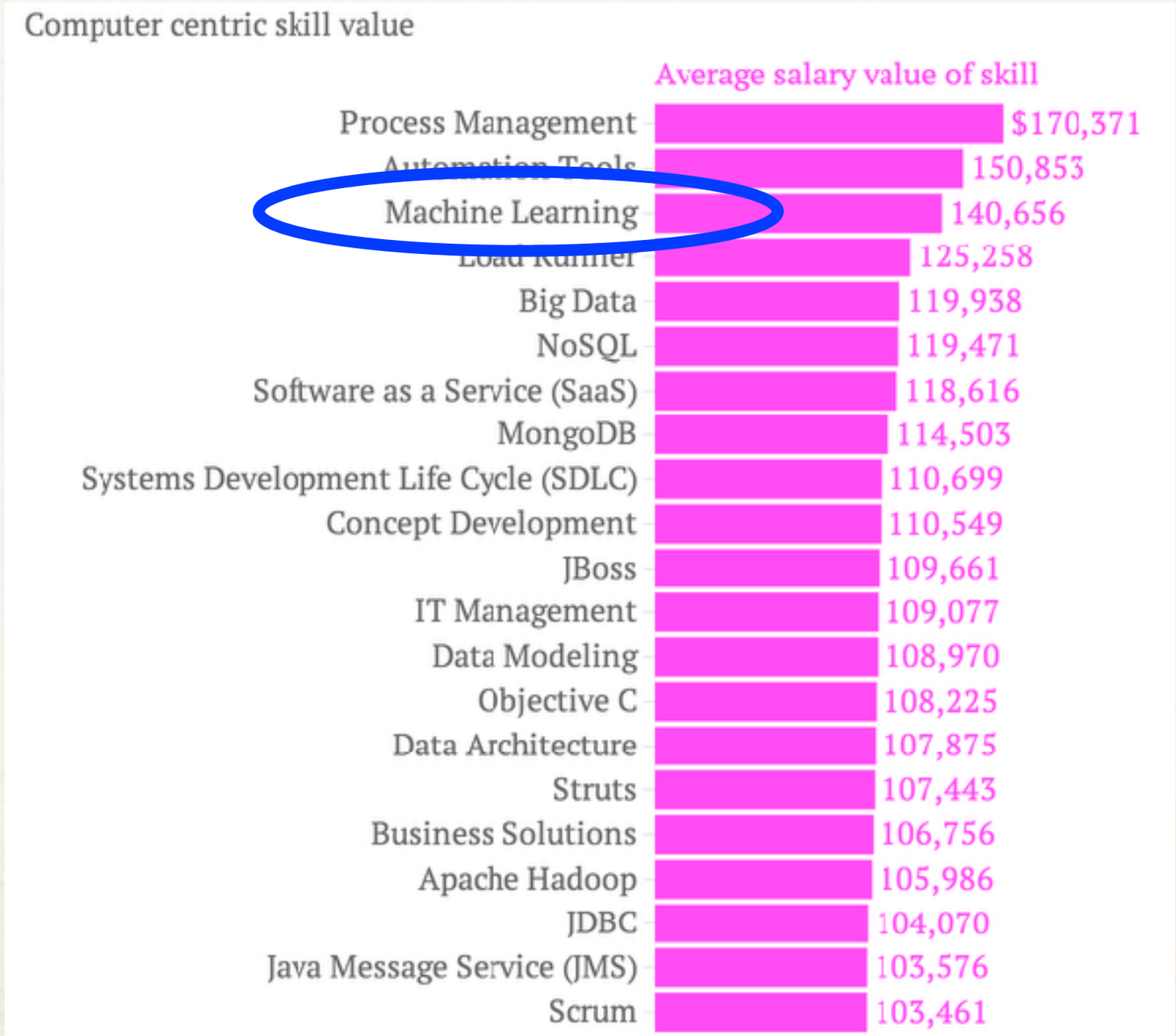
As far as this course is concerned:

- **supervised learning:** Given labeled examples, find the right prediction of an unlabeled example. (e.g. *Given annotated images learn to detect faces.***)**

