# Introduksjon og lineær regresjon

Ole Christian Eidheim

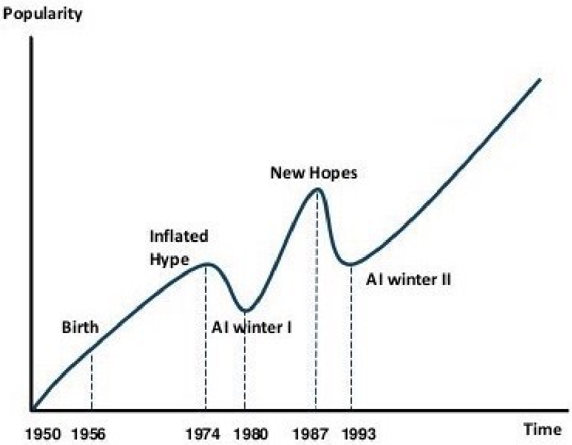
AIT, IDI, NTNU

25. august 2021

# Oversikt

- **Introduksjon**
- Lineær regresjon
- Øving 1

# AI historie




**Timeline of AI Development**

- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2011**: IBM's Watson won Jeopardy
- **2012: AlexNet won ImageNet competition by a large margin**

**But still far away from human like intelligence!**

# Tradisjonell programmering vs maskinlæring



- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(\boldsymbol{x})$ der input $\boldsymbol{x}$ gir svaret $\boldsymbol{y}$:
    - $\boldsymbol{y} = f(\boldsymbol{x})$

# Tradisjonell programmering vs maskinlæring

## - Oversikt over maskinlæring notasjon

- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(\boldsymbol{x})$ der input $\boldsymbol{x}$ gir svaret $\boldsymbol{y}$:
    - $\boldsymbol{y} = f(\boldsymbol{x})$
- Maskinlæring (regresjon og klassifikasjon):
  - Gitt $N$ observasjoner $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}), i = 1, ..., N$, bygger vi opp en modell $f(\boldsymbol{x})$ ved hjelp av utvalgte maskinlæringsmetoder.
  - Gjennom en automatisk operasjon kalt *optimalisering* blir interne variabler i modellen $f(\boldsymbol{x})$ justert slik at input $\boldsymbol{x}^{(i)}$ gir et resultat som er tilnærmet likt $\boldsymbol{y}^{(i)}$:
    - $\boldsymbol{y}^{(i)} \approx f(\boldsymbol{x}^{(i)})$

# Tradisjonell programmering vs maskinlæring
## - Oversikt over maskinlæring notasjon

- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(\boldsymbol{x})$ der input $\boldsymbol{x}$ gir svaret $\boldsymbol{y}$:
    - $\boldsymbol{y} = f(\boldsymbol{x})$
- Maskinlæring (regresjon og klassifikasjon):
  - Gitt $N$ observasjoner $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}), i = 1, ..., N$, bygger vi opp en modell $f(\boldsymbol{x})$ ved hjelp av utvalgte maskinlæringsmetoder.
  - Gjennom en automatisk operasjon kalt *optimalisering* blir interne variabler i modellen $f(\boldsymbol{x})$ justert slik at input $\boldsymbol{x}^{(i)}$ gir et resultat som er tilnærmet likt $\boldsymbol{y}^{(i)}$:
    - $\boldsymbol{y}^{(i)} \approx f(\boldsymbol{x}^{(i)})$
  - En *tapsfunksjon* er brukt for å styre optimaliseringen. Denne funksjonen indikerer hvor godt tilpasset en modell $f(\boldsymbol{x})$ er for gitte observasjoner $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$.

