Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Introduction to Machine Learning and Deep Learning

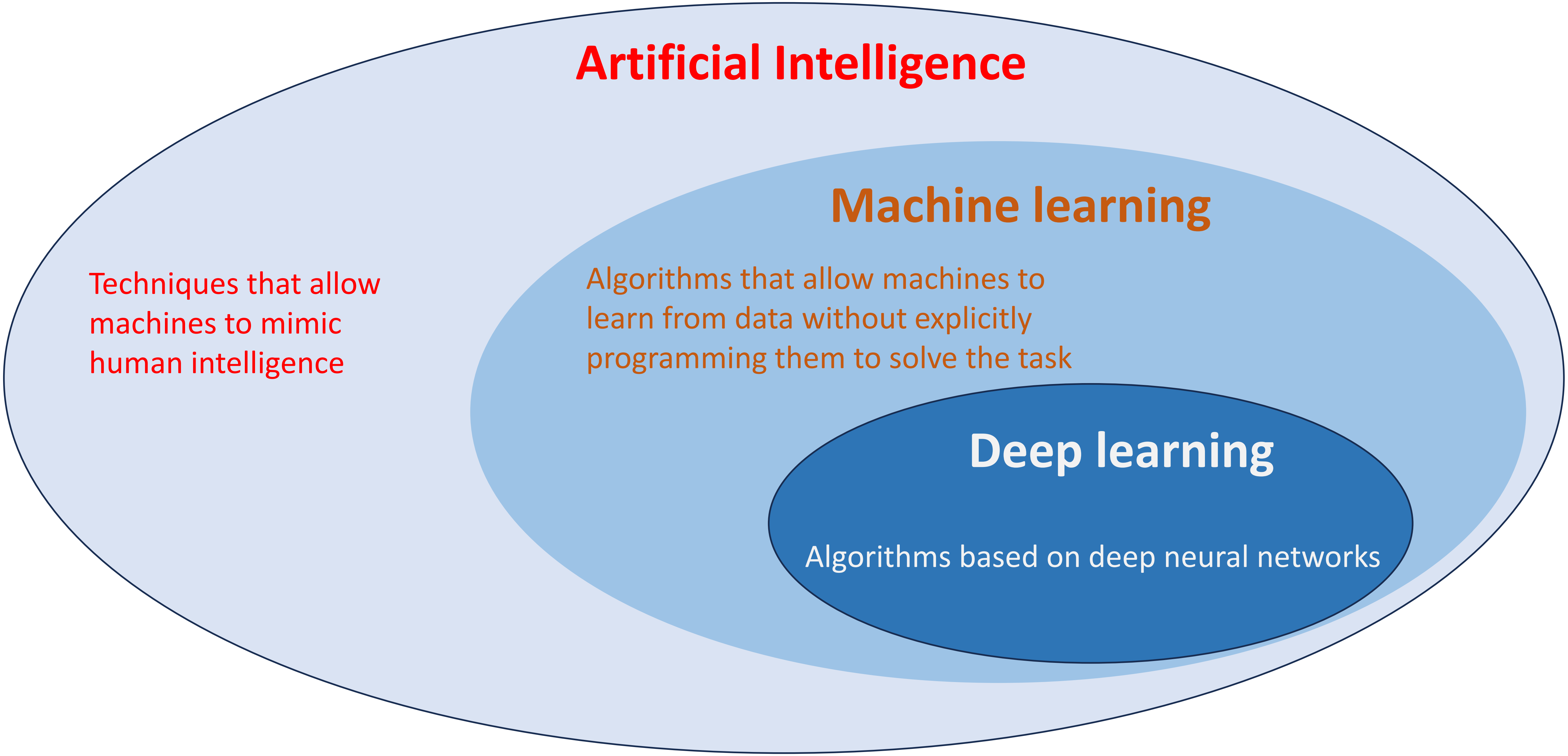
Dario Piga

SUPSI - Scuola Universitaria Professionale della Svizzera Italiana

IDSIA – Dalle Molle Institute for Artificial Intelligence

dario.piga@supsi.ch

Artificial Intelligence, Machine Learning, Deep Learning



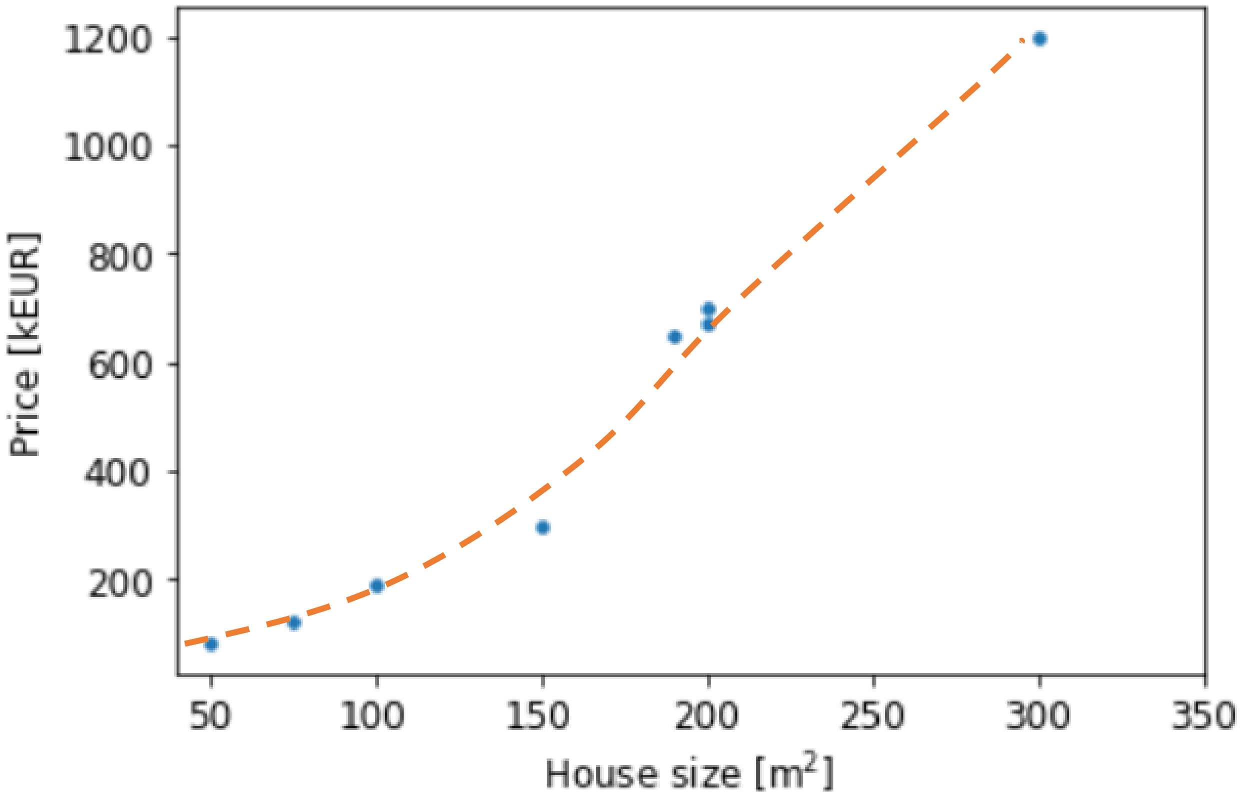
Machine learning: Regression

$\{(x^i, y^i)\}\quad i = 1, \dots, N$ available dataset

$\hat{y}^i = M(x^i; \theta)$ parametric model

$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(y^i - \hat{y}^i(\theta)\right)^2$ Loss (MSE)