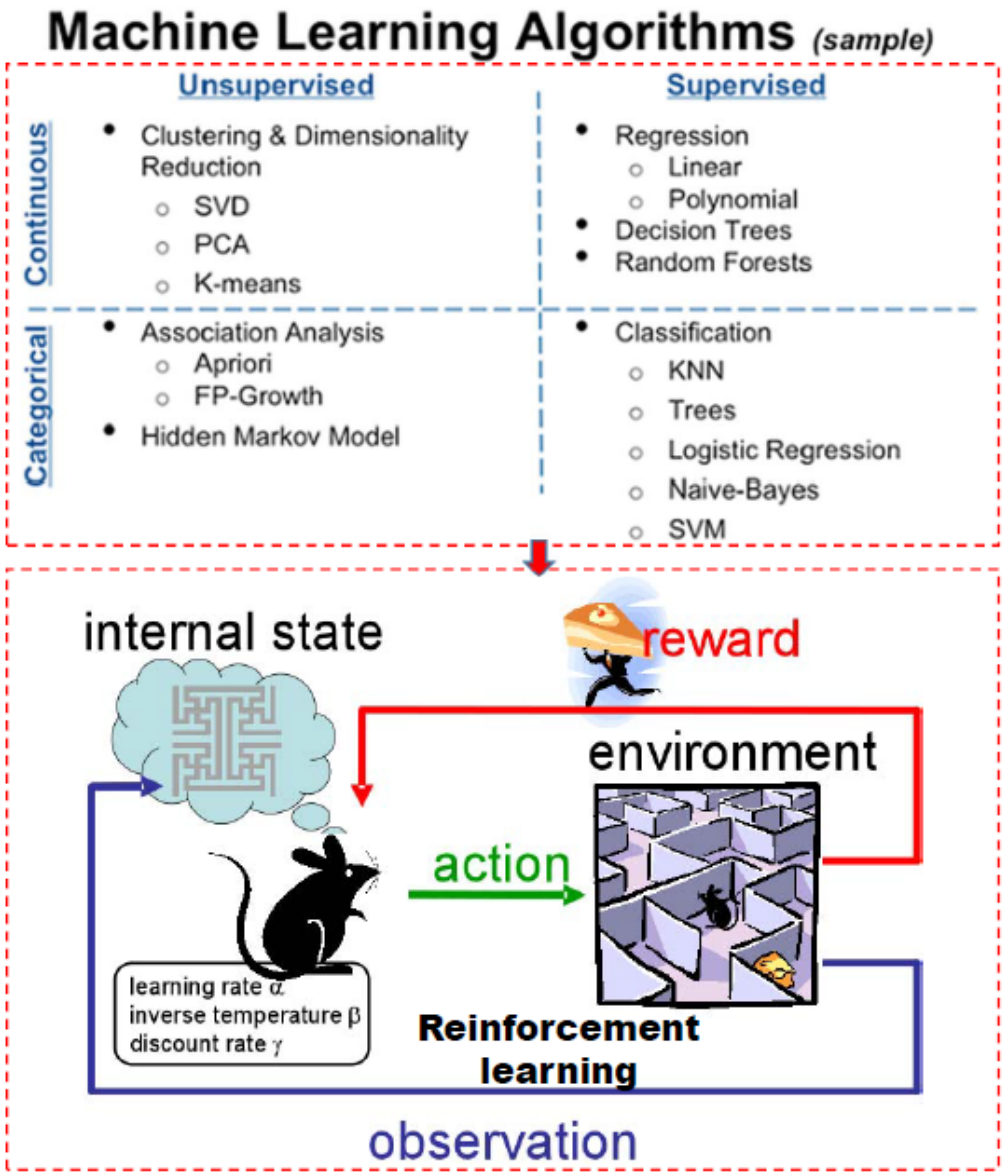
- **unsupervised learning:** Given data try to discover similar patterns, structure, sub-spaces (e.g. *automatically cluster news articles by topic*)

- **reinforcement learning:** Try to learn from delayed feedback (e.g. *robot learns to walk, fly, play chess)*

Average salary by skill



Supervised Learning VS Unsupervised Learning



History of AI

(1)1952 AI landmark: Samuel Checkers-playing Program-IBM

Herbert A. Simon

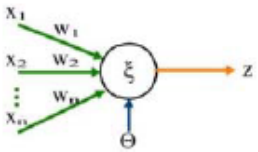
Allen Newell

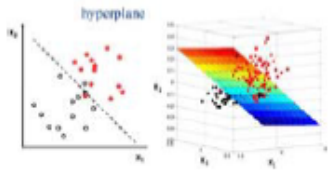
Logical Theory

Principia Mathematica

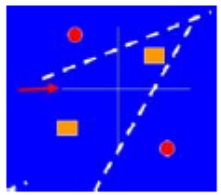
(2) perceptron (1957)

Frank Rosenblatt (1928-1971)





(3) MLP (1969) Minsky & Papert (1969)