# Tradisjonell programmering vs maskinlæring

## - Oversikt over maskinlæring notasjon

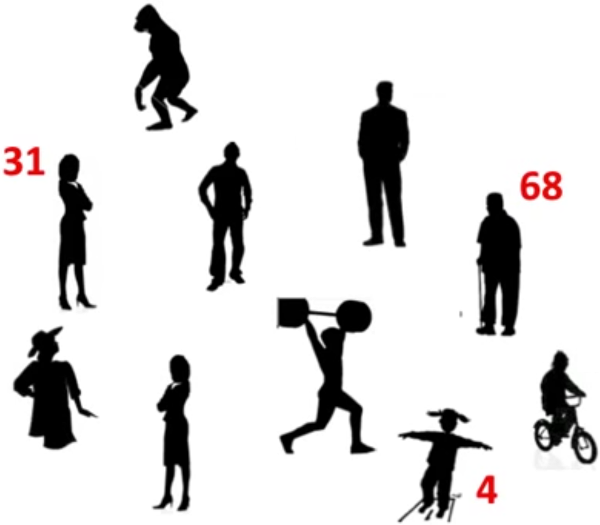
- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(\boldsymbol{x})$ der input $\boldsymbol{x}$ gir svaret $\boldsymbol{y}$:
    - $\boldsymbol{y} = f(\boldsymbol{x})$
- Maskinlæring (regresjon og klassifikasjon):
  - Gitt $N$ observasjoner $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}), i = 1, ..., N$, bygger vi opp en modell $f(\boldsymbol{x})$ ved hjelp av utvalgte maskinlæringsmetoder.
  - Gjennom en automatisk operasjon kalt *optimalisering* blir interne variabler i modellen $f(\boldsymbol{x})$ justert slik at input $\boldsymbol{x}^{(i)}$ gir et resultat som er tilnærmet likt $\boldsymbol{y}^{(i)}$:
    - $\boldsymbol{y}^{(i)} \approx f(\boldsymbol{x}^{(i)})$
  - En *tapsfunksjon* er brukt for å styre optimaliseringen. Denne funksjonen indikerer hvor godt tilpasset en modell $f(\boldsymbol{x})$ er for gitte observasjoner $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$.
  - Til slutt kan modellen brukes til å gjøre en predikasjon $\hat{\boldsymbol{y}}$ fra en ny observasjon $\boldsymbol{x}$:
    - $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$

# Tradisjonell programmering vs maskinlæring
## - Oversikt over maskinlæring notasjon

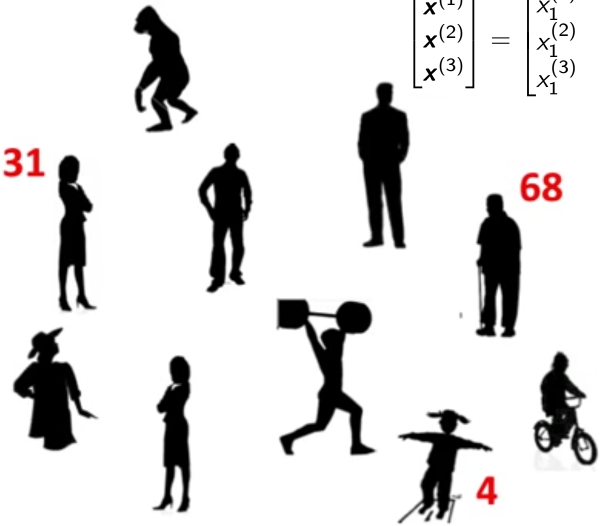
- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(\boldsymbol{x})$ der input $\boldsymbol{x}$ gir svaret $\boldsymbol{y}$:
    - $\boldsymbol{y} = f(\boldsymbol{x})$
- Maskinlæring (regresjon og klassifikasjon):
  - Gitt $N$ observasjoner $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}), i = 1, ..., N$, bygger vi opp en modell $f(\boldsymbol{x})$ ved hjelp av utvalgte maskinlæringsmetoder.
  - Gjennom en automatisk operasjon kalt *optimalisering* blir interne variabler i modellen $f(\boldsymbol{x})$ justert slik at input $\boldsymbol{x}^{(i)}$ gir et resultat som er tilnærmet likt $\boldsymbol{y}^{(i)}$:
    - $\boldsymbol{y}^{(i)} \approx f(\boldsymbol{x}^{(i)})$
  - En *tapsfunksjon* er brukt for å styre optimaliseringen. Denne funksjonen indikerer hvor godt tilpasset en modell $f(\boldsymbol{x})$ er for gitte observasjoner $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$.
  - Til slutt kan modellen brukes til å gjøre en predikasjon $\hat{\boldsymbol{y}}$ fra en ny observasjon $\boldsymbol{x}$:
    - $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$
  - Observasjonene deles ofte opp i to datasett: *treningsdata* og *testdata*
    - *treningsdata* $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ blir brukt i optimaliseringen
    - *testdata* blir brukt til å måle nøyaktigheten av modellen (hvor god modellen er) etter optimalisering