Real estate application



$$\boxed{\theta^* = arg\min_{\theta} L(\theta)}$$

Machine learning: (binary) Classification

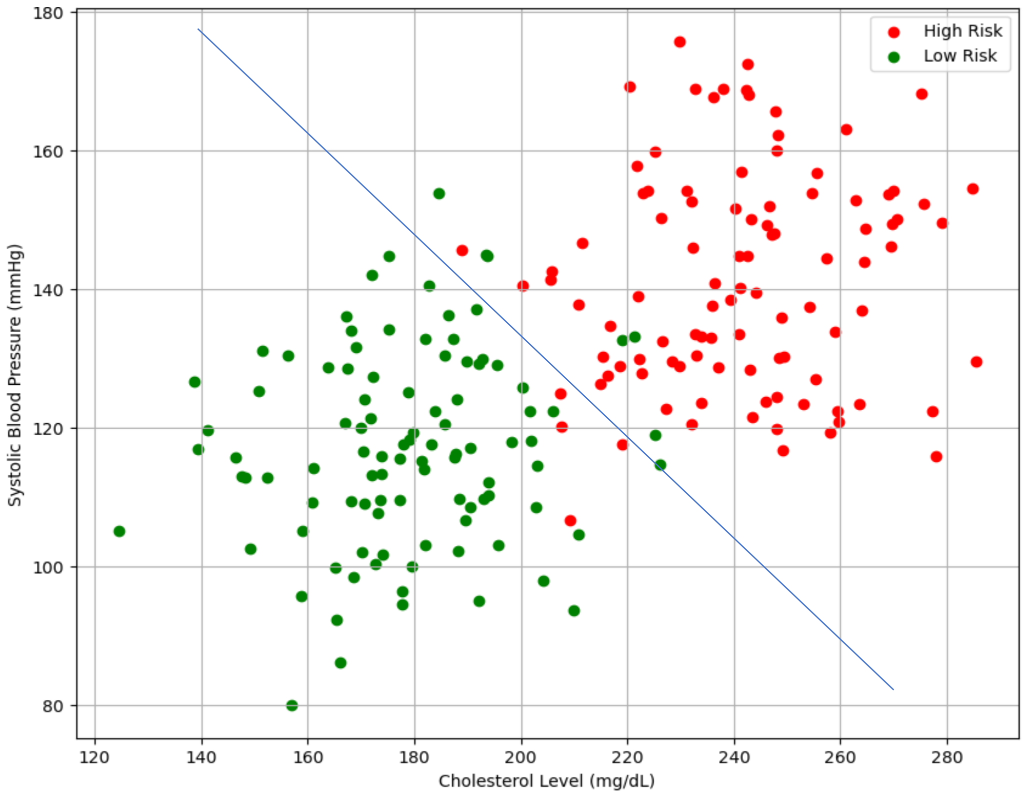
$\{0,1\}$

$\{(x^i, y^i)\}\ i = 1, \ldots, N$ available dataset

$p_i = M(x^i; \theta)$ parametric model

$$L(\theta) = -\sum_{i=1}^{N} y^i \log p_i(\theta) + (1 - y^i) \log(1 - p_i(\theta))$$

Cardiovascular risk classification



$$\boxed{\theta^* = arg\min_{\theta} L(\theta)}$$

Regression and Classification

- Choice of the model structure

$$\hat{y}^i = M(x^i; \theta)$$

$$p_i = M(x^i; \theta)$$

- How to optimize the loss?

$$\boxed{\theta^* = arg \min_{\theta} L(\theta)}$$

Linear regression

- Linear model $\hat{y} = M(x; \theta)$ $\hat{y} = \theta_0 + \sum_{j=1}^{d} \theta_j x_j = \sum_{j=0}^{d} \theta_j x_j = \theta^T x$

- Quadratic Loss $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y^i - \hat{y}^i(\theta)\right)^2$ $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y^i - \theta^T x^i\right)^2$

