# FIT 3181 Deep Learning

## Week 01: Mathematics and Machine Learning Revisit

**Lecturer: Dr Lim Chern Hong**

Email: lim.chernhong@monash.edu

GROUP OF EIGHT AUSTRALIA

Department of Data Science and AI
Faculty of Information Technology, Monash University, Australia

# Outline

- Unit Logistics
- Mathematical Background Revisit
- Machine Learning Revisit
- Summary

# Unit Logistics (Will go through this in the forum)

# Unit logistics

## FIT3181, 2022

- FIT 3181 Team
  - CE/Lecturer (Clayton): **Dr Trung Le**
    - Email: trunglm@monash.edu
  - Lecturer (Malaysia): **Dr Lim Chern Hong**
    - Email: lim.chernhong@monash.edu
  - Head Tutor: **Mr Thanh Nguyen**
    - Email: Thanh.Nguyen4@monash.edu
  - Tutor: **Mr James Tong**
    - Email: james.tong1@monash.edu
  - Tutor: **Mr Anh Bui (Tony)**
    - Email: tuananh.bui@monash.edu
  - Tutor: **Dr Binh Nguyen**
    - Email: binh.nguyen1@monash.edu
  - Tutor: **Mr Tuan Nguyen**
    - Email: Tuan.Ng@monash.edu
  - Tutor: **Mr Md Mohaimenuzzaman**
    - Email: md.mohaimen@monash.edu


Trung


Lim


Thanh


Tuan


James


Tony


Md Mohaimenuzzaman


Binh

# Lecture and consultation times

- Lectures for FIT 3181 will be pre-recorded
  - Please refer to the watch section in moodle
- Forum for FIT 3181 will be conducted via Zoom
  - Time: Tuesday 8:00 am – 9:00 am
- Consultation will be conducted via Zoom (by appointment)
  - Please email to your tutor with heading [FIT3181]
- Important notes:
  - There will be 12 Forums in total, each is 1 hour long.
  - There will be 12 labs in total, each is 2 hours long.

# Tutorials (Lab)

- Make sure you have been allocated to one tutorial.
  - **Three tutorial slots** per week for FIT3181
- **They will be on-campus**
  - **On-campus tutorial classes**
    - Make sure you know the tutorial room
- If you miss a tutorial and would like to join the next (same) tutorial, you must email the tutor for permission and detail to attend.

# Assessments

## Tests, Assignments and Exam

- **<u>Two</u>** in-semester online tests: **10% each**
  - <u>During</u> week **7** and week **12**
  - Approximately 1 hour long, online test, questions will be randomised each time.
- **<u>Two</u>** assignments: **20% each**
  - (Tentatively) due in week **7** and week **12** (Friday)
- **<u>One</u>** final exam (<u>closed</u> book): **40%**
  - **10** minutes reading
  - Duration: **2 hours**
- To pass this unit:
  - 45% or more in the final exam
  - 45% or more for assignments + in-semester tests.
  - An overall unit mark of 50% or more

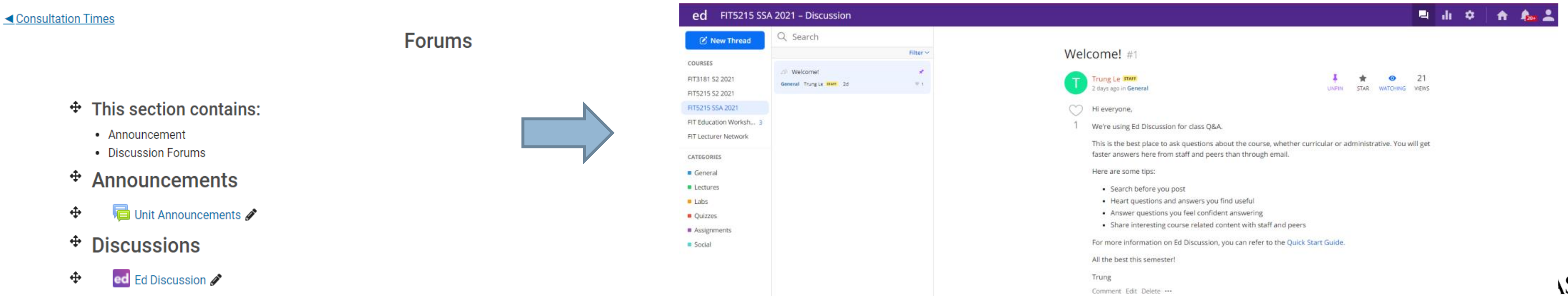
# Assessments

## Tests, Assignments and Exam

- Every student is given a total quota of 4 days for late submission
  - Use this quota wisely!
- Late submission
  - Subject to marking penalty
  - 5% deduction per day once exceeding the quota
- Apply for extension:
  - Assignment extensions are only given when applied before the due date.
  - Supporting documents are required.
  - Maximum extension is 10 days.

# Communication

- Communication will be done mainly via emails and discussion (using Ed discussion via the link from Moodle site)
  - For FIT3181: email subject must start with "FIT3181: …" and address to Dr Lim Chern Hong
    - If you don't follow this rule, your email might be missed!
  - I/we will normally answer your email within 2 working days
- Contact your lecturer, tutor for your concerns as soon as possible.
- Strongly recommend to use Ed discussion as much as possible!



# Learning objectives

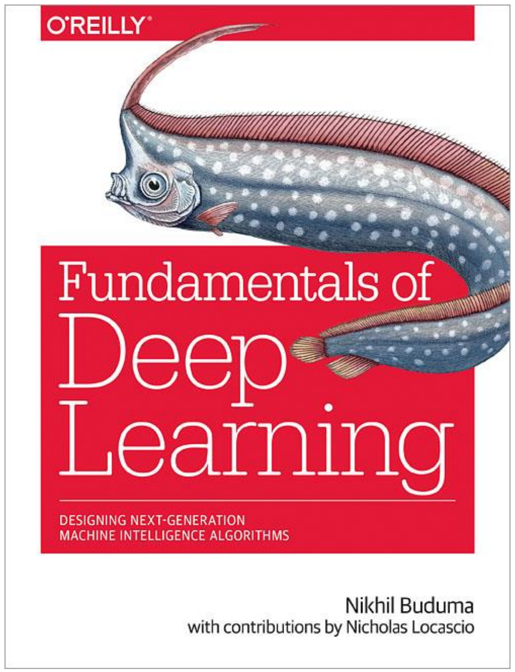
- Fundamental elements of deep learning
- Basic and advanced concepts of machine learning, AI and deep learning
- Hand-on experience with practical deep learning models and framework

# Course structure

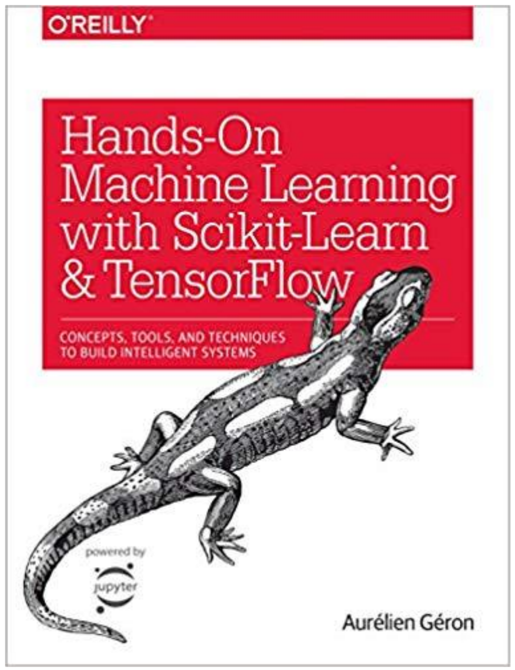
| Wk | Lectures | Assessment |
|---|---|---|
| 0 | Revision and Preparation: Students are required to install Anaconda/Tensorflow before going to the tutorial 1 | |
| 1 | Mathematics and Machine learning revisited | Assignment 1 released |
| 2 | Introduction to Deep Learning, from multilayer feedforward networks to modern deep neural networks (DNNs) | |
| 3 | Stochastic Gradient Descent and Optimization for Deep Learning | |
| 4 | Deep Learning for Vision (I): convolutional neural networks (CNNs) | |
| 5 | Training (very) deep networks:: optimizers and practical skills | |
| 6 | Deep Learning for Vision (II): network architectures, visualization, interpretability and robustness | |
| 7 | Deep learning for time-series and temporal data: RNNs and LSTMs | Online Quiz Test 1 (10%)<br>, Assignment 1 due (20%)<br>Assignment 2 released |
| 8 | Representation and deep learning for language: word2vec, Slim and Glove | |
| | **Semester Break** | |
| 9 | Seq2Seq and End-to-End Deep Models | |
| 10 | Autoencoder and Variational Autoencoder (VAE) | |
| 11 | Deep generative models and GANs | Online Quiz Test 2 (10%),<br>Assignment 2 due (20%) |
| 12 | Revision and Exam Information | |

# Unit Logistics

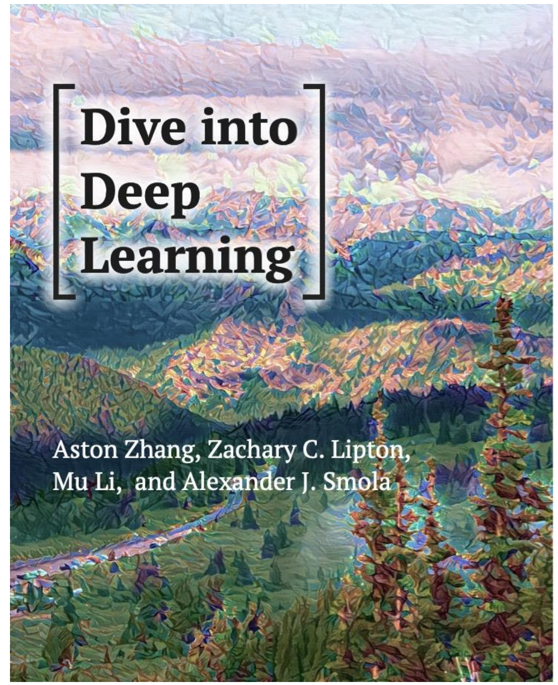
## Textbook and References



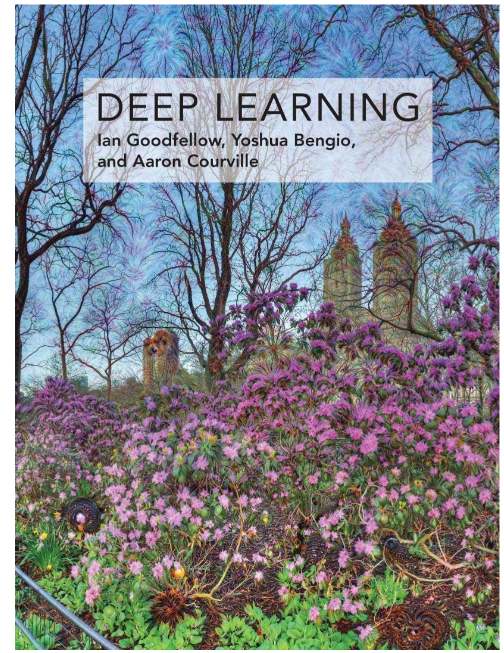

Introductory and hand-on




Intermediate and hand-on, both $1^{st}$ and $2^{nd}$ edition




Intermediate and hand-on (but don't use tensorflow)



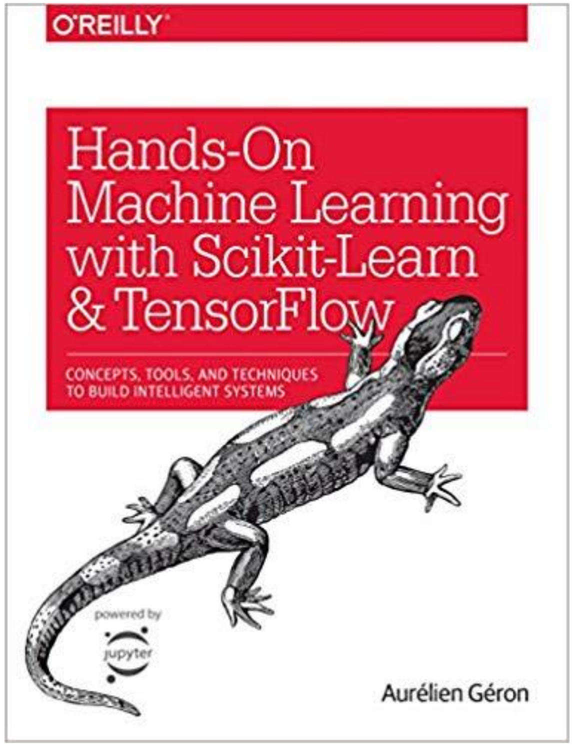

Advanced and theoretical

- No <u>single</u> textbook, some tutorial/lectures parts might use these books
- All of these books are <u>available</u> as online resources from Monash library.