AI winter: AI-logic(Top Down)→
ML: bottom up/Statistical optimization

(4) Edward Albert Feigenbaum(1970)

father of expert systems

(5)Enforcement learning(1994) Deep Blue

(6)Nowadays

Deep learning, Knowledge base, reinforcement learning

Table of Contents

supervised machine learning setup

Let us formalize the supervised machine learning setup. Our training data comes in pairs of inputs $(\mathbf{x}, y)$, where $\mathbf{x} \in \mathcal{R}^d$ is the input instance and y its label. The entire training data is denoted as

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{R}^d \times \mathcal{C}$$

where:

- $\mathcal{R}^d$ is the d-dimensional feature space
- $\mathbf{x}_i$ is the input vector of the i^{th} sample
- y_i is the label of the i^{th} sample
- $\mathcal{C}$ is the label space

The data points $(\mathbf{x}_i, y_i)$ are drawn from some (unknown) distribution $\mathcal{P}(X, Y)$. Ultimately we would like to learn a function h such that for a new pair $(\mathbf{x}, y) \sim \mathcal{P}$, we have $h(\mathbf{x}) = y$ with high probability (or $h(\mathbf{x}) \approx y$). We will get to this later. For now let us go through some examples of X and Y.

Examples of Label Spaces

There are multiple scenarios for the label space $\mathcal{C}$:

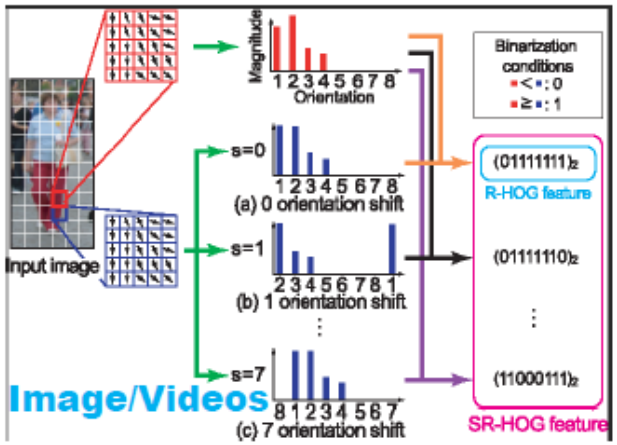
Binary classification	$\mathcal{C} = \{0, 1\}$ or $\mathcal{C} = \{-1, +1\}$.	Eg. spam filtering. An email is either spam $(+1)$, or not (-1).
Multi-class classification	$\mathcal{C} = \{1, 2, \cdots, K\}$ $(K \geq 2)$.	Eg. face classification. A person can be exactly one of K identities (e.g., 1="Barack Obama", 2="George W. Bush", etc.).
Regression	$\mathcal{C} = \mathbb{R}$	Eg. predict future temperature or the height of a person.

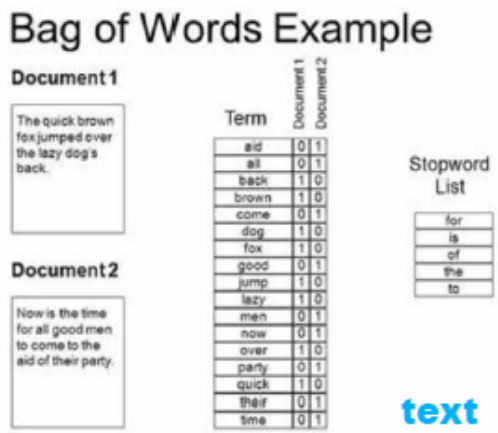
Examples of feature vectors

Hand-crafted feature

Patient Data

Patient No./ Sex/Age, y:mo	Initial Diagnosis	Injured Side	Time From Injury to Surgery, d	Ogden Fracture Type
1/M/13:8	ER	Right	7	3A
2/M/12:12	ER	Right	5	3A
3/M/12:9	ER	Left	3	2A
4/M/13:3	ER	Right	3	3A
5/M/14:4	ER	Left	3	2A
6/M/14:0	ER	Left	2	2A
7/M/15:6	OC	Right	11	2A
8/M/13:6	OC	Right	16	3A