Easy problem with analytical solution: solve $\nabla_\theta L(\theta) = 0$

Towards nonlinear models

- Linear-in the parameter model

$$\hat{y} = \sum_{j=0}^{d} \theta_j \varphi_j(x) = \theta^T \varphi(x)$$

- Quadratic Loss

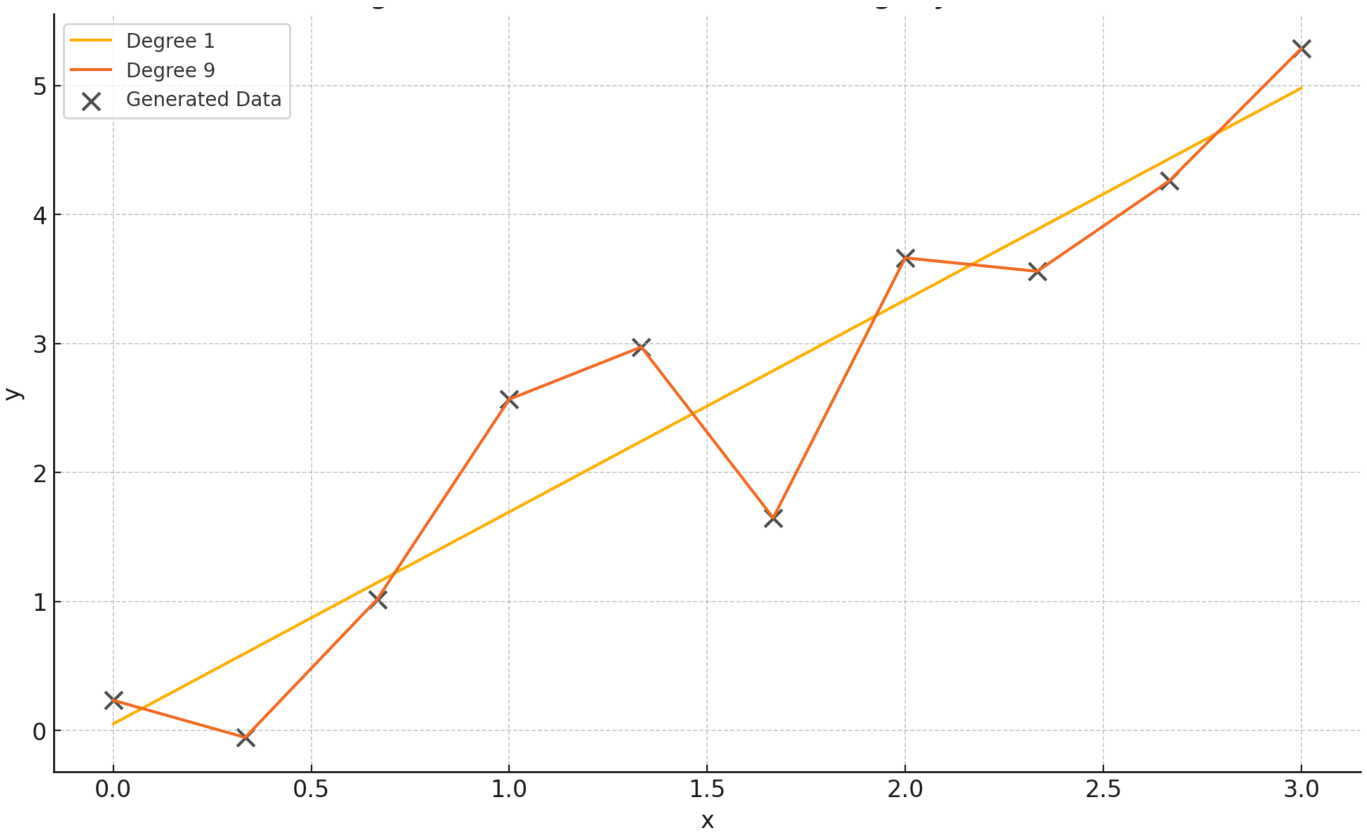
$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y^i - \theta^T \varphi(x^i)\right)^2$$

Still an easy problem with analytical solution: $\nabla_\theta L(\theta) = 0$

- How to select the basis functions?
- How to assess model performance?

Assess model performance

- Split your data into a training and test dataset
- Estimate the model based on training data and assess the performance in test
- Test data should be never used for model training
- Avoid overfitting: model perform well on training data, but not on test data



Assess model performance: metrics

- Plot true output vs predicted output
- RMSE (root mean square error): $\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y^i - \theta^T x^i)^2}$
- R^2 index: $1 - \frac{\sum_{i=1}^{N}(y^i - \theta^T x^i)^2}{\sum_{i=1}^{N}(y^i - \bar{y})^2}$
- and many others…