# Unit Logistics
## Textbook and References

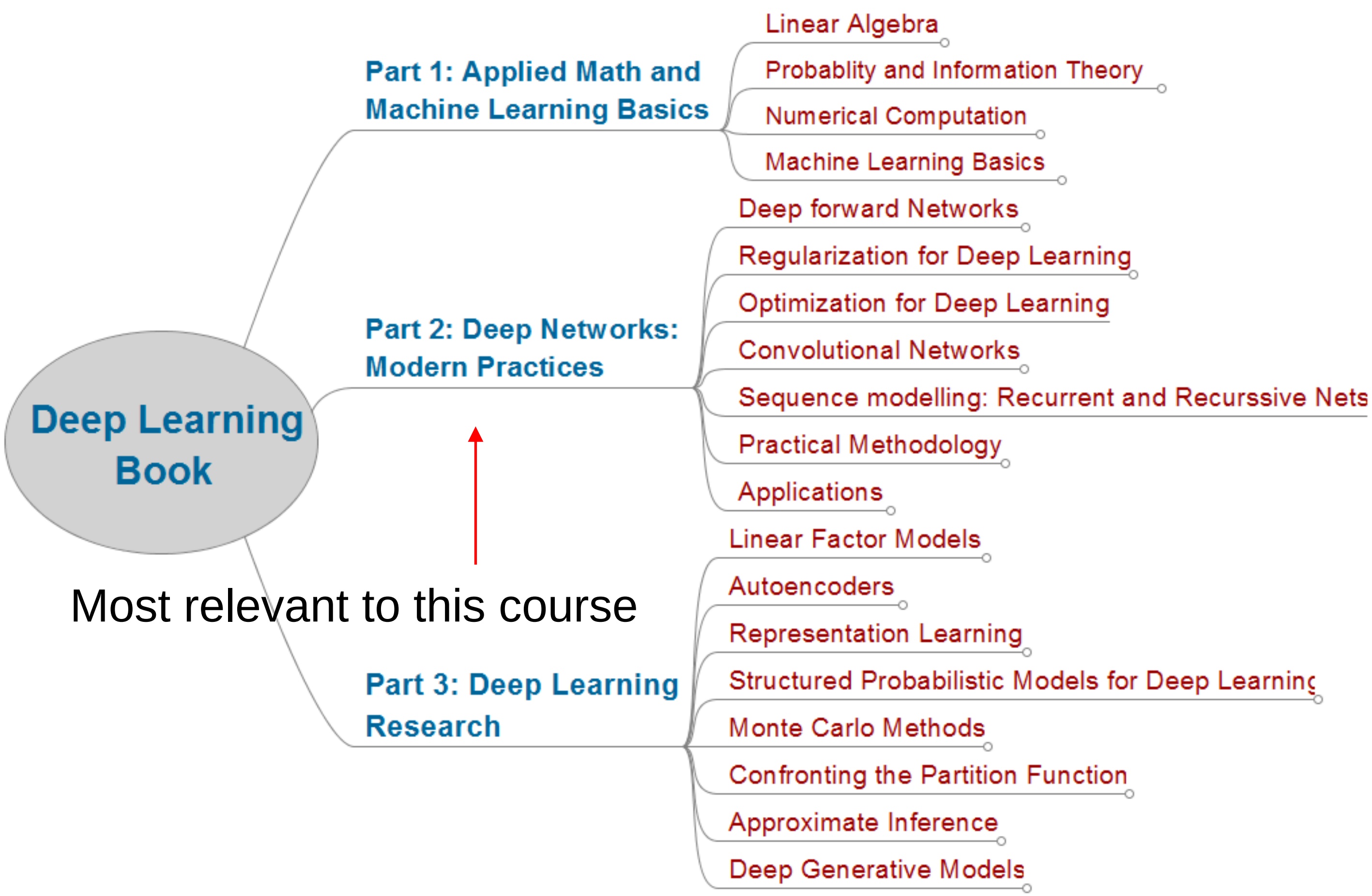
- ***Hand-on Machine Learning* Scikit-Learn and TensorFlow (Geron, 2017)**
  - **Part 1: Fundamentals of Machine Learning**
    - Machine Learning Landscape
    - End-to-End Machine Learning Project
    - Classification
    - Training Models
    - Support Vector Machines
    - Decision Trees
    - Ensemble Learning and Random Foreests
    - Dimensionality Reduction
  - **Part 2: Neural Networks and Deep Learning**
    - Up and Running with TensorFlow
    - Introduction to Artificial Neural Networks
    - Training Deep Neural Networks
    - Distributing TensorFlow Across Devices and Servers
    - Convolutional Neural Networks
    - Recurrent Neural Networks
    - Autoencoders
    - Reinforcement Learning

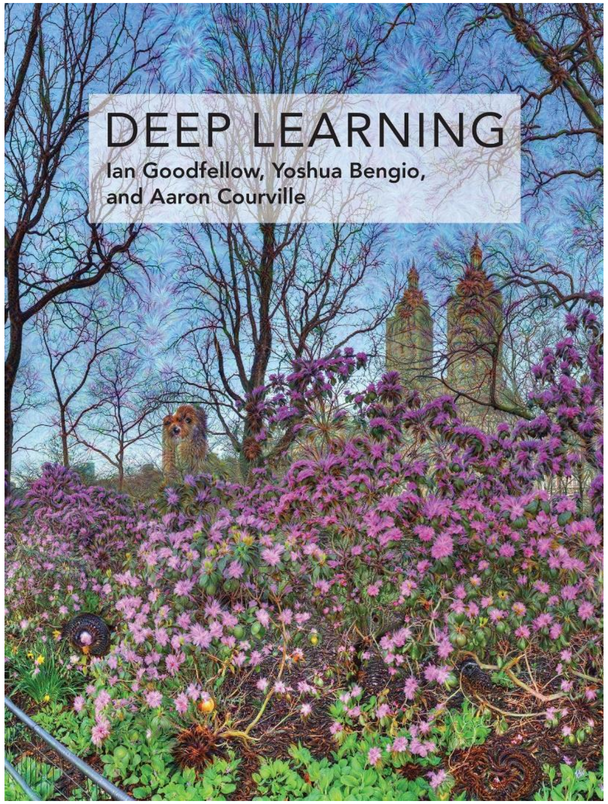
Deep learning part, using TF 2.x




Intermediate and hand-on

# Unit Logistics
## Textbook and References




Advanced and theoretical

# Unit materials

- Materials available in Moodle:
  - Unit handbook
  - Lecture slides and subsequent recorded lectures
  - News and occasional required readings
  - Tutorial and hand-on practical materials:
    - See Week 00 for preparation and revision: installation, anaconda, python, numpy, scikit-learn, etc.
    - Deep learning algorithms and models (DNN, CNNs, LSTM, Word2Vec, etc). Code is given for some of them.
    - Jupyter notebooks
    - …
  - Formal assessment instructions

# Unit Logistics

- How to succeed in this unit?
  - Work on the  hand-on tutorials and master them as quickly as you can
  - Where applicable, attempt all exercises and questions during lectures and tutorials
  - Revise key points after each lesson to prepare for the in-semester test
  - Start on your assignments early!
  - Build your knowledge systematically from lectures and recommended readings
    - If it helps, use some mindmap tool such as xmind
- How to become extra … ordinary?
  - Read and learn beyond what is being taught
  - Work on project/ideas that inspire you
  - Be proactive, github and build DL profile

# Plagiarism Notice

- Plagiarism:

  *“the practice of taking someone else's work or ideas and passing them off as one's own”* (Oxford Dictionary)

- It is a serious academic offence and you can be expelled from the University!

# Mathematical Background Revisited

# Linear algebra revision

# Why do we need vectors? (Forum discussion)

## Hand-crafted feature

- **Traditional approach** (hand-crafted feature learning)

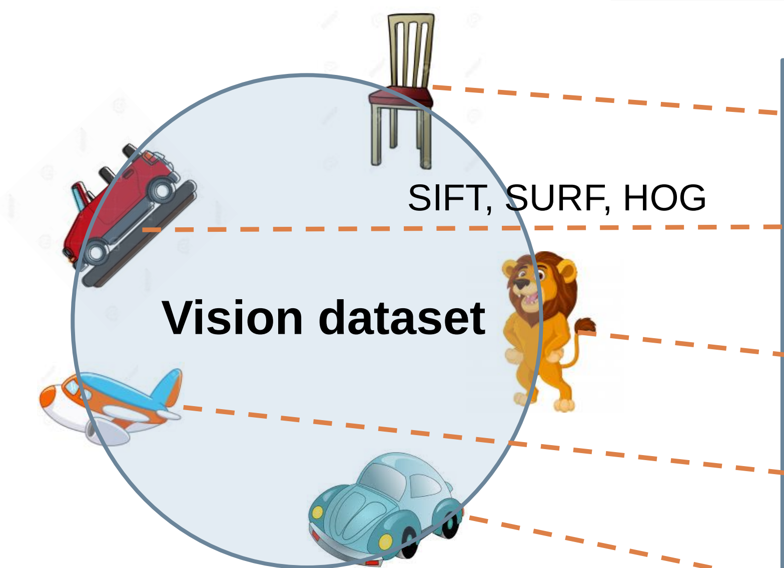
**Input**

**Feature extraction** (SIFT, SURF, HOG)

**Hand-crafted features**

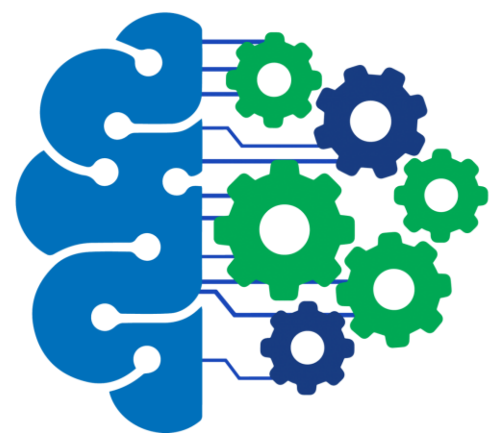
**Learning algorithms** (SVM, XGBoost)

[Source: Internet]




| | | | | |
|---|---|---|---|---|
| 0.1 | 0.5 | -1 | 2.1 | 1.3 |
| 0.6 | 0.2 | 1.3 | 2.5 | 3.3 |
| 0.7 | 0.5 | -2 | 2.2 | 2.3 |
| 0.8 | 0.4 | 0.9 | 2.4 | 1.2 |
| 0.6 | 0.9 | 1.5 | 2.0 | 1.6 |

**Feature vectors**



**Prediction**

**ML model**
**(SVM, k-NN, decision tree, XG Boost)**

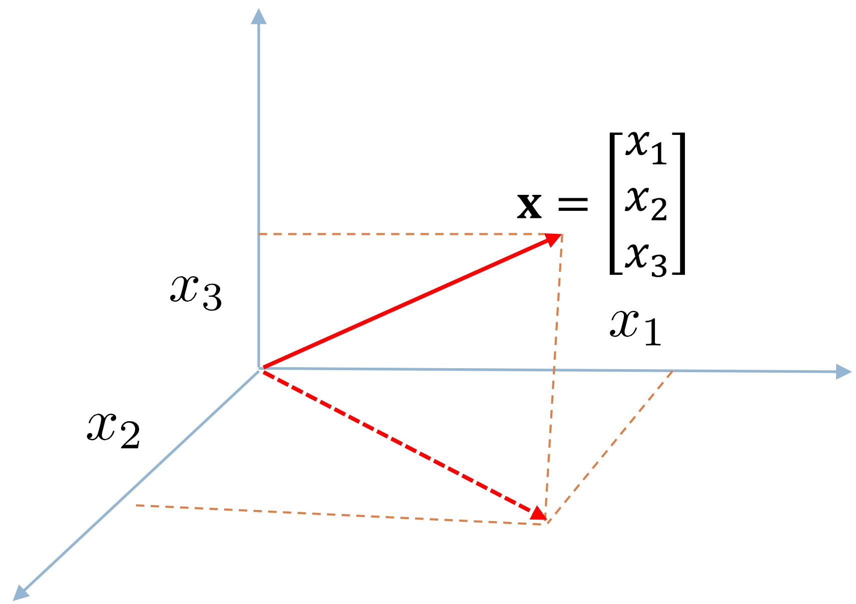
# Vector

## n-dimensional vector

dimension

$$\mathbf{x} \in \mathbb{R}^n, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$x_i$ : $i$-th element



# Vector

## Operations on vector

transpose: column vector to row vector

$$\mathbf{x}^{\mathsf{T}} = [x_1 \; x_2 \; \dots \; x_n] \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

addition

$$\mathbf{x}^{\mathsf{T}}\mathbf{y} = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n$$

inner product

# Vector

## p-norm

$$||\mathbf{x}||_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}} \text{ with } p \geq 0$$

- p = 0 : how many elements in x are non-zeros (sparsity)
- p = 1 : sum of absolute values of elements
- p = 2 : length of the vector

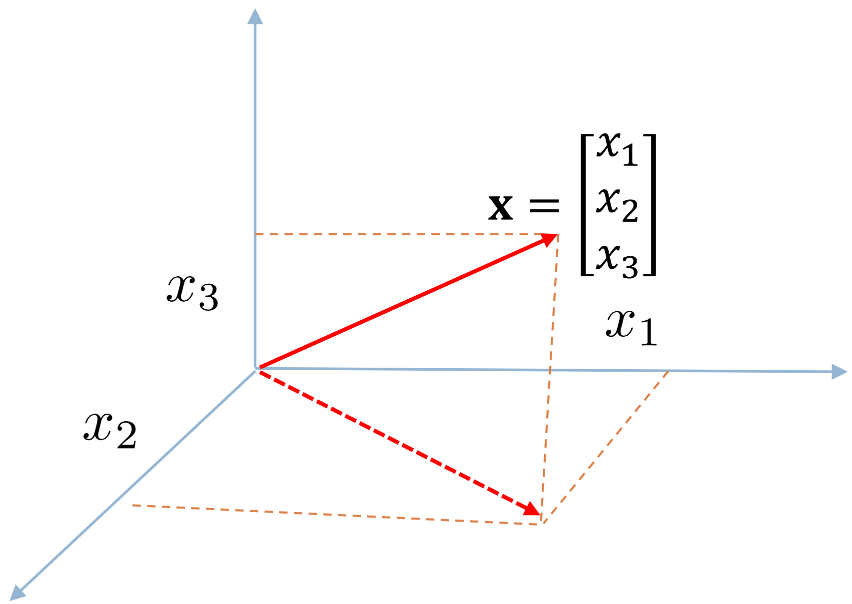
- $||\mathbf{x}||_p \geq 0$ and $||\mathbf{x}||_p = 0$ if only if $\mathbf{x} = \mathbf{0}$.
- $||\mathrm{k}\mathbf{x}||_p = |k|||\mathbf{x}||_p$ for any scalar $k \in \mathbb{R}$.
- $||\mathbf{x} + \boldsymbol{y}||_p \leq ||\mathbf{x}||_p + ||\boldsymbol{y}||_p$ (triangle inequality).