Automatic learning feature

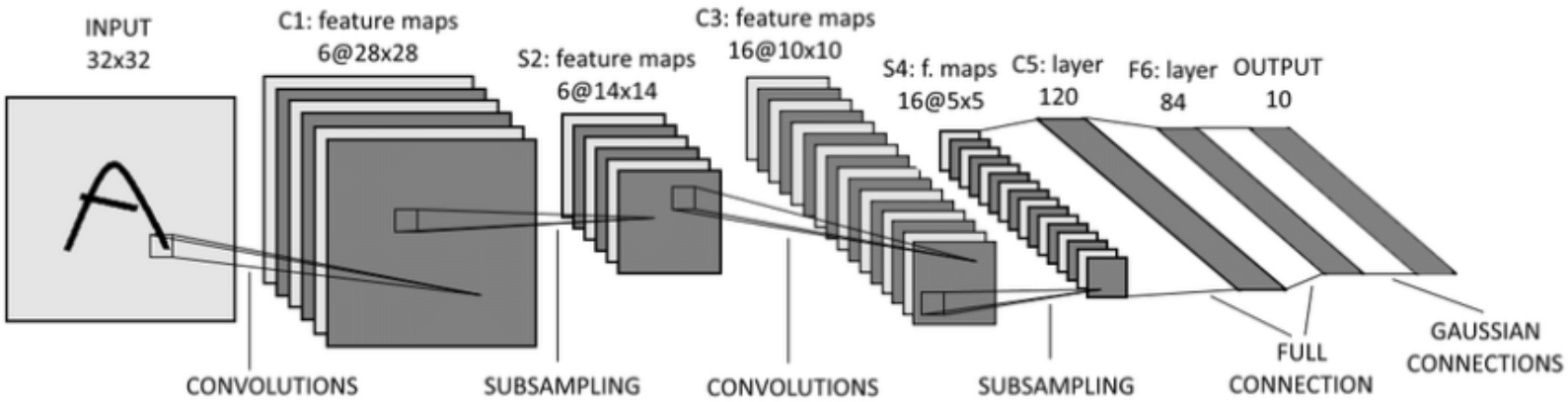


Table of Contents

What's a Loss Function

At its core, a loss function is incredibly simple: it' s a method of evaluating how well your algorithm models your dataset. If your predictions are totally off, your loss function will output a higher number. If they' re pretty good, it' ll output a lower number. As you change pieces of your algorithm to try and improve your model, your loss function will tell you if you' re getting anywhere.

What's a Loss Function

In fact, we can design our own (very) basic loss function to further explain how it works. For each prediction that we make, our loss function will simply measure the absolute difference between our prediction and the actual value. In mathematical notation, it might look something like $abs(y_predictedy)$.