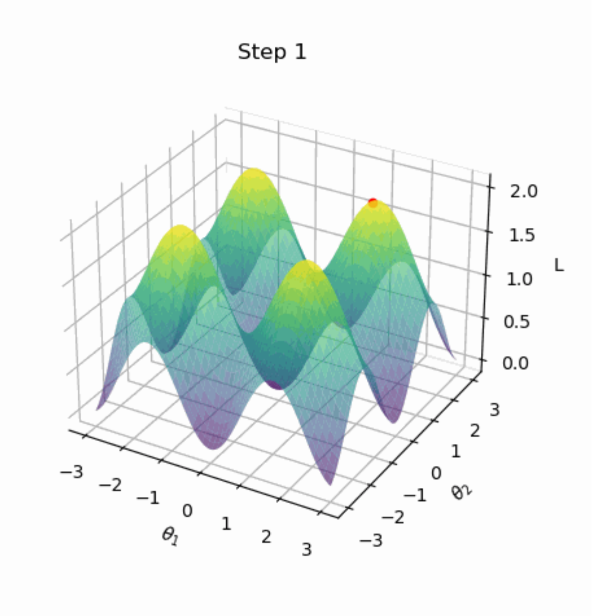
Good practice: always check performance both in training and test data

Gradient-based optimization

Vanilla Gradient Descent

1. **Initialize Parameters:** $\theta^{(0)}$
2. **for** $k = 0, 1, \ldots$: **until** maximum number of iterations or convergence **do**:
   (a) **Compute Gradient:** $\nabla_\theta \mathcal{L}(\theta^{(k)})$
   (b) **Update Parameters:**

