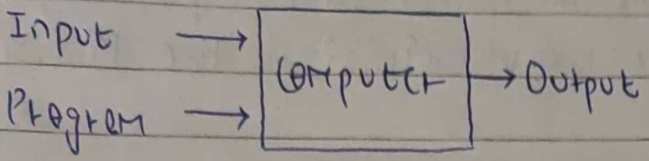
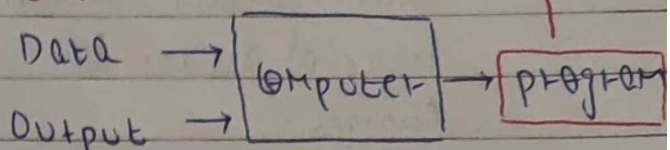
* What is Machine Learning?

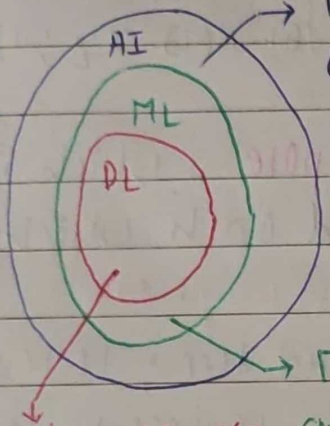


Traditional programming

This program can be just numbers, weights, biases or literally C Code



ML



More based towards logic, & the idea to create something like human. eg. Minimax algorithm resembles what a human would think, that opponent might play this best move which I can counter with this etc.

More based toward statistics, optimization

DL is when we use and pattern recognition. eg. Naive Bayes Neural networks would look into words which make an to identify pattern. email spam. Not logic, but pattern. (i.e. it is Neural Network)

* Supervised learning

statistics

distribution → probability ← Experiments

$$D = \{ (\vec{x}_1, y_1), (\vec{x}_2, y_2) \cdots (\vec{x}_n, y_n) \} \subseteq X \times Y$$

$(\vec{x}_1, y_1) \sim P$

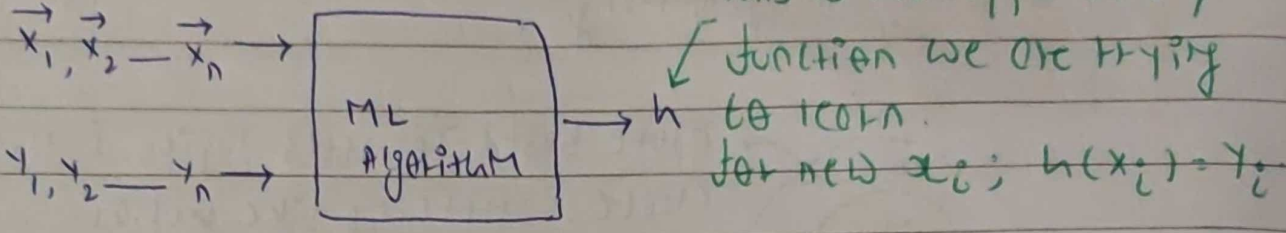
This data is sampled from prob. distribution 'P'; which is the true distribution we want to know. But we don't know. So we approximate it using ML.

$X = \mathbb{R}^d$ ['d' dimensional, where d is num of features]

Y depends on type of task

The goal is to learn a function $f(x)$, which is a close approximation of P.

It is also very imp to ensure D is obtained from some Probability distribution. Eg. for face recognition, if indian faces are used in dataset, it will work on indian faces only. Mixing a few images of african faces won't work well, since the P(x, y) changes.



H represents the hypothesis space; which is collection of all possible h. We need to find an h which is good approximation of P.

For eg. For ML algorithm as decision tree; H represents all possible decision tree, while h represents the hypothesis which works best for our task.

$h \in H$ ← Hypothesis space (to be chosen by user)

eg. of H: decision tree, neural networks, linear classifiers, SVMs.

Note: The choice of H (Hypothesis space) is to be made on how data is. The first step always is data exploration where we understand how data is distributed, related etc. Only then we choose a H of ML algorithm.

It is important that the data does have some pattern to it & is not random. Otherwise ML can not be applied. Also; each data has to have some nice properties or assumptions. It is based on these assumptions that we choose a model.

* Loss function.

To find the best h in H; we use loss function, which is just a measure of how good h is on your D.

eg. 0/1-loss

$$\hookrightarrow loss(h, D) = \frac{1}{n} \sum_{(\vec{x}_i, y_i \in D)} \delta[h(\vec{x}_i) \neq y_i]$$

lower the loss, better the h is. Also; loss function is non-negative always.

eg. Squared loss: $loss(h, D) = \frac{1}{n} \sum_{(\vec{x}_i, y_i \in D)} (h(x_i) - y_i)^2$

Absolute loss: $loss(h, D) = \frac{1}{n} \sum_{(\vec{x}_i, y_i \in D)} |h(x_i) - y_i|$

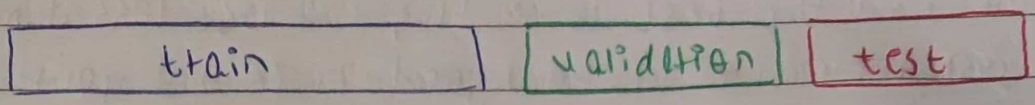
The choice of loss function highly depends on specific problem, Absolute loss is preferred when Outlier resistance, interpretability are crucial, but Squared error is preffered when Optimization is crucial.

* Generalization.

We do not want the model to memorize the D, or overfit. In other words, we want $E[loss(h, (\vec{x}, y))]$ to be less.

for any $(\vec{x}, y)$ in probability distribution P. $(\vec{x}, y) \sim P$

We ensure that with help of training, testing and validation set.



(1) If the data has a temporal component, always split the data based on time. eg. data before 2023 is train & after is test. However, if data is independently and identically distributed (iid), the the split can be done randomly.

(2) You use test set ONLY ONCE. That is the final performance of model. We can NOT tune our hyperparameters or M after looking at test accuracy. All this should be done only using validation set. Otherwise it would be overfitting to test set.

(3) The data in test should make sense with data in train i.e. P must be same.

Hence; the loss on test set gives us $\frac{1}{n}\sum_{i=1}^{n} l(h, (\vec{x}_i, y_i))$.

Now, as per weak law of large numbers, as $n \to \infty$; this represents $E[l(h, (\vec{x}, y))]$ $(\vec{x}, y) \sim p$. & evaluate.

Thus, with a big test size, we can approximate our function property.

Note: we require a balanced dataset as well.