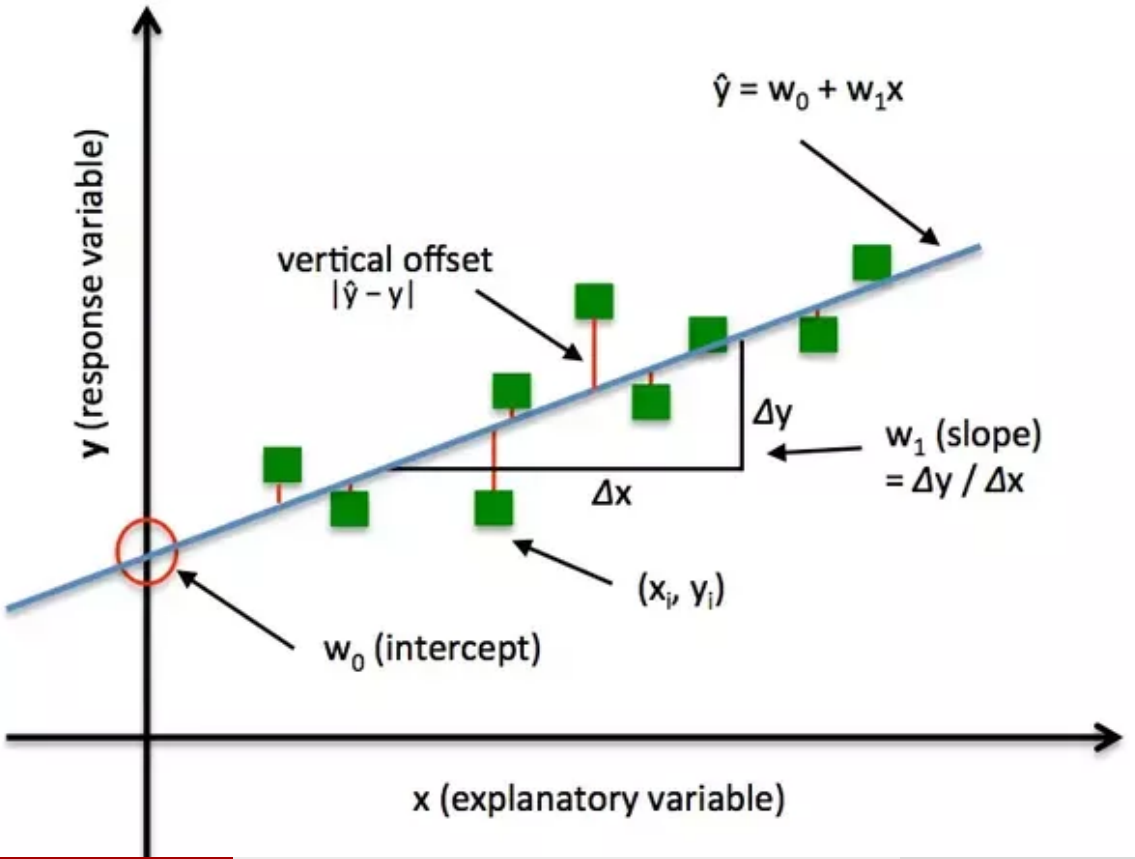
What's a Loss Function

Here's what some situations might look like if we were trying to predict how expensive the rent is in some NYC apartments:

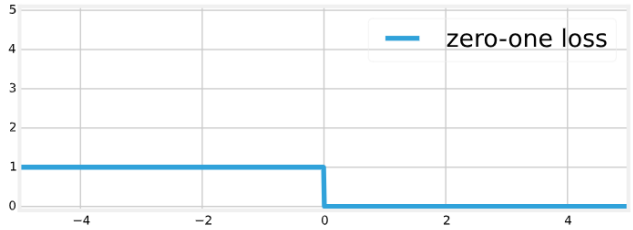
Our Predictions	Actual Values	Our Total Loss
Harlem: $1,000 SoHo: $2,000 West Village: $3,000	Harlem: $1,000 SoHo: $2,000 West Village: $3,000	0 (we got them all right!)
Harlem: $500 SoHo: $2,000 West Village: $3,000		500 (we were off by $500 in Harlem)
Harlem: $500 SoHo: $1,500 West Village: $4,000		2000 (we were off by $500 in Harlem, $500 in SoHo, and $1,000 in the West Village)

What's a Loss Function

Notice how in the loss function we defined, it doesn't matter if our predictions were too high or too low. All that matters is how incorrect we were, directionally agnostic. This is not a feature of all loss functions: in fact, your loss function will vary significantly based on the domain and unique context of the problem that you're applying machine learning to. In your project, it may be much worse to guess too high than to guess too low, and the loss function you select must reflect that.



Zero-one loss



Zero-one loss

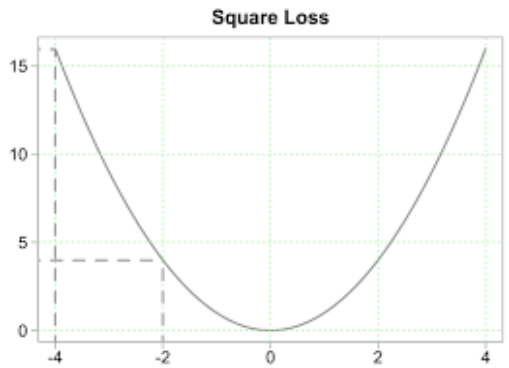
Formally, the zero-one loss can be stated has:

$$\mathcal{L}_{0/1}(h) = \frac{1}{n}\sum_{i=1}^{n} \delta_{h(\mathbf{x}_i)\neq y_i}, \text{ where } \delta_{h(\mathbf{x}_i)\neq y_i} = \begin{cases} 1, & \text{if } h(\mathbf{x}_i) \neq y_i \\ 0, & \text{o.w.} \end{cases}$$

This loss function returns the error rate on this data set D. For every example that the classifier misclassifies (i.e. gets wrong) a loss of 1 is suffered, whereas correctly classified samples lead to 0 loss.

Disadvantage: The 0-1 loss imposes the same penalty for each misclassified point, so that the points of "wrongly outrageous" (ie margin→?∞) do not receive much attention, which is not intuitively appropriate. In addition, the 0-1 loss is discontinuous, non-convex, and the optimization is difficult.

Squared loss



The squared loss function is typically used in regression settings. Formally the squared loss is:

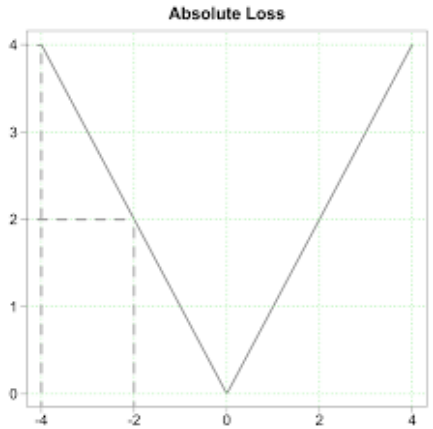
$$\mathcal{L}_{sq}(h) = \frac{1}{n}\sum_{i=1}^{n}(h(\mathbf{x}_i) - y_i)^2.$$

The squaring has two effects:

- 1., the loss suffered is always nonnegative;
- 2., the loss suffered grows quadratically with the absolute mispredicted amount.