$$\theta^{(k+1)} = \theta^{(k)} - \gamma \cdot \nabla_\theta \mathcal{L}(\theta^{(k)})$$




```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

Boston Housing (notebook)

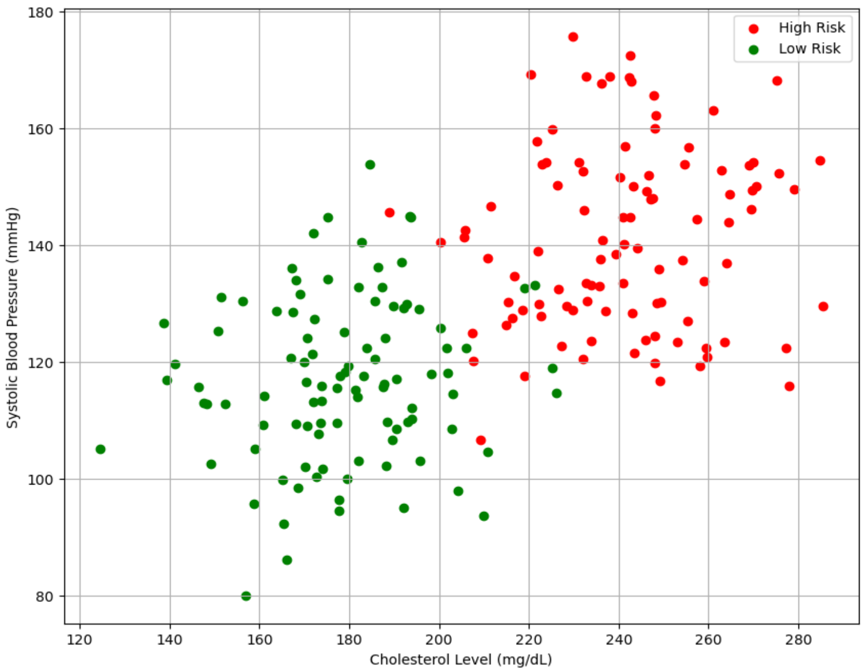
Logistic Regression for binary classification

$$\{0,1\}$$

$$\{(x^i, y^i)\}\ i = 1, \dots, N$$ available dataset

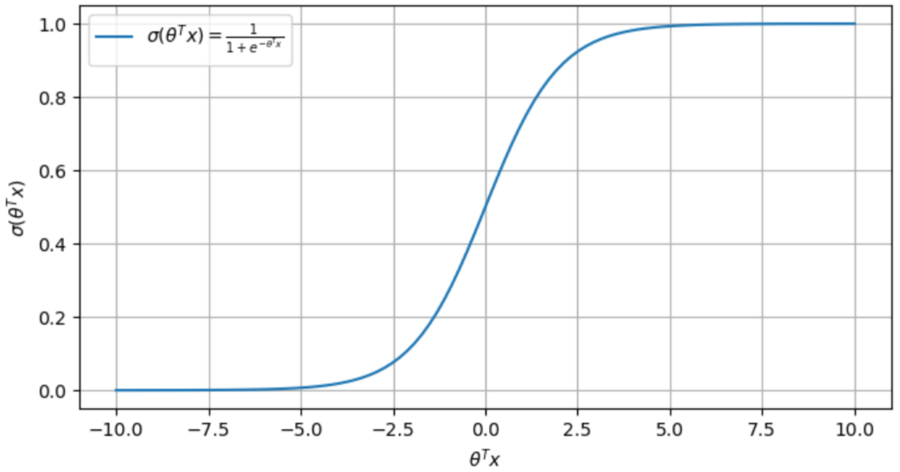
$$p_i = M(x^i; \theta)$$ parametric model

Cardiovascular risk classification



$$p = M(x; \theta) = \sigma(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

sigmoid



Logistic Regression for binary classification

$$p = M(x; \theta) = \sigma(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

$$L(\theta) = -\sum_{i}^{N} y^i \log p_i(\theta) + (1 - y^i) \log(1 - p_i(\theta))$$