# Vector

## Length of a vector

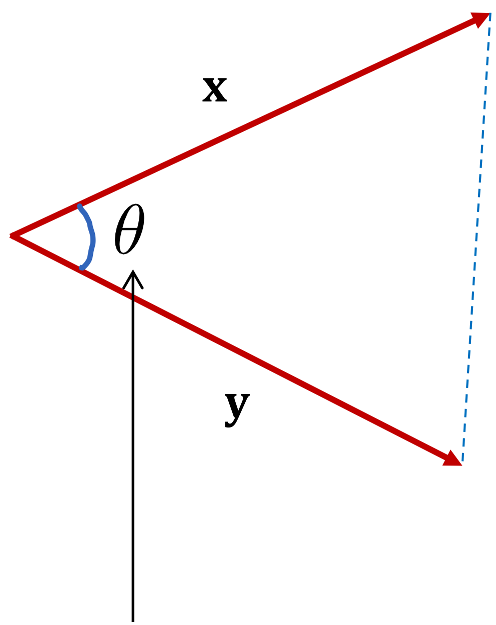
$$\text{length}(\mathbf{x}) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$



# Vector

## Distance between two vectors




Euclidean distance

$$||\mathbf{x}-\boldsymbol{y}|| = \sqrt{(x_1-y_1)^2+\cdots+(x_n-y_n)^2}$$

Cosine similarity

$$\cos(\theta) = \cos(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^T\mathbf{y}}{||\mathbf{x}||\,||\mathbf{y}||} = \frac{\mathbf{x}^T\mathbf{y}}{\sqrt{\mathbf{x}^T\mathbf{x}}\sqrt{\mathbf{y}^T\mathbf{y}}}$$

$$-1 \le \cos(\mathbf{x},\mathbf{y}) \le 1$$

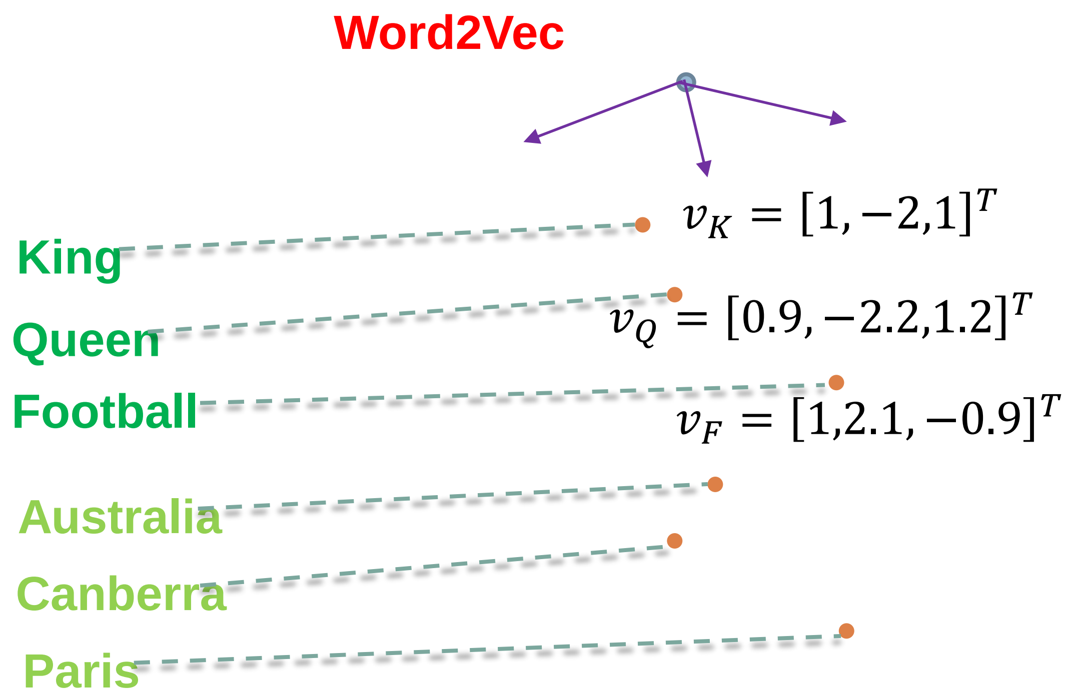
Cosine distance is defined as

$$d_{cosine}(\mathbf{x},\mathbf{y}) = 1 - \cos(\mathbf{x},\mathbf{y})$$

Cosine distance can be computed via Euclidean distance if vectors are made unit vectors! (why?)

# Example of cosine similarity (Forum discussion)

- Assume that a **Word2Vec model** transforms three words **King**, **Queen**, and **Football** to three vectors $v_K, v_Q$, and $v_F$ as shown in the figure. Between two words King and Football, which word is **more similar** to word Queen?




- $cos\_sim(King, Queen) = \frac{v_K^T v_Q}{\|v_K\|\|v_Q\|} = \frac{1\times0.9+(-2)\times(-2.2)+1\times1.2}{\sqrt{1^2+(-2)^2+1^2}\sqrt{0.9^2+(-2.2)^2+1.2^2}} = 0.996$
- $cos\_sim(Football, Queen) = \frac{v_F^T v_Q}{\|v_F\|\|v_Q\|} = \frac{1\times0.9+2.1\times(-2.2)+(-0.9)\times1.2}{\sqrt{1^2+2.1^2+(-0.9)^2}\sqrt{0.9^2+(-2.2)^2+1.2^2}} = -0.72$
- King is **more similar** to Queen than Football.

# Matrix (2D tensor)

number of columns

$$A \in \mathbb{R}^{m \times n}$$

number of rows

number of columns

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{in} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

number of rows

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{in} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

row vector

column vector

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & \cdots & a_{nj} & \cdots & a_{mn} \end{bmatrix}$$

# Matrix operation

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a_{11}} & \cdots & \boldsymbol{a_{1n}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{i1}} & \dots & \boldsymbol{a_{in}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{m1}} & \cdots & \boldsymbol{a_{mn}} \end{bmatrix} \in \mathbb{R}^{\boldsymbol{m \times n}}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{b_{11}} & \cdots & \boldsymbol{b_{1n}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{b_{i1}} & \dots & \boldsymbol{b_{in}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{b_{m1}} & \cdots & \boldsymbol{b_{mn}} \end{bmatrix} \in \mathbb{R}^{\boldsymbol{m \times n}}$$

$$\boldsymbol{C} = \boldsymbol{A^T} = \begin{bmatrix} \boldsymbol{a_{11}} & \cdots & \boldsymbol{a_{m1}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{1j}} & \dots & \boldsymbol{a_{mj}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{1n}} & \cdots & \boldsymbol{a_{mn}} \end{bmatrix} \in \mathbb{R}^{\boldsymbol{n \times m}}$$

$\boldsymbol{C} = \boldsymbol{A^T}$ if only if $\boldsymbol{C_{ij}} = \boldsymbol{A_{ji}}$ for any $1 \leq i \leq n, 1 \leq j \leq m$

$$\boldsymbol{\alpha A} = \begin{bmatrix} \boldsymbol{\alpha a_{11}} & \cdots & \boldsymbol{\alpha a_{1n}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{\alpha a_{i1}} & \dots & \boldsymbol{\alpha a_{in}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{\alpha a_{m1}} & \cdots & \boldsymbol{\alpha a_{mn}} \end{bmatrix} \in \mathbb{R}^{\boldsymbol{m \times n}}$$

$$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} \boldsymbol{a_{11}} + \boldsymbol{b_{11}} & \cdots & \boldsymbol{a_{1n}} + \boldsymbol{b_{1n}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{i1}} + \boldsymbol{b_{i1}} & \dots & \boldsymbol{a_{in}} + \boldsymbol{b_{in}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{m1}} + \boldsymbol{b_{m1}} & \cdots & \boldsymbol{a_{mn}} + \boldsymbol{b_{mn}} \end{bmatrix} \in \mathbb{R}^{\boldsymbol{m \times n}}$$

# Matrix multiplication

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ \boxed{a_{i1}} & \boxed{\dots} & \boxed{a_{in}} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & \dots & \boxed{b_{1j}} & \dots & b_{1p} \\ \dots & \dots & \boxed{\dots} & \dots & \dots \\ b_{n1} & \dots & \boxed{b_{nj}} & \dots & b_{np} \end{bmatrix}$$

$$c_{ij} = A[i,:]B[:,j] = \sum_{k=1}^{n} a_{ik}b_{kj}$$

- Given **two matrices** $A = [a_{ij}] \in \mathbb{R}^{m\times n}$ and $B = [b_{ij}] \in \mathbb{R}^{n\times p}$, we can compute the **multiplication** $C = AB$ of $A$ and $B$ as
  - $C = [c_{ij}] \in \mathbb{R}^{m\times p}$
  - $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$ for $1 \le i \le m$ and $1 \le j \le p$
  - $c_{ij} = A[i,:]B[:,j]$ which is the **dot product** of the **row *i* in A** and the **column *j* in B**.

# Matrix multiplication with numpy

```
A = np.array([
    [10,20,30],
    [40,50,60]
])
A
```

```
array([[10, 20, 30],
       [40, 50, 60]])
```

**Declare matrix $A$ with the size $2 \times 3$**

```
D = np.array([
        [ 2,  3,  5,  7],
        [11, 13, 17, 19],
        [23, 29, 31, 37]
    ])
D
```

```
array([[ 2,  3,  5,  7],
       [11, 13, 17, 19],
       [23, 29, 31, 37]])
```

**Declare matrix $D$ with the size $3 \times 4$**

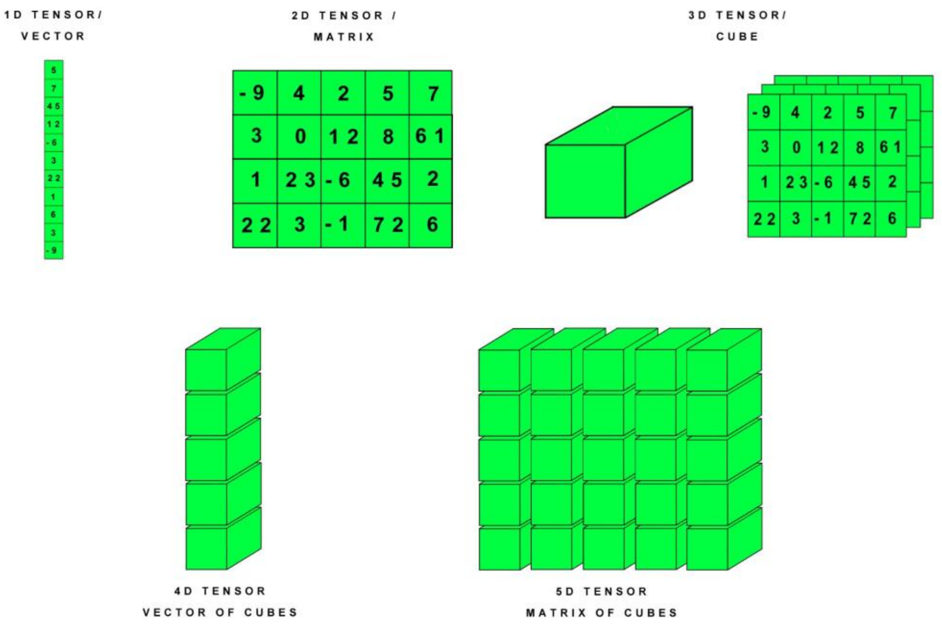
```
E = A.dot(D)
E
```

```
array([[ 930, 1160, 1320, 1560],
       [2010, 2510, 2910, 3450]])
```

**$E = A \times D$ has the size $2 \times 4$**

# Tensor

- A **tensor** is simply a **multidimensional** array
  - 0D tensor: a scalar
  - 1D tensor: a vector
  - 2D tensor: a matrix
  - 3D tensor: a 3D hyperrectangle of numerals




Examples of tensors. [Source: Internet]

# Calculus revision

# A small detour to calculus

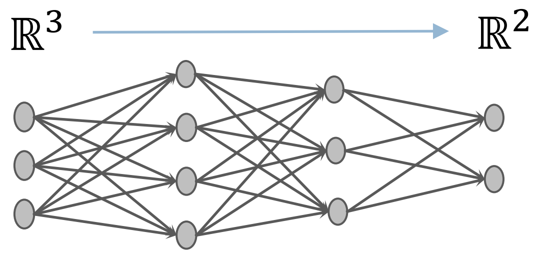
- Calculus = **mathematics of change** (very important for deep learning)
- Properties of derivative:
  - $f'(x) = \nabla f(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$
  - $(uv)' = u'v + uv'$
  - $\left(\frac{u}{v}\right)' = \frac{u'v-uv'}{v^2}$
  - $(e^u)' = u'e^u$
  - $(\log u)' = \frac{u'}{u}$
- Multi-variate function $f: \mathbb{R}^n \to \mathbb{R}$ with $y = f(x) = f(x_1, \dots, x_n)$.
  - Gradient/derivative: $\frac{\partial f}{\partial x}(a) = \nabla_x f(a) = [\nabla_{x_1} f(a), \nabla_{x_2} f(a), \dots, \nabla_{x_n} f(a)]$.
- Chain rule :
  - $\frac{\partial u}{\partial x} = \frac{\partial u}{\partial v} \times \frac{\partial v}{\partial x}$

# Example

- Consider the function $f(x, y, z) = \log(\exp(x) + \exp(y) + \exp(z))$. What are $f_x' = \nabla_x f$, $f_y' = \nabla_y f$, and $f_z' = \nabla_z f$?

- $f(x, y, z) = \log(u)$ with $u = \exp(x) + \exp(y) + \exp(z)$.
- $f_x' = \frac{u_x'}{u} = \frac{\exp(x)}{\exp(x)+\exp(y)+\exp(z)}$.
- $f_y' = \frac{u_y'}{u} = \frac{\exp(y)}{\exp(x)+\exp(y)+\exp(z)}$.
- $f_z' = \frac{u_z'}{u} = \frac{\exp(z)}{\exp(x)+\exp(y)+\exp(z)}$.
- $\nabla f = \left[f_x', f_y', f_z'\right] = softmax([x, y, z])$.