# Typper maskinlæringsmetoder - regresjon

# Typer maskinlæringsmetoder
# - regresjon



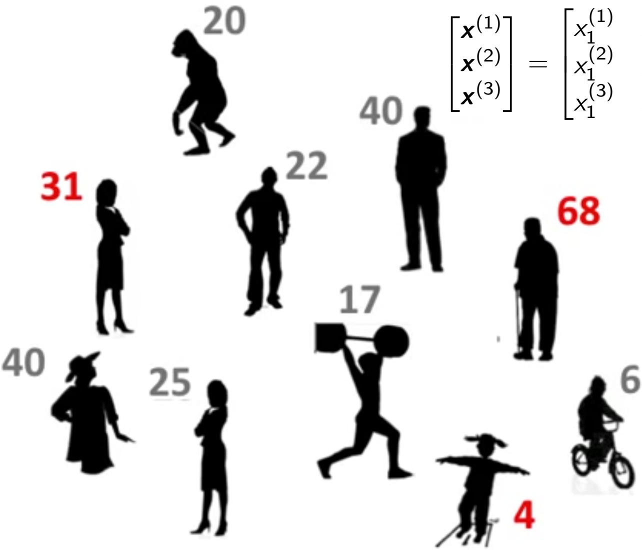
# Typer maskinlæringsmetoder
# - regresjon




$$\begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \boldsymbol{x}^{(3)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_{\dots}^{(1)} & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_{\dots}^{(2)} & x_D^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_{\dots}^{(3)} & x_D^{(3)} \end{bmatrix} =$$

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \end{bmatrix} = \begin{bmatrix} 31 \\ 68 \\ 4 \end{bmatrix}$$

# Typer maskinlæringsmetoder
# - regresjon




$$\begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \boldsymbol{x}^{(3)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_{\dots}^{(1)} & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_{\dots}^{(2)} & x_D^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_{\dots}^{(3)} & x_D^{(3)} \end{bmatrix} =$$

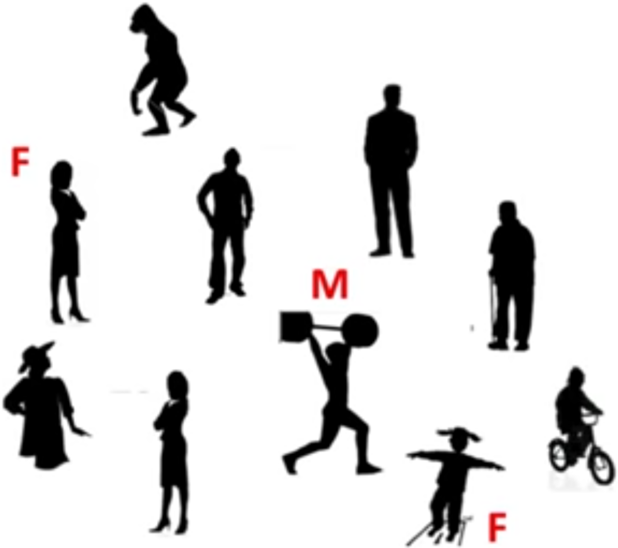
$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \end{bmatrix} = \begin{bmatrix} 31 \\ 68 \\ 4 \end{bmatrix}$$