Disadvantage: if a prediction is very close to be correct, the square will be tiny and little attention will be given to that example to obtain zero error. For example, $|h(\mathbf{x}_i) - y_i| = 0.001$

Absolute loss



Similar to the squared loss, the absolute loss function is also typically used in regression settings. It suffers the penalties $|h(\mathbf{x}_i) - y_i|$. Because the suffered loss grows linearly with the mispredictions it is more suitable for noisy data (when some mispredictions are unavoidable and shouldn't dominate the loss). If, given an input $\mathbf{x}$, the label y is probabilistic according to some distribution $P(y|\mathbf{x})$ then the optimal prediction to minimize the absolute loss is to predict the median value, i.e. $h(\mathbf{x}) = \text{MEDIAN}_{P(y|\mathbf{x})}[y]$. Formally, the absolute loss can be stated as:

$$\mathcal{L}_{abs}(h) = \frac{1}{n}\sum_{i=1}^{n}|h(\mathbf{x}_i) - y_i|.$$

Table of Contents

Generalization

Given a loss function, we can then attempt to find the function h that minimizes the loss:

$$h = \text{argmin}_{h \in \mathcal{H}} \mathcal{L}(h)$$

A big part of machine learning focuses on the question, how to do this minimization efficiently.If you find a function $h(\cdot)$ with low loss on your data D, how do you know whether it will still get examples right that are not in D?

Bad example: "memorizer" $h(\cdot)$

$$h(x) = \begin{cases} y_i, & \text{if } \exists(\mathbf{x}_i, y_i) \in D, \text{ s.t., } \mathbf{x} = \mathbf{x}_i, \\ 0, & \text{o.w.} \end{cases}$$

For this $h(\cdot)$, we get 0% error on the training data D, but does horribly with samples not in D, i.e., there's the overfitting issue with this function.

What is overfitting?

Whenever working on a data set to predict or classify a problem, we tend to find accuracy by implementing a design model on first train set, then on test set. If the accuracy is satisfactory, we tend to increase accuracy of data-sets prediction either by increasing or decreasing data feature or features selection or applying feature engineering in our machine learning model.

But sometime our model maybe giving poor result. The poor performance of our model maybe because, the model is too simple to describe the target, or may be model is too complex to express the target.