# Derivative for multi-variate functions




$x = \begin{bmatrix} 1.2 \\ -1 \\ 2 \end{bmatrix}$

- Given a function $f: \mathbb{R}^m \to \mathbb{R}^n$
  - $f(x) = (f_1(x), \dots, f_n(x))$ where $f_1, \dots, f_n: \mathbb{R}^m \to \mathbb{R}$ and $x = (x_1, \dots, x_m)$. Let denote $y = f(x)$.
  - The **derivative** of $f$ at the **point** $\boldsymbol{a} \in \mathbb{R}^m$, denoted by $\nabla f(a)$ (**function related** notion) or $\frac{\partial y}{\partial x}(a)$ (**variable related** notion) is a matrix $n$ by $m$ (i.e., the Jacobian matrix).

$$\frac{\partial y}{\partial x}(a) = \nabla f(a) = \begin{array}{c} \begin{array}{cccccc} & & & m & & \end{array} \\ \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(a) & \dots & \dots & \frac{\partial f_1}{\partial x_j}(a) & \dots & \frac{\partial f_1}{\partial x_m}(a) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \frac{\partial f_i}{\partial x_1}(a) & \dots & \dots & \frac{\partial f_i}{\partial x_j}(a) & \dots & \frac{\partial f_i}{\partial x_m}(a) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \frac{\partial f_n}{\partial x_1}(a) & \dots & \dots & \frac{\partial f_n}{\partial x_j}(a) & \dots & \frac{\partial f_n}{\partial x_m}(a) \end{bmatrix} n \end{array}$$

**Jacobian matrix**

- **Chain rule** 🔗
  - Given a function $\boldsymbol{f}: \mathbb{R}^{\boldsymbol{m}} \to \mathbb{R}^{\boldsymbol{n}}$, $\boldsymbol{g}: \mathbb{R}^{\boldsymbol{n}} \to \mathbb{R}^{\boldsymbol{p}}$, denote $\boldsymbol{h} = \boldsymbol{g} \circ \boldsymbol{f}: \mathbb{R}^{\boldsymbol{m}} \to \mathbb{R}^{\boldsymbol{p}}$, meaning that $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}))$. We further define $y = f(x)$ and $z = g(y) = g(f(x)) = h(x)$.
  - For $\mathrm{x} \in \mathbb{R}^m$, $\underbrace{\nabla h(x)}_{p \times m} = \underbrace{\nabla g(f(x))}_{p \times n} \times \underbrace{\nabla f(x)}_{n \times m}$ or equivalently $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} . \frac{\partial y}{\partial x}$.

$$\begin{tikzcd} x & y = f(x) & z = g(y) = h(x) \\ \mathbb{R}^m \arrow[r, "f"] \arrow[rr, bend right, "h"'] & \mathbb{R}^n \arrow[r, "g"] & \mathbb{R}^p \end{tikzcd}$$

# Example

- $y = f(x) = f(x_1, x_2, x_3) = (x_1^2 + x_2^2, x_2^2 + x_3^2 x_2)$
  - $f: \mathbb{R}^3 \to \mathbb{R}^2$
  - $f_1(x) = f_1(x_1, x_2, x_3) = x_1^2 + x_2^2$
  - $f_2(x) = f_2(x_1, x_2, x_3) = x_2^2 + x_3^2 x_2$
  - $\frac{\partial y}{\partial x} = \nabla f \in \mathbb{R}^{2\times 3}$

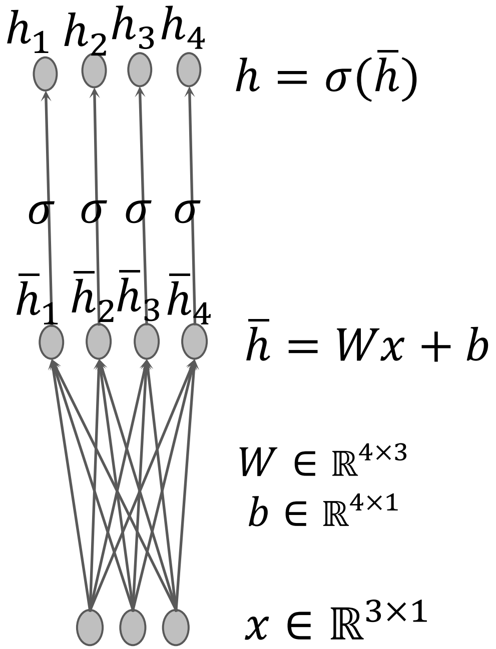
$$\frac{\partial y}{\partial x} = \nabla_x f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_2 & 0 \\ 0 & 2x_2 + x_3^2 & 2x_2 x_3 \end{bmatrix}$$

# Example

- $\bar{h} = Wx + b$ and $h = \sigma(\bar{h})$
  - $h = \sigma(Wx + b)$
  - $\sigma$ is the **activation function**
- $\frac{\partial h}{\partial x} = \frac{\partial h}{\partial \bar{h}} \times \frac{\partial \bar{h}}{\partial x} = diag\left(\sigma'(\bar{h})\right) W$

- $$\frac{\partial h}{\partial \bar{h}} = \begin{bmatrix} \frac{\partial h_1}{\partial \bar{h}_1} & \frac{\partial h_1}{\partial \bar{h}_2} & \frac{\partial h_1}{\partial \bar{h}_3} & \frac{\partial h_1}{\partial \bar{h}_4} \\ \frac{\partial h_2}{\partial \bar{h}_1} & \frac{\partial h_2}{\partial \bar{h}_2} & \frac{\partial h_2}{\partial \bar{h}_3} & \frac{\partial h_2}{\partial \bar{h}_4} \\ \frac{\partial h_3}{\partial \bar{h}_1} & \frac{\partial h_3}{\partial \bar{h}_2} & \frac{\partial h_3}{\partial \bar{h}_3} & \frac{\partial h_3}{\partial \bar{h}_4} \\ \frac{\partial h_4}{\partial \bar{h}_1} & \frac{\partial h_4}{\partial \bar{h}_2} & \frac{\partial h_4}{\partial \bar{h}_3} & \frac{\partial h_4}{\partial \bar{h}_4} \end{bmatrix} = \begin{bmatrix} \sigma'(\bar{h}_1) & 0 & 0 & 0 \\ 0 & \sigma'(\bar{h}_2) & 0 & 0 \\ 0 & 0 & \sigma'(\bar{h}_3) & 0 \\ 0 & 0 & 0 & \sigma'(\bar{h}_4) \end{bmatrix} = diag(\sigma'(\bar{h}))$$

- $\frac{\partial \bar{h}}{\partial x} = W$



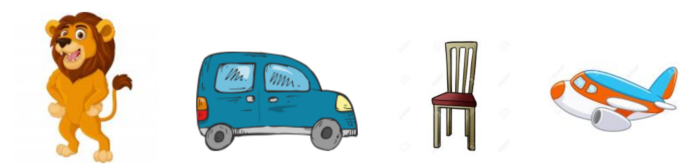
# Information theory revision

# Why need to work with discrete distributions?

**Ground-truth one-hot label**

| lion | car | chair | airplane |
|---|---|---|---|
| 0 | 1 | 0 | 0 |

**Image classification task**

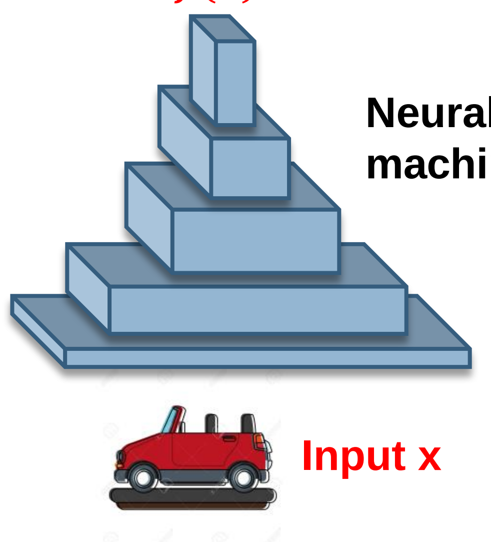


**Prediction probability**

| lion | car | chair | airplane |
|---|---|---|---|
| 0.6 | 0.2 | 0.1 | 0.1 |

**How diverge and distant?**

$f(x)$



**Neural Net or machine learning model $f$**

**Input x**

# Revision of basic informatic quantities

- Given two **discrete distributions** $p = [p_i]_{i=1}^{d}$ ($p_i \geq 0$ and $\sum_{i=1}^{d} p_i = 1$) and $q = [q_i]_{i=1}^{d}$ ($q_i \geq 0$ and $\sum_{i=1}^{d} q_i = 1$).

- **Kullback-Leibler (KL) divergence between $\boldsymbol{p}, \boldsymbol{q}$**
  - $KL(p, q) = \sum_{i=1}^{d} p_i \log \frac{p_i}{q_i}$
  - $KL(p, q) \geq 0$ and $KL(p, q) = 0$ if only if $p = q$

- **Entropy of the distribution $\boldsymbol{p}$**
  - $H(p) = \sum_{i=1}^{d} p_i \log \frac{1}{p_i} = -\sum_{i=1}^{d} p_i \log p_i$ where $0 \times \log 0 = 0$

- **Cross-entropy (CE) divergence between $\boldsymbol{p}, \boldsymbol{q}$**
  - $CE(p, q) = -\sum_{i=1}^{d} p_i \log q_i$.
  - $CE(p, q) = KL(p, q) + H(p)$.

# Example

- Consider $p = [0.1,0.4,0.5]$ and q $= [0.2,0.5,0.3]$. Compute $KL(p, q)$, $CE(p, q)$, and $H(p)$.
  - $KL(p,q) = 0.1 \times \log\frac{0.1}{0.2} + 0.4 \times \log\frac{0.4}{0.5} + 0.5 \times \log\frac{0.5}{0.3} \approx 0.042$
  - $CE(p,q) = -0.1 \times \log 0.2 - 0.4 \times \log 0.5 - 0.5 \times \log 0.3 \approx 0.452$
  - $H(p) = 0.1 \times \log\frac{1}{0.1} + 0.4 \times \log\frac{1}{0.4} + 0.5 \times \log\frac{1}{0.5} \approx 0.41$

- Consider $p = [0,1,0]$ and q $= [0.2,0.3,0.5]$. Compute $CE(p, q)$.
  - $CE(p,q) = -0 \times \log 0.2 - 1 \times \log 0.3 - 0 \times \log 0.5 \approx -\log 0.3 = 0.523$

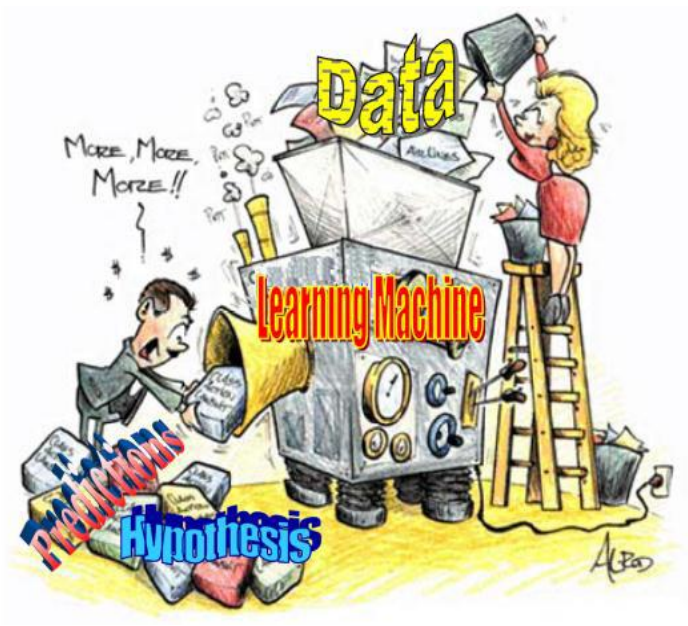
**Question**: when the entropy is maximized or minimized?

# Machine Learning Revisited

# Machine learning

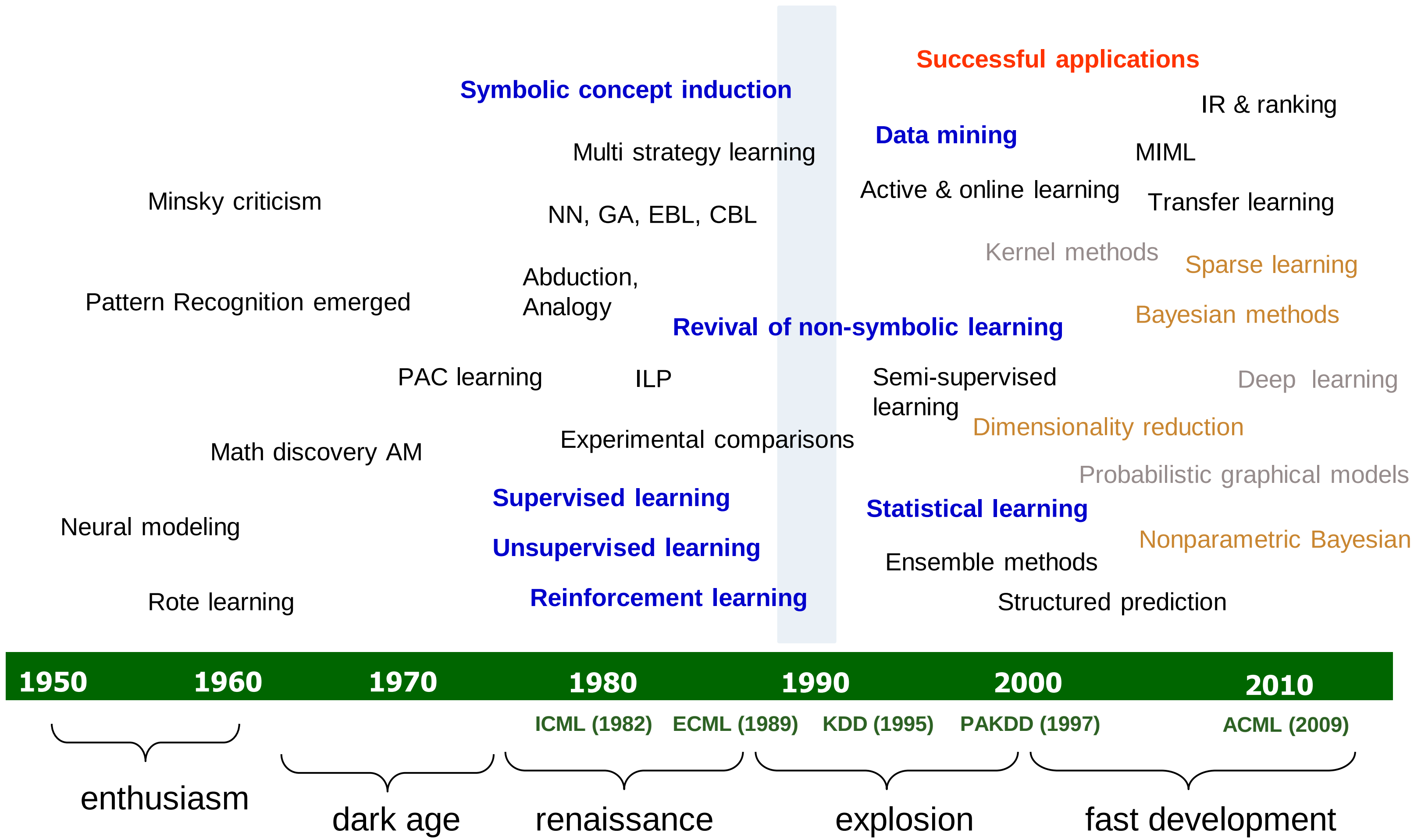
- "Field of study that gives computers the ability to learn without being explicitly programmed" (Arthur Samuel, 1959).
- "Machine learning sits at the crossroads of computer science, statistics and a variety of other disciplines concerned with automatic improvement over time, and inference and decision-making under uncertainty." (Jordan & Michell, 2015).
- *"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"*, (Tom Mitchell, 1997)

M.I. Jordan and T. Mitchell, "Machine Learning: Trends, perspectives, and prospects", *Science*, 17 July 2015.