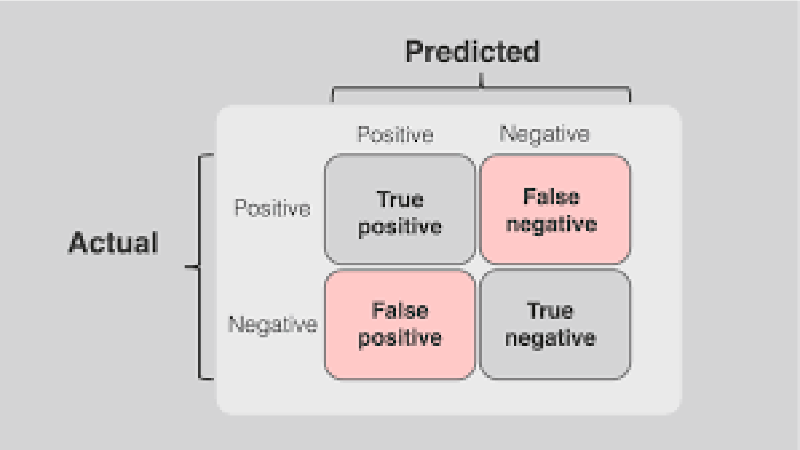
$$\boxed{\theta^* = arg \min_{\theta} L(\theta)}$$

- Inference: if $p_i = M(x_i; \theta) \geq 0.5 \ \rightarrow \ y_i = 1; \quad y_i = 0$ otherwise

Assess model performance: metrics

- Accuracy: $\frac{\#\ Correcly\ classified\ samples}{Total\ number\ of\ samples}$

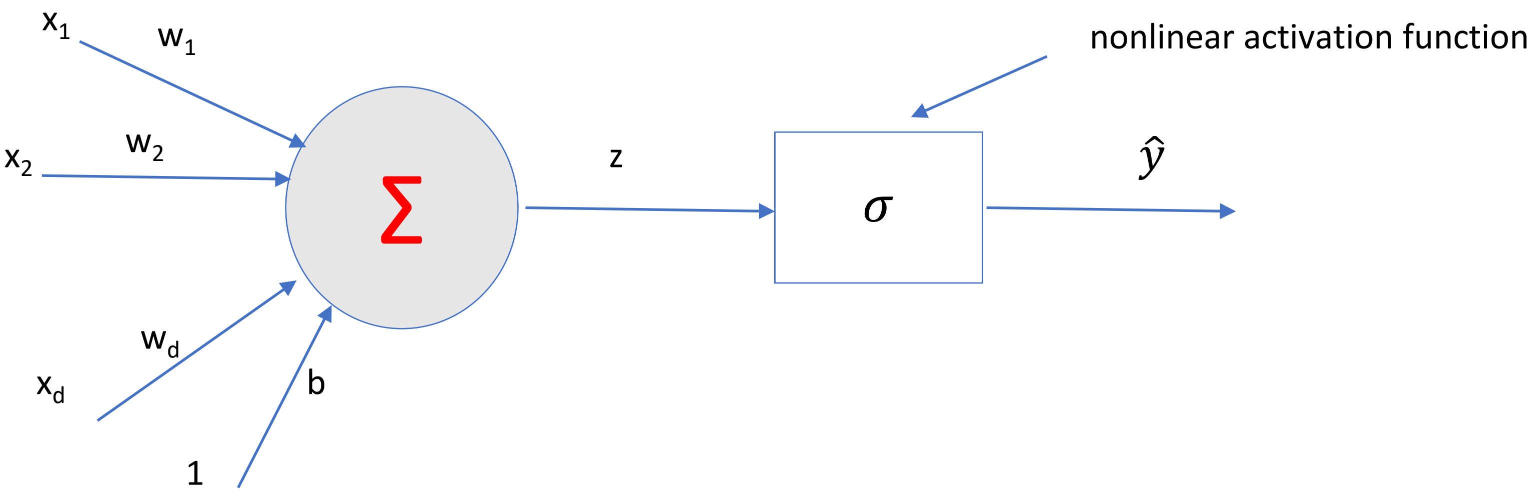
- Confusion matrix



Titanic dataset (notebook)

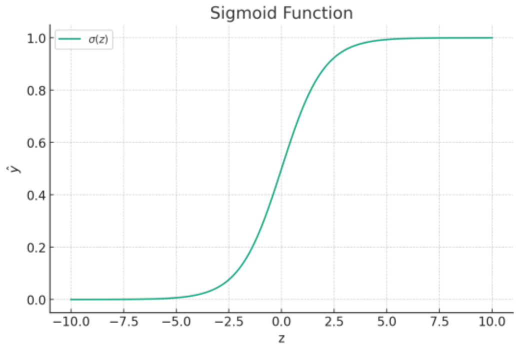
Deep Learning

Basic units for Neural Networks

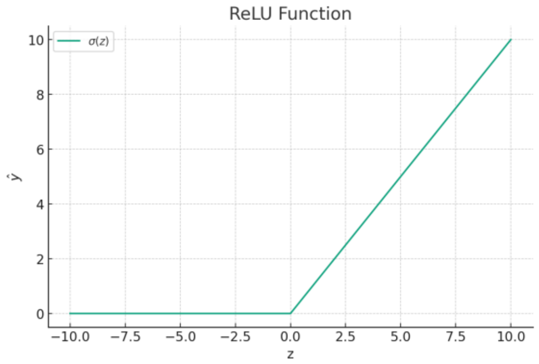


$$\hat{y} = \sigma\left(\sum_{j=1}^{n} w_j x_j + b\right)$$

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



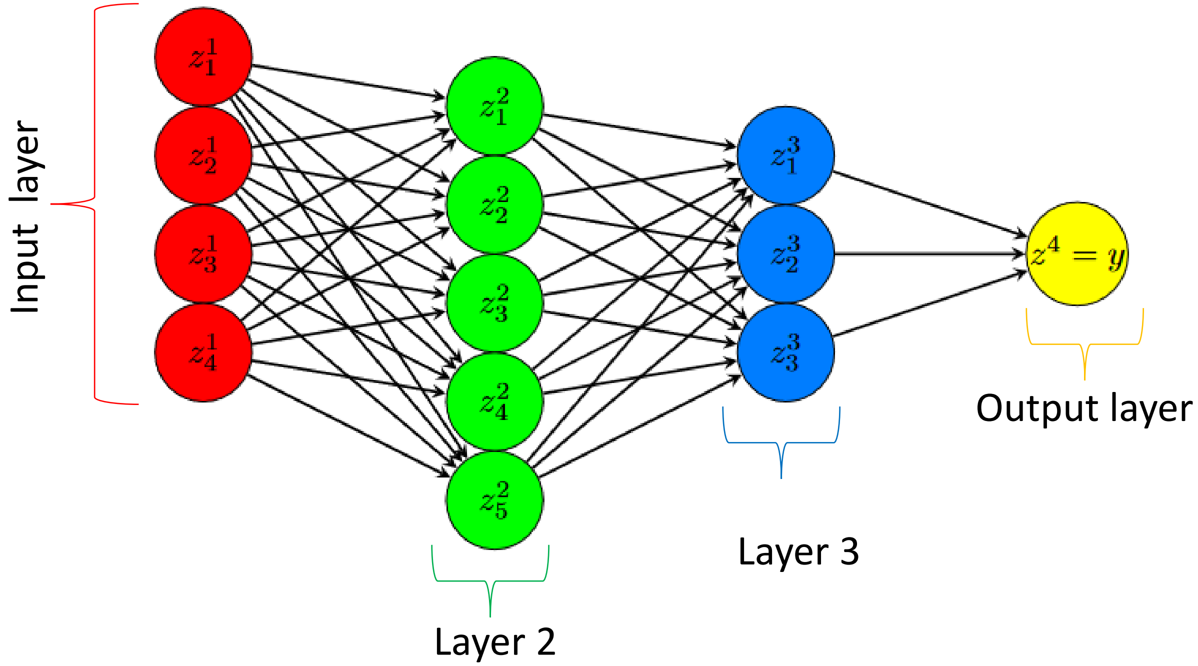
$$\sigma(z) = \begin{cases} z & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases}$$



We have a non-linear relation between inputs and outputs!

Fully-connected Feedforward Neural Networks

Move from one neuron to a hierarchical structure with fully connected neurons



$$z_1^2 = \sigma\left(\sum_{j=1}^{4} w_{1,j}^1 z_j^1 + b_1^1\right) = \sigma\left(W_1^1 z^1 + b_1^1\right)$$

$$z_2^3 = \sigma\left(\sum_{j=1}^{5} w_{2,j}^2 z_j^2 + b_2^2\right) = \sigma\left(W_2^2 z^2 + b_2^2\right)$$

$$y = z^4 = \sum_{j=1}^{3} w_{1,j}^3 z_j^3 + b^3 = W_1^3 z^3 + b^3$$

$$y = z^4 = f(z^3(z^2(z^1)); W, b)$$

- **For regression problems, the output node typically does not have a nonlinear activation function**
- **For binary classification problems, the output node has a sigmoid as an activation function**

FFN: PyTorch