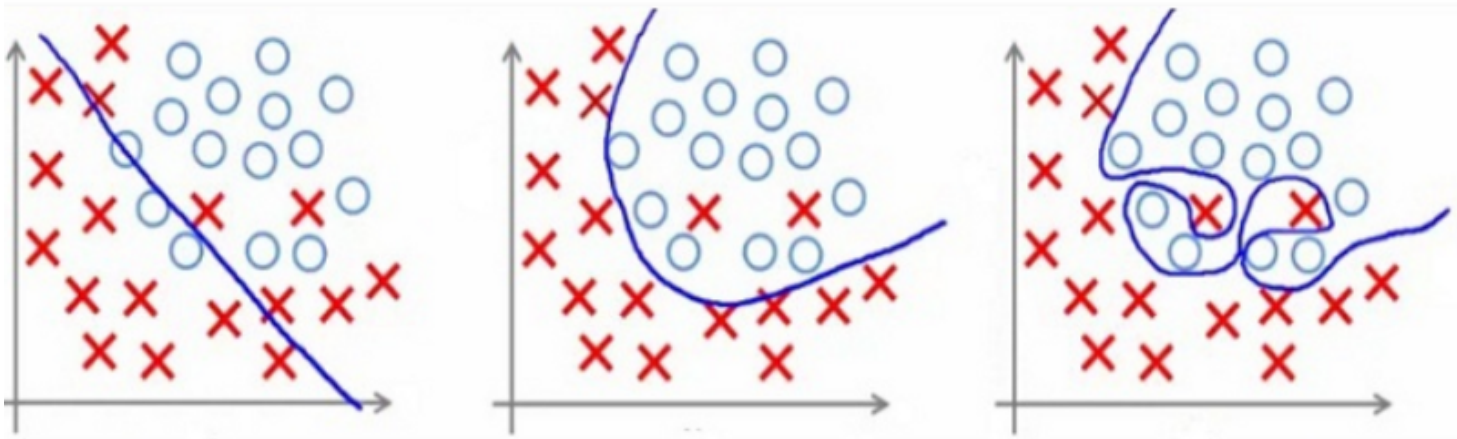
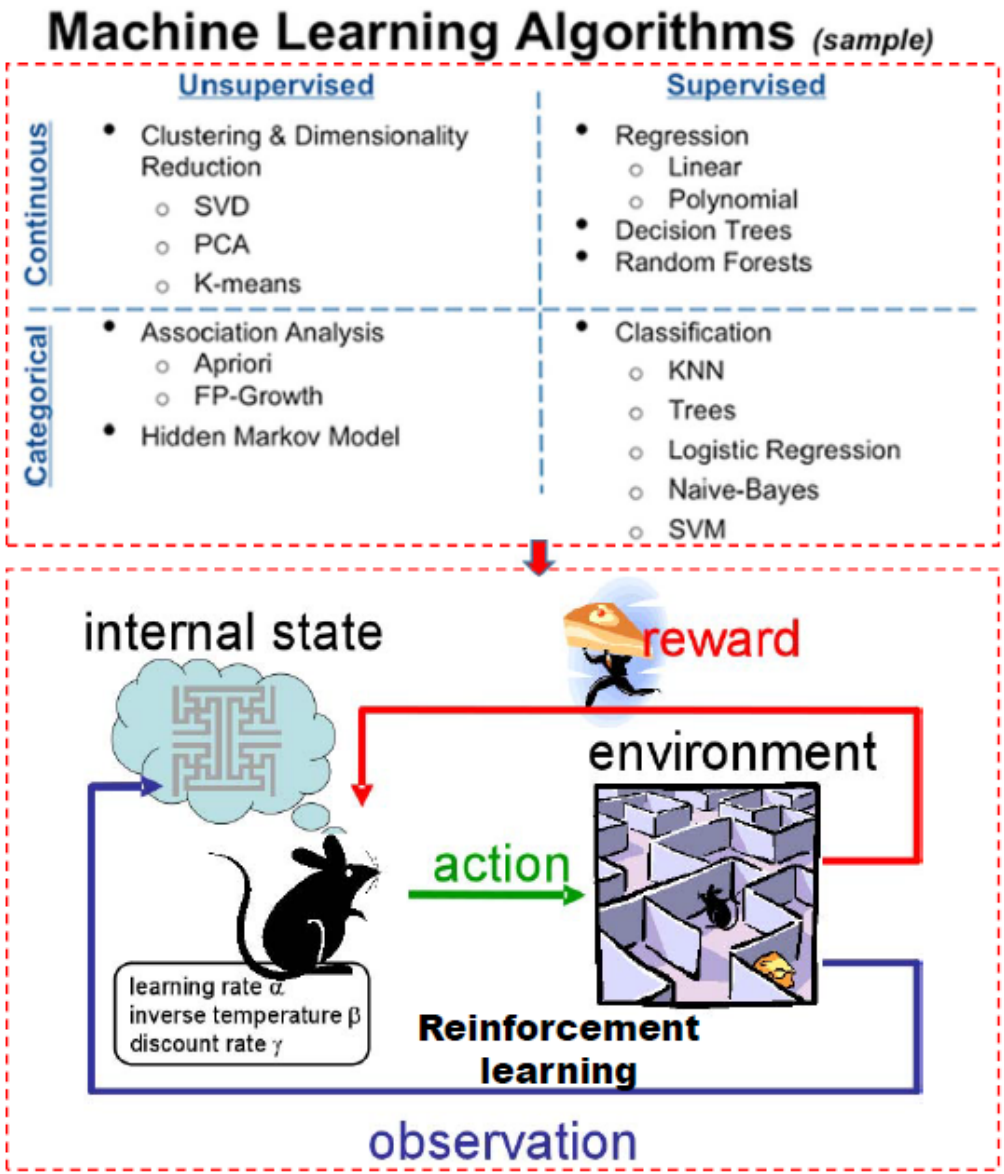