(from Eric Xing lecture notes)

# A brief history of ML




[credit: Prof Bao Ho, JAIST]

# Machine learning


Face recognition system

| Experience E | Task T | Performance P |
|---|---|---|
| dataset of thousands of known faces | given a new photo, recognise the name of the face<br>**Label set**: names of all people. | how **accurate** the recognition is<br>Measure: **Accuracy** |


Detecting Spam Emails

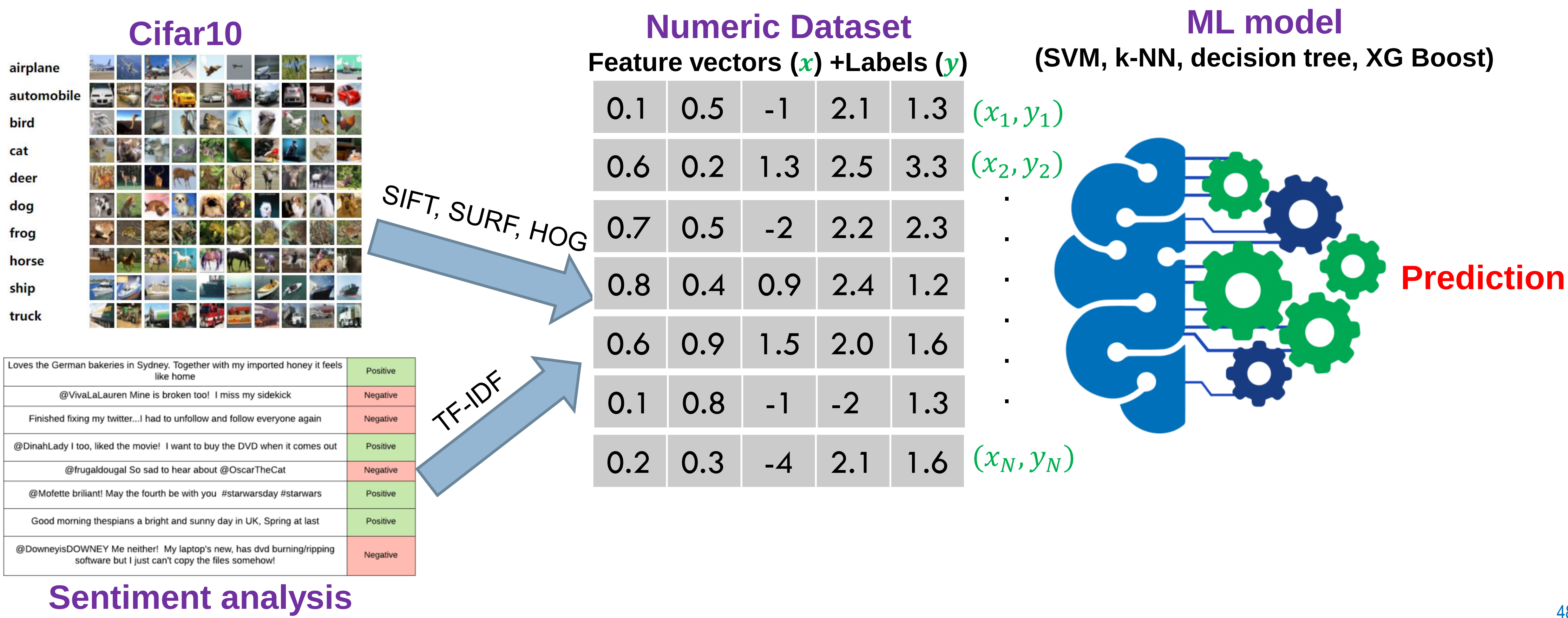
| Experience E | Task T | Performance P |
|---|---|---|
| Examples of spam emails and not-spam emails | To assign a label "spam" or "not-spam" to an email<br>Label set: **spam** and **not spam** | how **accurate** spam email can be detected<br>Measure: **accuracy** |


Detect credit card fraud

| Experience E | Task T | Performance P |
|---|---|---|
| Data collected for credit-card transactions deemed as fraud and not-fraud | To assign a label 'fraud' or "not fraud" to a given credit-card transaction<br>Label set: **fraud** and **not fraud** | how accurate a credit-card fraud transaction can be detected.<br>Measure: **accuracy** |

# Feature extraction (Forum discussion)

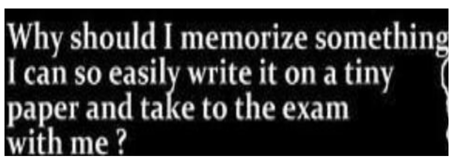
- Most machine learning algorithms (e.g, Support Vector Machine or XG Boosts) only **work with numeric data**
  - Given **structural data** such as images or texts, we **extract numeric features** that preserve characteristics of raw data
  - **Feature extraction methods**
    - **Images**: SIFT, SURF, and HOG.
    - **Texts**: bag of words and TF-IDF.



# Elements of machine learning

## 1 Data

**Vector** $\boldsymbol{x} \in \mathbb{R}^d$
**Image** $\boldsymbol{x}$
**Sentence** $\boldsymbol{x}$
**Speech** $\boldsymbol{x}$
**Ground-truth label** $y \in Y$

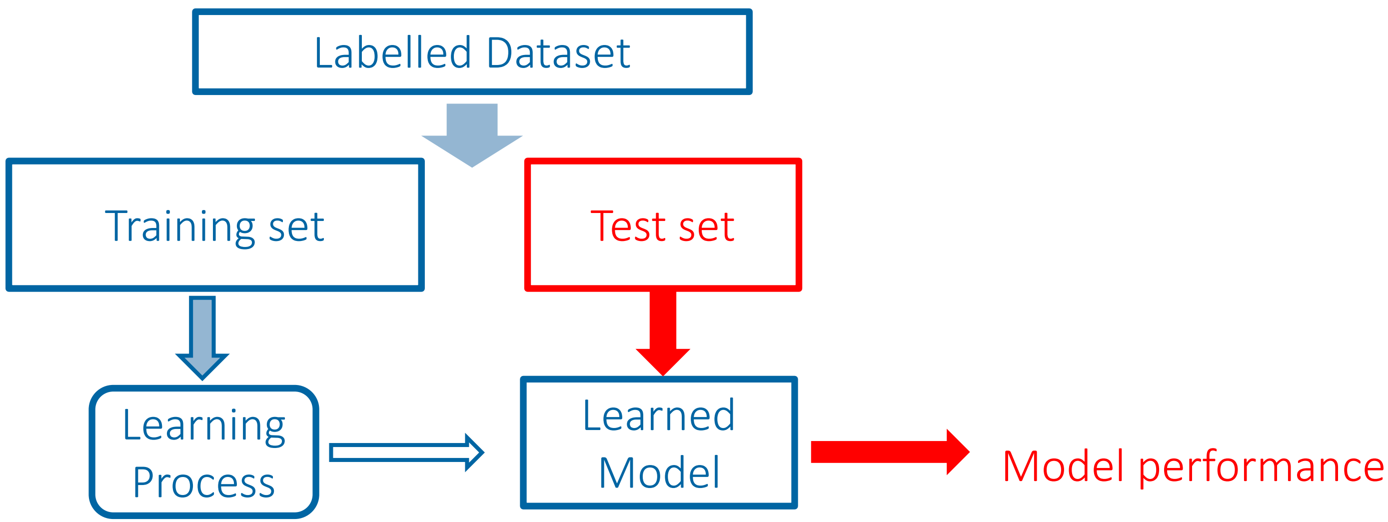



## 2 Model

- $f: X \to Y$
- $X$ is **data space**, $Y$ is **label space**
- **Classification**: $Y = \{1,2, \dots, M\}$
- **Regression**: $Y = \mathbb{R}$
- **Prediction**: $f(x) = \hat{y} \in Y$

## 3 Assessment

- How **good** is the model doing its tasks?
- **Performance metrics**
  - Accuracy, Recall, Precision, F-score
  - $Accuracy = \frac{\#Correct\ Predictions}{\#All\ Predictions}$
    : % of correctly classified examples

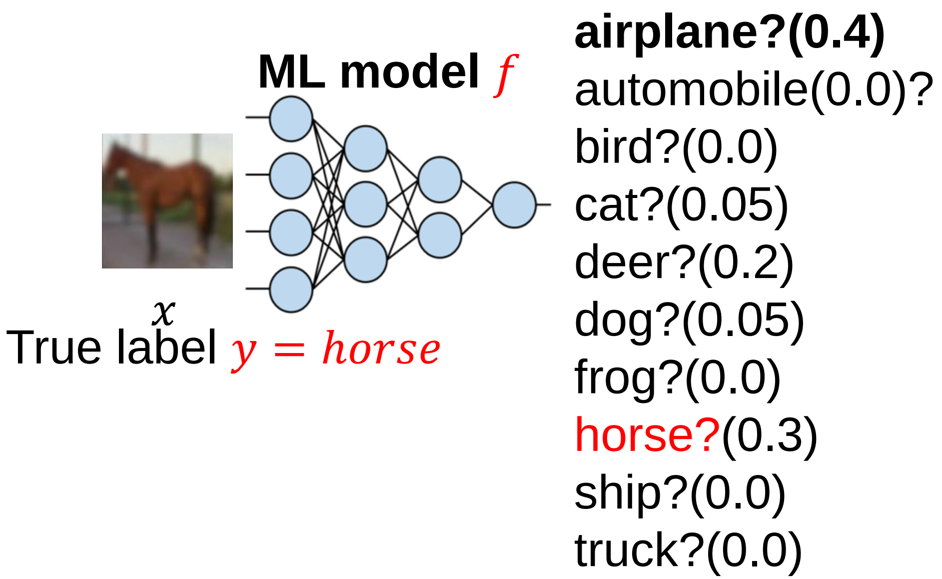



Machine learning pipeline.

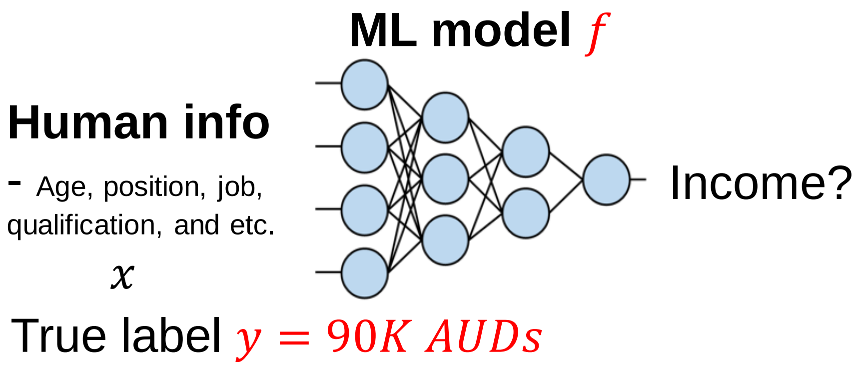
# Classification and Regression

- Two **main tasks** in machine learning
  1. **Classification** and **regression**
  2. Other tasks in ML include density estimation, clustering, anomaly detection, recommendation, forecasting, and ranking

- **Classification**
  1. The label set $Y = \{1,2,\ldots,M\}$ and the data space $X$ (vectors, images).
  2. Training set $\boldsymbol{D} = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_N}, \boldsymbol{y_N})\}$ where $\boldsymbol{x_i} \in \mathbb{R}^d$ and $y_i \in Y = \{1,2,\ldots,M\}$
  3. Learn a model $f: X \to Y$
  4. Given data example $\boldsymbol{x}$, render the prediction $\hat{y} = f(\boldsymbol{x}) \in \{1,2,\ldots,M\}$




- **Regression**
  1. The label set $Y = \mathbb{R}$ and the data space $X$ (vectors, images).
  2. Training set $\boldsymbol{D} = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_N}, \boldsymbol{y_N})\}$ where $\boldsymbol{x_i} \in \mathbb{R}^d$ and $y_i \in Y = \mathbb{R}$
  3. Learn a model $f: X \to Y$
  4. Given data example $\boldsymbol{x}$, render the prediction $\hat{y} = f(\boldsymbol{x}) \in \mathbb{R}$



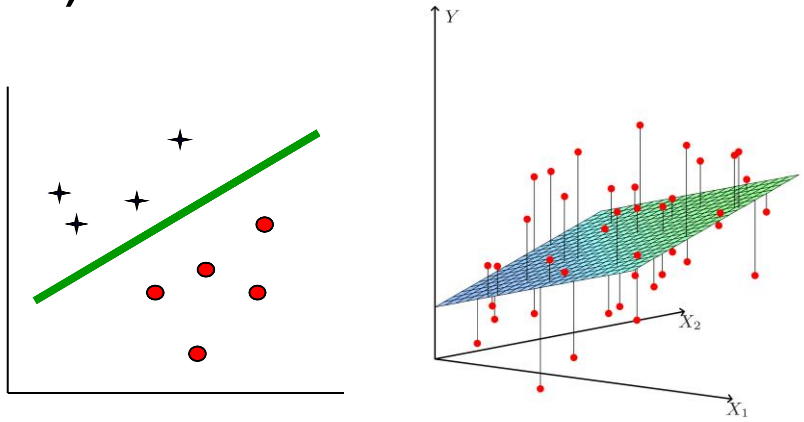
# Supervised learning (Forum discussion)

Problem: Learn a **function** from **data** to **relate the inputs** with **outputs**. Training data include output information (labels).