```
import torch.nn.functional as F

class FeedforwardNN(nn.Module):
    def __init__(self, input_dim=32*32*3, hidden1=512, hidden2=256, num_classes=10):
        super(FeedforwardNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden2, num_classes)


    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the image
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        z = self.fc3(x)
        return z
```

```
model = FeedforwardNN(input_dim = X_train.shape[1]*X_train.shape[2]*X_train.shape[3], hidden1=254, hidden2=64, num_classes=10)

print(f"Model structure: {model}")

for name, params in model.named_parameters():
    print(f"parameter name: {name}. Value {params.data}")

# check what model provides:
y_hat = model(X_train)
```

FFN: Training in PyTorch

```python
# Define the loss function
criterion = nn.CrossEntropyLoss()  # Multi-class classification loss

# define optimizer
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)
```

```python
# Training loop with accuracy calculation
max_epochs = 8000
for it in range(max_epochs):
    optimizer.zero_grad()
    i = np.random.randint(0, X_train.shape[0], size = 1024) # mini-batch implementation
    z_hat = model(X_train[i,:])  # Logit Predictions
    loss = criterion(z_hat, y_train[i])  # Compute loss
    loss.backward()
    optimizer.step()

    if it % 5 == 0:  # Print every 5 iterations
        with torch.no_grad():
            z_hat = model(X_train)
            predicted_labels = torch.argmax(z_hat, dim=1)  # Get predicted class
            accuracy = (predicted_labels == y_train).float().mean().item()  # Compute accuracy

        print(f"Iteration: {it}. Loss: {loss.item():.3f}, Accuracy: {accuracy:.2%}")
```