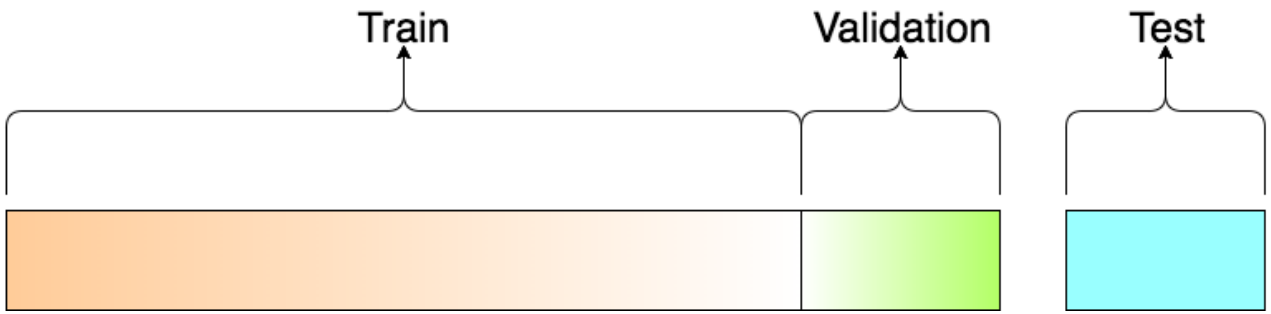
Table of Contents

Supervised Learning VS Unsupervised Learning



Train / Test splits

To resolve the overfitting issue, we usually split D into three subsets: D_{TR} as the training data, D_{VA}, as the validation data, and D_{TE}, as the test data. Usually, they are split into a proportion of 80%, 10%, and 10%. Then, we choose $h(\cdot)$ based on D_{TR}, and evaluate $h(\cdot)$ on D_{TE}.



Quiz:Why do we need D_{VA}?

D_{VA} is used to check whether the $h(\cdot)$ obtained from D_{TR} suffers from the overfitting issue. $h(\cdot)$ will need to be validated on D_{VA}, if the loss is too large, $h(\cdot)$ will get revised based on D_{TR}, and validated again on D_{VA}. This process will keep going back and forth until it gives low loss on D_{VA}. Here's a trade-off between the sizes of D_{TR} and D_{VA}: the training results will be better for a larger D_{TR}, but the validation will be more reliable (less noisy) if D_{VA} is larger.

How to Split the Data?

You have to be very careful when you split the data in Train,Validation,Test. The test set must simulate a real test scenario, i.e. you want to simulate the setting that you will encounter in real life. For example, if you want to train an email spam filter, you train a system on past data to predict if future email is spam. Here it is important to split train / test temporally - so that you strictly predict the future from the past. If there is no such thing as a temporal component, it is often best to split uniformly at random. Definitely never split alphabetically, or by feature values. By time, if the data is temporally collected. In general, if the data has a temporal component, we must split it by time. Uniformly at random, if (and, in general, only if) the data is $i.i.d.$.

The test error (or testing loss) approximates the true generalization error/loss.

Putting everything together

We train our classifier by minimizing the training loss:

$$\text{Learning: } h^*(\cdot) = \text{argmin}_{h(\cdot)\in\mathcal{H}} \frac{1}{|D_{\text{TR}}|} \sum_{(\mathbf{x},y)\in D_{\text{TR}}} \ell(\mathbf{x}, y|h(\cdot)),$$

where $\mathcal{H}$ is the hypothetical class (i.e., the set of all possible classifiers $h(\cdot)$). In other words, we are trying to find a hypothesis h which would have performed well on the past/known data. We evaluate our classifier on the testing loss:

$$\text{Evaluation: } \epsilon_{\text{TE}} = \frac{1}{|D_{TE}|} \sum_{(\mathbf{x},y)\in D_{\text{TE}}} \ell(\mathbf{x}, y|h^*(\cdot)).$$

If the samples are drawn i.i.d. from the same distribution $\mathcal{P}$, then the testing loss is an unbiased estimator of the true generalization loss:

$$\text{Generalization: } \epsilon = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}[\ell(\mathbf{x}, y|h^*(\cdot))].$$

Putting everything together

Quiz

Why does $\epsilon_{\mathrm{TE}} \to \epsilon$ as $|D_{\mathrm{TE}}| \to +\infty$? This is due to the weak law of large numbers, which says that the empirical average of data drawn from a distribution converges to its mean.

No free lunch

Every ML algorithm has to make assumptions on which hypothesis class $\mathcal{H}$ should you choose? This choice depends on the data, and encodes your assumptions about the data set/distribution $\mathcal{P}$. Clearly, there's no one perfect $\mathcal{H}$ for all problems.

Example

Assume that $(\mathbf{x}_1, y_1) = (1, 1)$, $(\mathbf{x}_2, y_2) = (2, 2)$, $(\mathbf{x}_3, y_3) = (3, 3)$, $(\mathbf{x}_4, y_4) = (4, 4)$, and $(\mathbf{x}_5, y_5) = (5, 5)$.

Question

what is the value of y if $\mathbf{x} = 2.5$? Well, it is utterly impossible to know the answer without assumptions. The most common assumption of ML algorithms is that the function to be approximated is locally smooth.

References: 1) CS4780 course of Cornell University, taught by Prof. Kilian Weinberger.
2) Prof. Kun He's teaching of machine learning at HUST basing on CS4780 of Cornell.