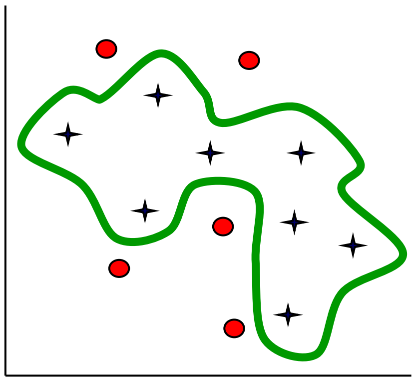
Data: $D = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}$

Function: $f: X \rightarrow Y$

$x \in \mathbb{R}^d$ = feature
$y \in \{1,2, \ldots, M\}$ = a discrete label (classification),
$y \in \mathbb{R}$ = a continuous value (regression)



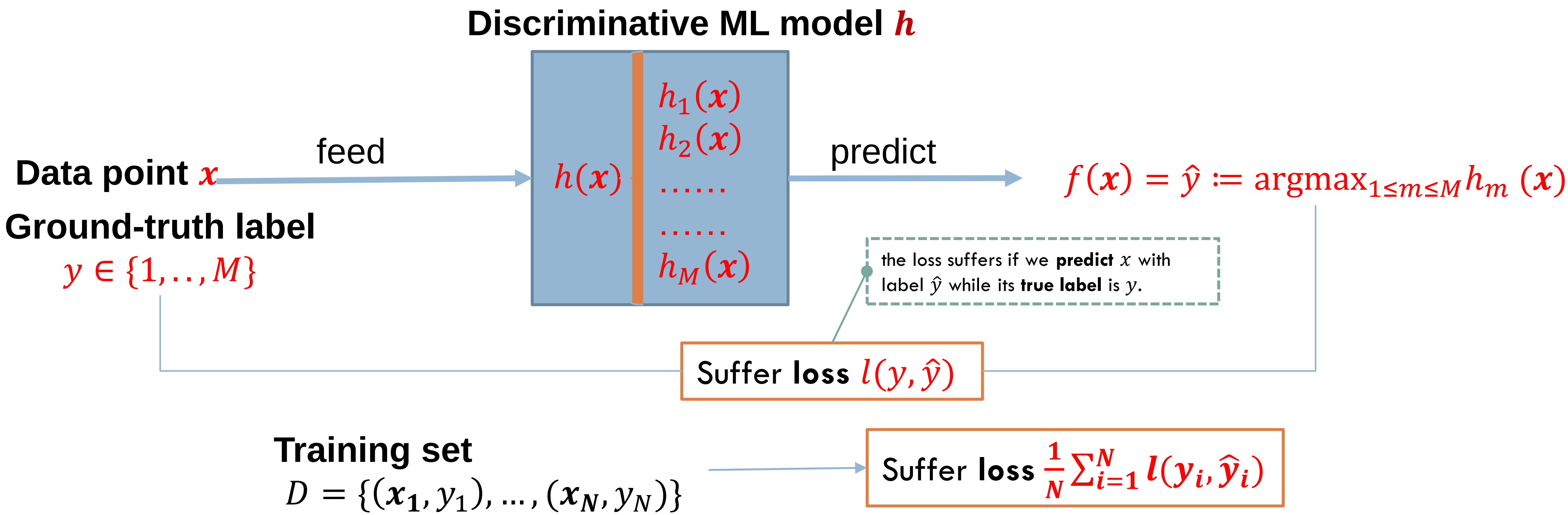

e.g., linear functions



e.g., nonlinear functions

# Discriminative machine learning

## Classification

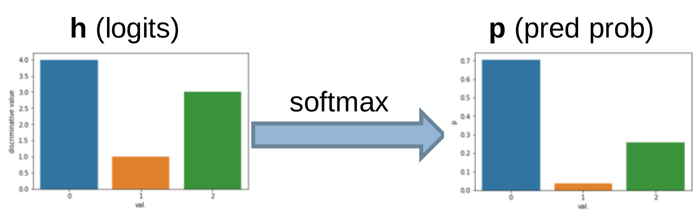



- A data point $x$ receives **discriminative values** $h_1(\boldsymbol{x}), \dots, h_M(\boldsymbol{x})$ from the model
  - $h_m(\boldsymbol{x})$ represents the **possibility** to **classify** $x$ to **class** $\boldsymbol{m}$ for $1 \le m \le M$
- We use these discriminative values to predict the label $\hat{y}$ as
  - $\hat{y} = \mathrm{argmax}_{1\le m\le M} h_m(\boldsymbol{x})$, meaning the class with highest discriminative value
- The prediction $x$ with the predicted label $\hat{y}$ suffers a loss
  - $l(\hat{y}, y)$ where $l$ is a **loss function** (if $\hat{y} = y$ then $l(\hat{y}, y) = 0$).
- Given a training set $D = \{(\boldsymbol{x_1}, y_1), \dots, (\boldsymbol{x_N}, y_N)\}$, the loss incurred is
  - $\frac{1}{N}\sum_{i=1}^{N} l(\boldsymbol{y_i}, \hat{\boldsymbol{y}}_i)$

# Softmax activation function

- Use to transform **real-valued discriminative scores** to **discrete probabilities**
  - $h = [h_m]_{m=1}^{M} \to p = \text{softmax}(h) := \left[\frac{\exp\{h_m\}}{\sum_{i=1}^{M} \exp\{h_i\}}\right]_{m=1}^{M}$
  - $p = [p_m]_{m=1}^{M}$ becomes a distribution over classes $\{1, \dots, M\}$
    - $p_m \geq 0 \ (1 \leq m \leq M)$ and $\sum_{m=1}^{M} p_m = 1$ .

- Assume that we do **sentiment classification** (analysis)
  - Data point $x$ is a sentence (e.g., $x =$ "$it\ is\ a\ beautiful\ day\ today$"), we need to predict **sentiment** of this sentence
  - Three **sentiment labels**: positive (happy, class 1), negative (sad, class 2), neural (none happy and sad, class 3)
  - The model gives three **discriminative values** for $x$
    - $h_1 = 2, h_2 = -1, h_3 = 1$ means that **highest possibility** to **classify** $x$ to the **class 1** with the label **positive**.
  - We apply **softmax** function on the **discriminative scores** $h$
    - $p = \text{softmax}(h)$
      - $p_1 = \frac{\exp\{2\}}{\exp\{2\}+\exp\{-1\}+\exp\{1\}} \approx 0.705$, $p_2 = \frac{\exp\{-1\}}{\exp\{2\}+\exp\{-1\}+\exp\{1\}} \approx 0.035$, $p_3 = \frac{\exp\{1\}}{\exp\{2\}+\exp\{-1\}+\exp\{1\}} \approx 0.259$
    - $p = [0.70538451, 0.03511903, 0.25949646]$ are the **probabilities** to **classify** $x$ to the **classes $1, 2, 3$** respectively.




**Softmax** *transforms real-valued discriminative scores (ranged in $(-\infty, +\infty)$) to probability values (ranged in $[0, 1]$) but preserving the order.*

# Cross-entropy loss

- Given $M$ labels $1,2,\ldots,M$, a label $y \in \{1,\ldots,M\}$ has two equivalent forms:
  - **Numeric form**: $y \in \{1,\ldots,M\}$ is an **index** in $\{1,\ldots,M\}$.
  - **One-hot vector form**: $1_y = [0,..,0,1_y,0,\ldots,0]$- the vector of **zeros** except **an unique 1** in the $y$-th position.

- Label set $\{happy, sad, neural\}$ with $1{:}\,happy$, $2{:}\,sad$, and $3{:}\,neural$
  - **[1,0,0]** → happy (numeric label 1), **[0,1,0]**→sad (numeric label 2), **[0,0,1]**→neural (numeric label 3)

- Given two **discrete distributions** over classes: $p = [p_m]_{m=1}^{M}$ and $\mathrm{q} = [q_m]_{m=1}^{M}$, the **cross-entropy divergence** between $p$ and $q$
  - $\mathrm{CE}(p,q) \coloneqq -\sum_{m=1}^{M} p_m \log q_m$ which measures how far (divergent) it is between $p$ and $q$.
  - $\mathrm{CE}(p,q) \geq \mathrm{H}(p) \coloneqq -\sum_{m=1}^{M} p_m \log p_m$ ($\mathrm{H}$ is the **Shannon entropy**).

# Cross-entropy loss

- **Cross-entropy loss**
  - $l(y, \hat{y}) = \mathrm{CE}\big(1_y, p\big) = -\log p_y$
  - $y \in \{1, \dots, M\}$ is the **ground-truth label** of a given $x$ and $p$ is **the prediction probabilities** of $x$ to classes

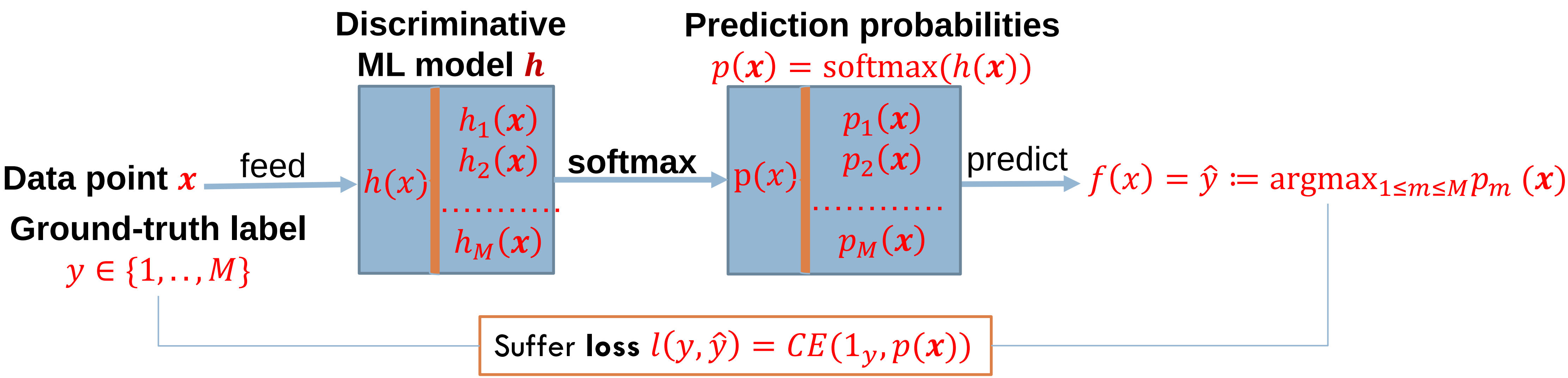
- Given **a sentence** $x$ with the **label positive/happy** (the class 1), assume that our model predicts it with **prediction probabilities** $p = [0.3, 0.4, 0.3]$, the **cross-entropy loss** for this prediction
  - $l(y, \hat{y}) = CE([1,0,0], [0.3,0.4,0.3]) = -1.\log 0.3 - 0.\log 0.4 - 0.\log 0.3 = -\log 0.3 \approx 1.204$

**Question**: why is the **cross-entropy loss** always **non-negative** and when it is **0**?

- CE loss is 0, when we have perfect accuracy

# Discriminative machine learning

## Classification with the cross-entropy loss



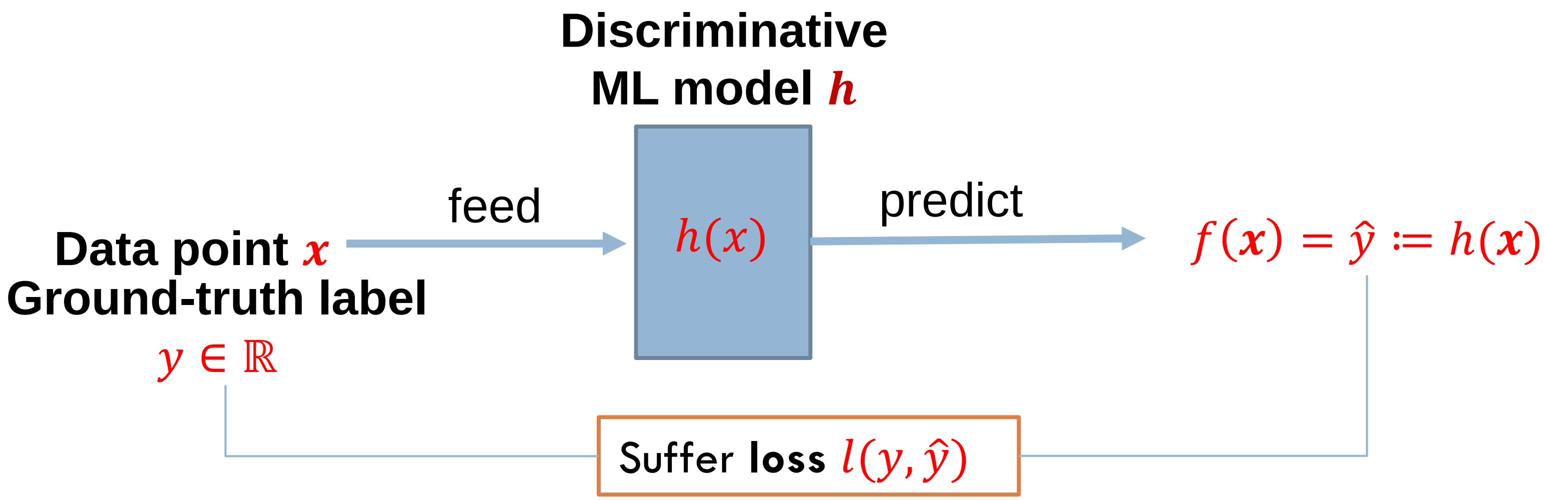

- We apply the **softmax function**: $\mathbf{p}(\boldsymbol{x}) = \mathbf{softmax}(\mathbf{h}(\boldsymbol{x}))$ to make $p(x)$ **prediction probabilities**
  - $p_m(x) = p(y = m|x)$ is the **probability** to classify $x$ to the class $m$ for $1 \le m \le M$
- The loss incurred for this prediction
  - $l(y, \hat{y}) = CE\left(1_y, p(\boldsymbol{x})\right) = -\log p_y(\boldsymbol{x})$ (**negative log likelihood**)

the log base used can be any, but must be uniform across all data points

- The **loss for the training set** $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
  - $L(D, h) = \frac{1}{N}\sum_{i=1}^{N} CE(1_{y_i}, p(\boldsymbol{x}_i)) = -\frac{1}{N}\sum_{i=1}^{N} \log p_{y_i}(\boldsymbol{x}_i)$

# Discriminative machine learning

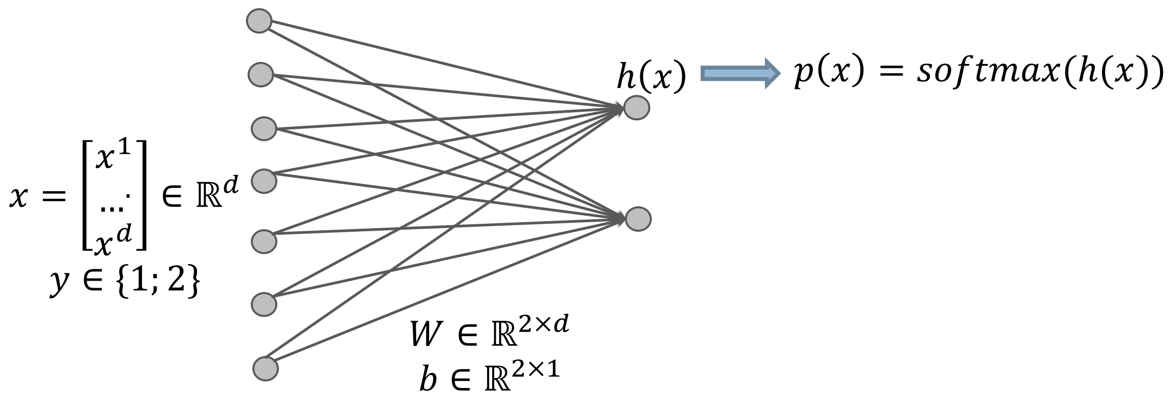
## Regression




- Only **one discriminative value** $h(\boldsymbol{x}) \in \mathbb{R}$, we use this to predict $\hat{y} = f(\boldsymbol{x}) := h(\boldsymbol{x})$.
- The **potential losses** could be
  - **L2 loss**: $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$
  - **L1 loss**: $l(y, \hat{y}) = |y - \hat{y}|$
  - **$\epsilon$-insensitive loss**: $l(y, \hat{y}) = \max\{0, |y - \hat{y}| - \epsilon\}$ where $\epsilon \geq 0$ .

# Logistic regression (Forum discussion)

## A simple feed-forward neural network

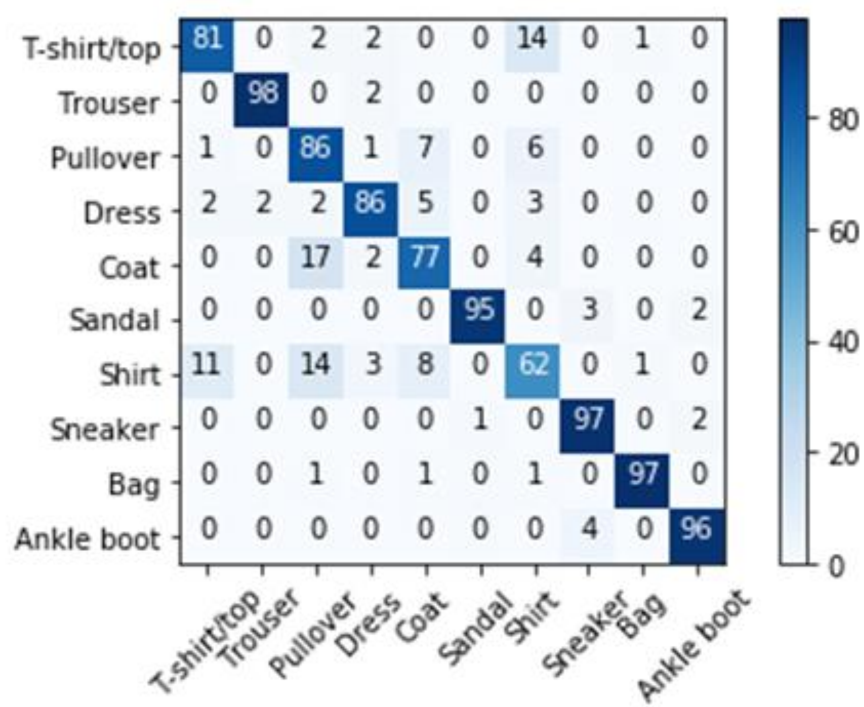



- **Computational process**
  - $h(x) = Wx + b$
    - $h(x) \in \mathbb{R}^{2\times 1}$ contains **discriminative scores** with respect to classes 1 and 2.
  - $p(x) = softmax(h(x))$
    - $p(x)$ is the **prediction probabilities** with respect to classes 1 and 2.
  - $\hat{\boldsymbol{y}} = \begin{cases} \mathbf{1}, & \boldsymbol{p_1(x)} \geq \boldsymbol{p_2(x)} \\ \mathbf{2}, & \boldsymbol{p_1(x)} < \boldsymbol{p_2(x)} \end{cases}$
  - $l(y, \hat{y}) = CE\left(1_y, p(x)\right) = -\log p_y(x)$ → the **cross entropy loss**.
- **Training process**
  - **Training set** $D = \{(\boldsymbol{x_1}, y_1), \dots, (\boldsymbol{x_N}, y_N)\}$
  - $L(D; W, b) = \frac{1}{N}\sum_{i=1}^{N} CE(1_{y_i}, p(\boldsymbol{x_i})) = -\frac{1}{N}\sum_{i=1}^{N} \log p_{y_i}(\boldsymbol{x_i})$
  - $(W^*, b^*) = \text{argmin}_{W,b} L(D; W, b)$

# Confusion matrix

- How **confusing** a machine learning classifier is?
  - On-diagonal entries mean **correct** predictions.
  - Off-diagonal entries mean **incorrect** predictions.

Predicted class



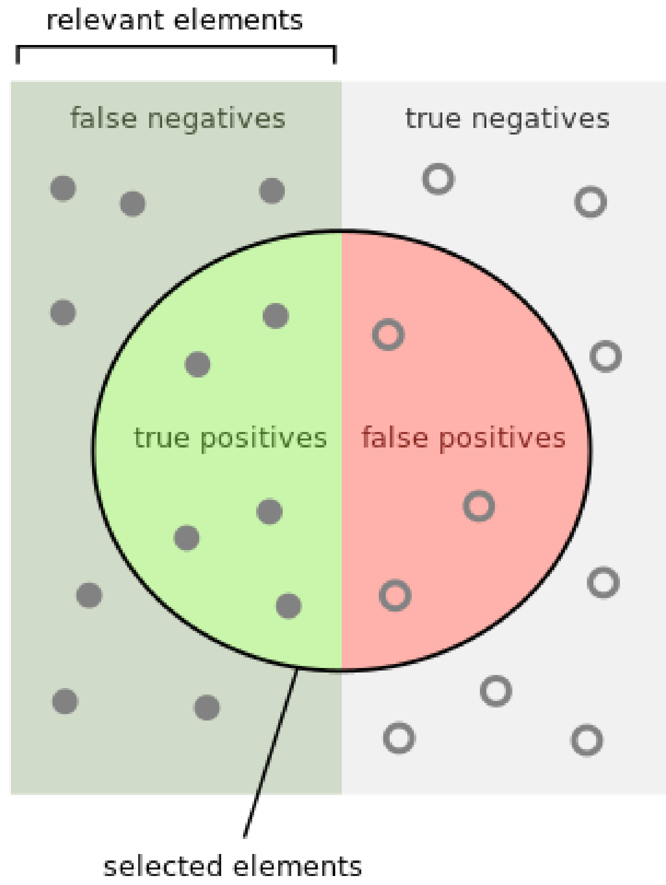

Actual class

[Source: pythonhealthcare.org]

# Other metrics for measuring performance

- TP, FP, TN, FN and confusion matrix

| | | actual class: positive | actual class: negative |
|---|---|---|---|
| predicted class | positive | true positives (TP) | false positives (FP) |
| | negative | false negatives (FN) | true negatives (TN) |




- We may think of positive class as "class 1" and Negative class as "class 0".
- The second letter says what we predicted and the first letter says whether it was true or false.

# Other metrics for measuring performance

- Precision, recall, and F-score

$$\text{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

$$\mathrm{F} - \mathrm{score} = 2\,\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$$

$$\mathrm{Accuracy} = \frac{\#\mathrm{corrects}}{\#\mathrm{examples}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{TN}}$$

| | | actual class | |
|---|---|---|---|
| | | positive | negative |
| predicted class | positive | true positives (TP) | false positives (FP) |
| | negative | false negatives (FN) | true negatives (TN) |

# Example

- Example: Email Spam Detection
  - In test set: 10 spam, 20 non-spam
  - Positive = spam

- TP = 7, TN = 15
- FP = 5, FN = 3
- Accuracy = 22/30
- Recall = 7/10
- Precision = 7/12
- F-score = ??

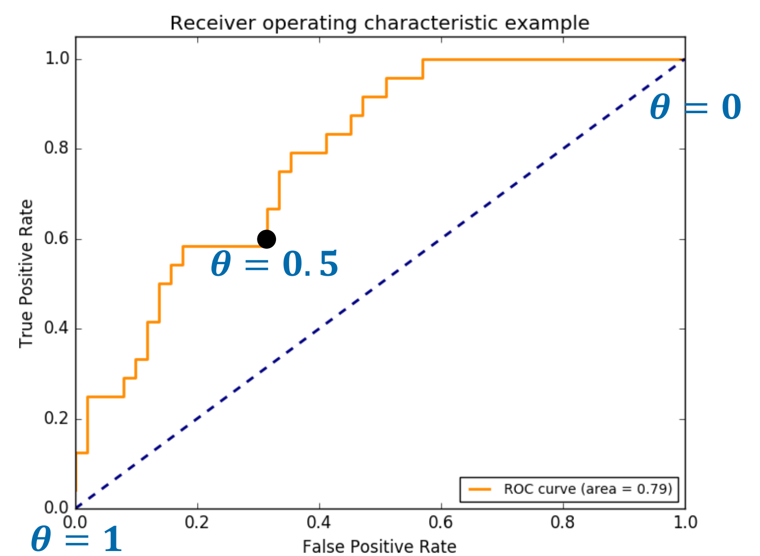
| Predicted as | True Labels: SPAM (1) | True Labels: NON-SPAM (0) |
|---|---|---|
| SPAM (1) | 7 | 5 |
| NON-SPAM (0) | 3 | 15 |

| predicted class | actual class: positive | actual class: negative |
|---|---|---|
| positive | true positives (TP) | false positives (FP) |
| negative | false negatives (FN) | true negatives (TN) |

# ROC curve (Forum discussion)

- For binary classification
  - **True positive rate** (TPR) (**Sensitivity)**
    - TPR = TP/(TP + FN)
  - **False positive rate** (FRR)
    - FPR = FP / (FP + TN)
- When output also includes probability (e.g., logistic regression), we can compute **ROC** by varying the detection threshold.
  - **ROC curve** = **true-positive rate** (TPR) **versus false-positive rate** (FPR)
  - Consider **a threshold** $\theta \in [0;1]$. Given a data point $x$, the ML model returns $p(x) = \mathbb{P}(y = 1|x)$ (i.e., the **probability to predict** $x$ as the label $1$). We predict $x$ as the **label 1** if $p(x) = \mathbb{P}(y = 1|x) \geq \theta$ and the **label -1** otherwise.
  - We compute **true positive rate** (TPR) and **false positive rate** (FPR) for the entire training set and plot **the point** [TPR, FPR].
  - We vary $\theta \in [0;1]$ to gain the ROC curve.

| | | actual class | |
|---|---|---|---|
| | | positive | negative |
| predicted class | positive | true positives (TP) | false positives (FP) |
| | negative | false negatives (FN) | true negatives (TN) |




[Source: Wikipedia]

# Summary

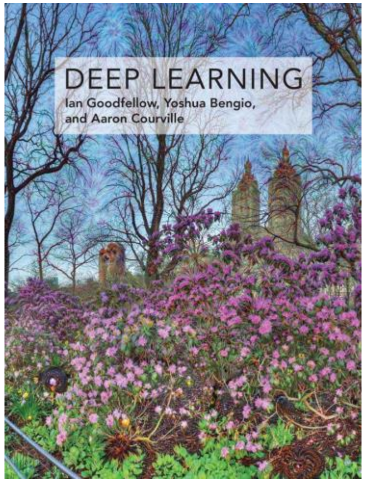
- Linear algebra :
  - Vectors, matrices and operations
- Calculus
  - Derivative and chain rule
- Information theory
  - Discrete distribution, KL divergence, entropy, and cross-entropy.
- Machine learning revisit
  - Classification and regression
  - Discriminative machine learning
  - Confusion table and other metrics of interest (e.g., precision, recall, and F-score).

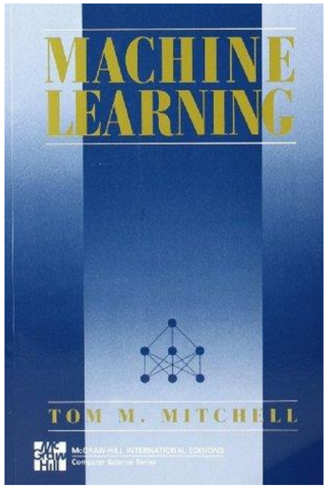
Thanks for your attention!

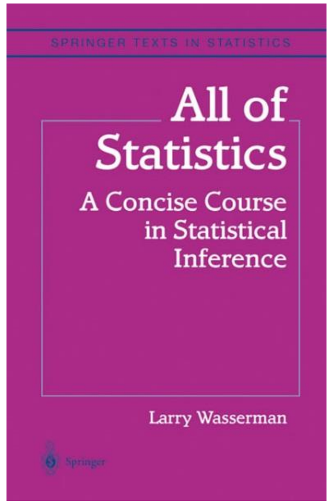
# Appendix

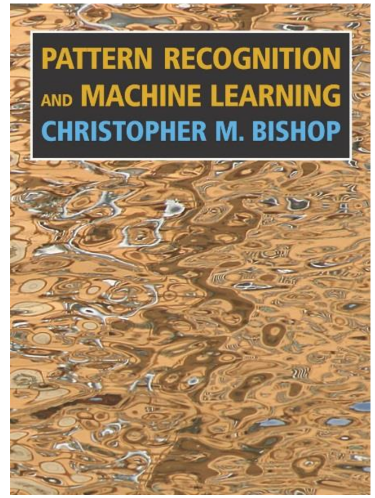
 68

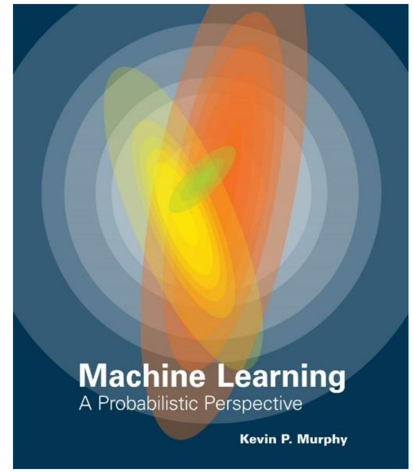
# Other ML Books

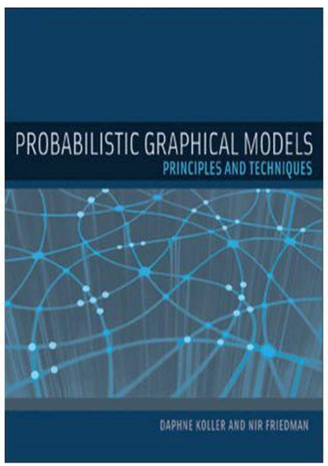


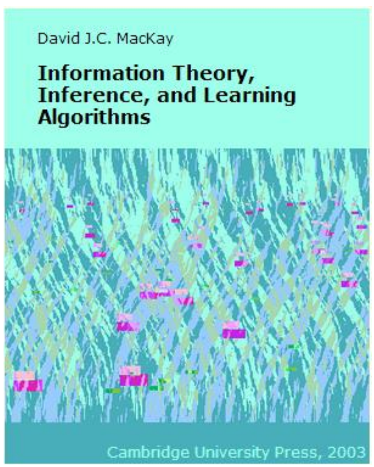



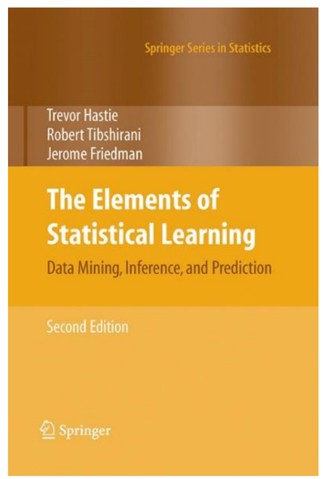



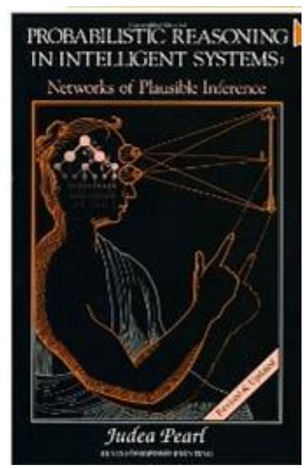



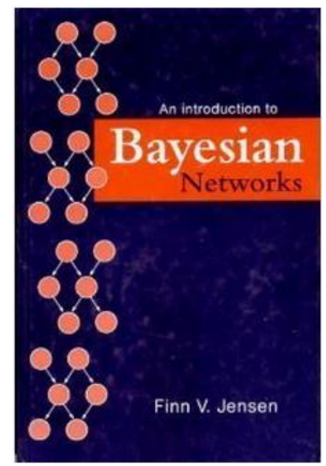



# Machine learning: five tribes

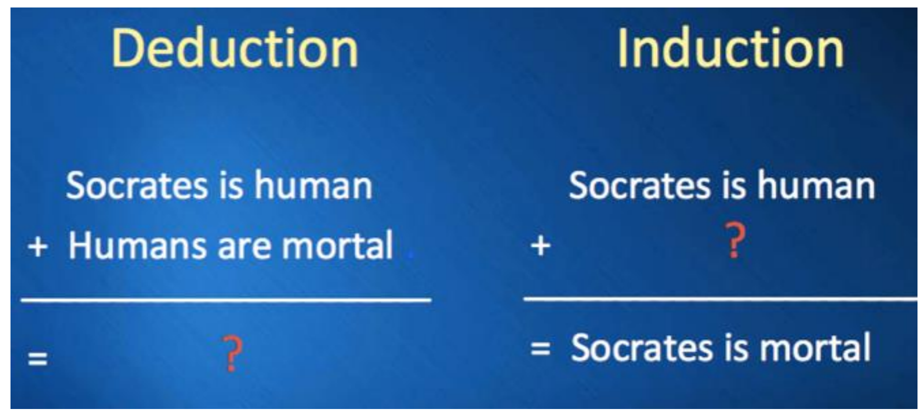
## The Symbolist School

| Tribes | Origins | Master Algorithm |
|---|---|---|
| Symbolists | Logic, philosophy | *Inverse deduction* |

Tom Mitchell

Steve Muggleton

Ross Quinlan




Decision trees, C4.5, first-order propositional logic, rule-based reasoning systems, etc.

# Machine learning: five tribes

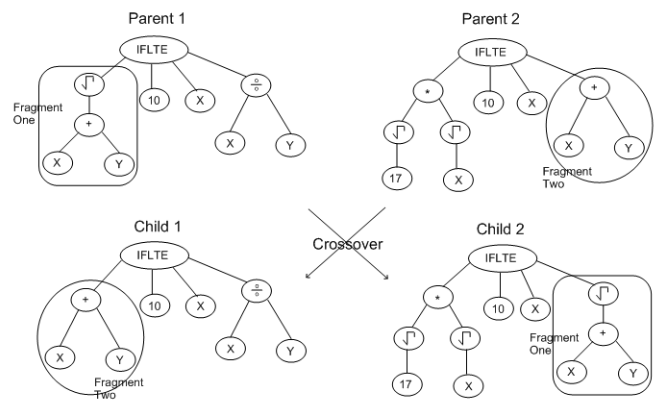
| Tribes | Origins | Master Algorithm |
|---|---|---|
| Symbolists | Logic, philosophy | *Inverse deduction* |
| Evolutionary | Evolutionary biology | *Genetic programming* |

## The Evolutionary School

John Koza

John Holland

Hod Lipson




Genetic programming, genetic algorithms, fuzzy systems

# Machine learning: five tribes

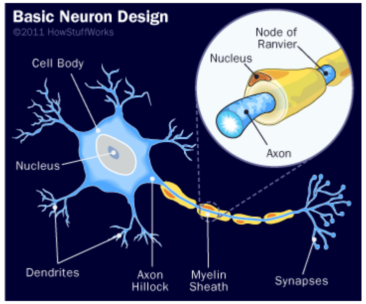
## The Connectionist

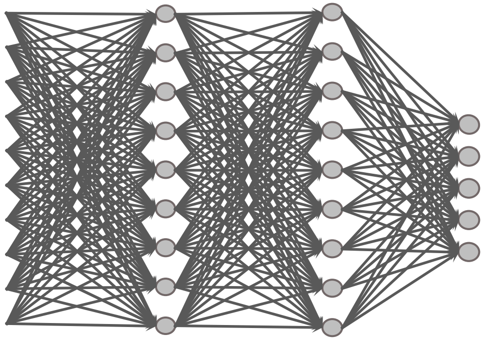
| Tribes | Origins | Master Algorithm |
|---|---|---|
| Symbolists | Logic, philosophy | *Inverse deduction* |
| Evolutionary | Evolutionary biology | *Genetic programming* |
| Connectionists | Neuroscience | *Backpropagation* |

Geoff Hinton

Yann LeCun

Yoshua Bengio



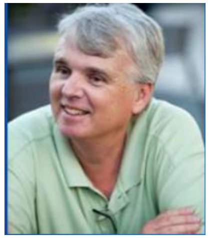



Deep Neural Networks, Convolutional Neural Networks, Deep learning, etc.

# Machine learning: five tribes

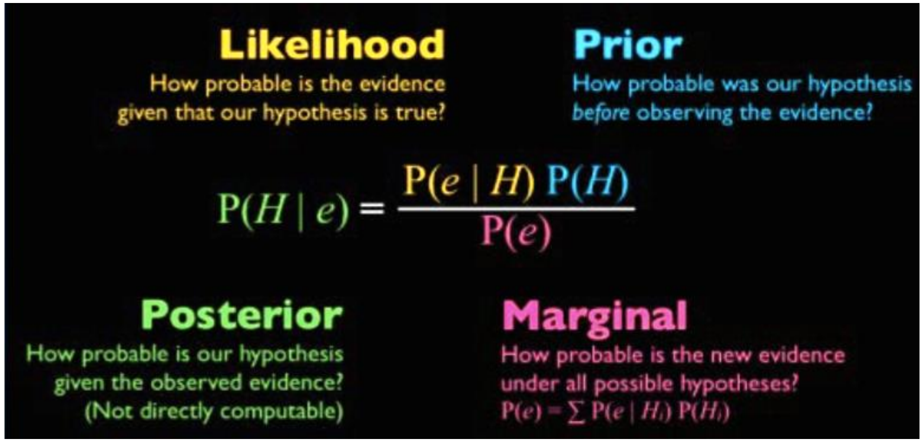
## The Bayesians

| Tribes | Origins | Master Algorithm |
|---|---|---|
| Symbolists | Logic, philosophy | *Inverse deduction* |
| Evolutionary | Evolutionary biology | *Genetic programming* |
| Connectionists | Neuroscience | *Backpropagation* |
| Bayesians | Statistics | *Bayes' theorem* |


David Heckerman


Judea Pearl


Michael Jordan




Bayesian networks classifiers, graphical models, Hidden Markov models, Conditional Random Fields, most of statistical machine learning approaches …

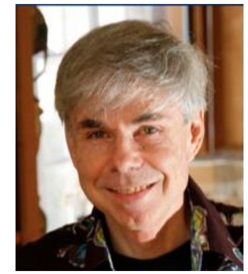
# Machine learning: five tribes

| Tribes | Origins | Master Algorithm |
|---|---|---|
| Symbolists | Logic, philosophy | *Inverse deduction* |
| Evolutionary | Evolutionary biology | *Genetic programming* |
| Connectionists | Neuroscience | *Backpropagation* |
| Bayesian | Statistics | *Bayes' theorem* |
| Analogizers | Psychology | *Support Vector Machines* |

## The Analogizers

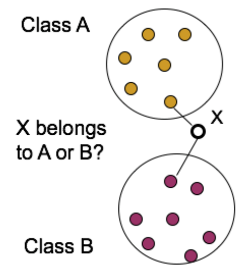
Vladimir Vapnik

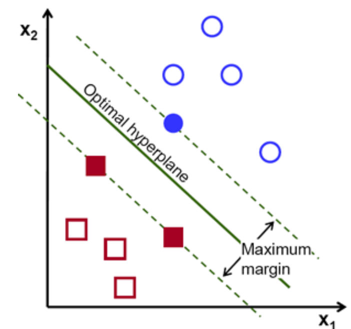
Peter Hart



Douglas Hofstadter

- **Instance-based classification**
  - Using most similar individual instances known in the past to classify a new instance
- **Typical approaches**
  - **k-nearest neighbor approach**
    - Instances represented as points in a Euclidean space



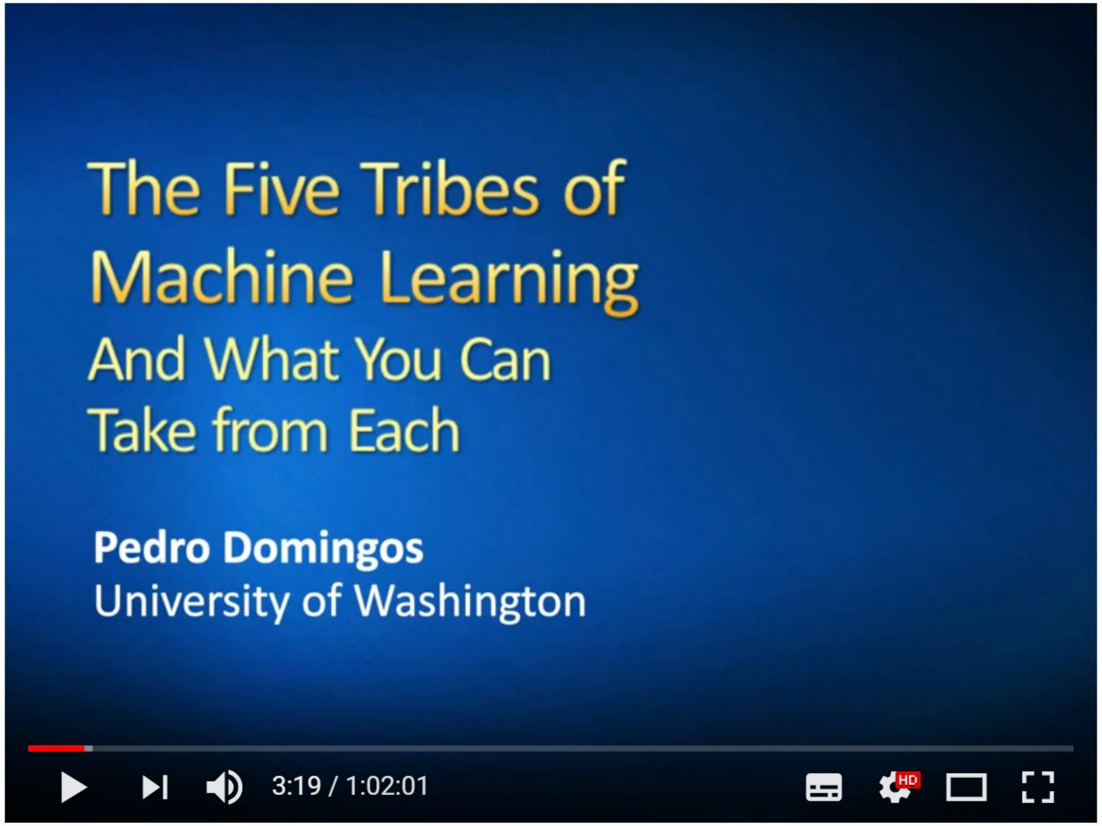




K-NN classifiers, Support vector machines, kernel machines, max-margin machines, instance-based classification, etc.

# Machine learning: five tribes

| Approaches | Problem Solved | Solution Origin | Solution |
|---|---|---|---|
| Symbolists | Knowledge acquisition | Logic, philosophy | *Inverse deduction* |
| Evolutionary | Structure discovery (nature) | Evolutionary biology | *Genetic programming* |
| Connectionists | Credit assignment (nurture) | Neuroscience | *Backpropagation and deep learning* |
| Bayesian | Uncertainty | Statistics | *Probabilistic Inference* |
| Analogizers | Similarity | Psychology | *Kernel machines* |

# Machine learning: five tribes




https://www.youtube.com/watch?v=E8rOVwKQ5-8