$f($  $) = 6$

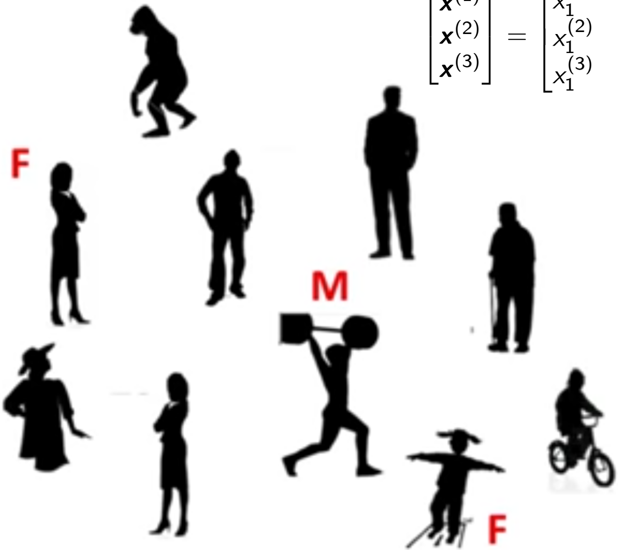
$f($  $) = 17$

$f($  $) = 20$

# Typer maskinlæringsmetoder
# - klassifikasjon



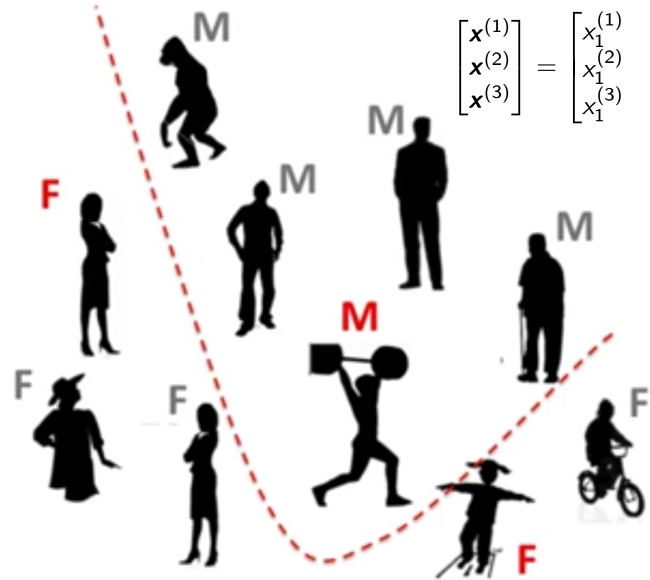
# Typer maskinlæringsmetoder
# - klassifikasjon




$$\begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \boldsymbol{x}^{(3)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_{\dots}^{(1)} & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_{\dots}^{(2)} & x_D^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_{\dots}^{(3)} & x_D^{(3)} \end{bmatrix} =$$

$$\begin{bmatrix} \boldsymbol{y}^{(1)} \\ \boldsymbol{y}^{(2)} \\ \boldsymbol{y}^{(3)} \end{bmatrix} = \begin{bmatrix} F \\ M \\ F \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

# Typer maskinlæringsmetoder
# - klassifikasjon




$$\begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \boldsymbol{x}^{(3)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_{\dots}^{(1)} & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_{\dots}^{(2)} & x_D^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_{\dots}^{(3)} & x_D^{(3)} \end{bmatrix} =$$

$$\begin{bmatrix} \boldsymbol{y}^{(1)} \\ \boldsymbol{y}^{(2)} \\ \boldsymbol{y}^{(3)} \end{bmatrix} = \begin{bmatrix} F \\ M \\ F \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$f($  $) \approx \begin{bmatrix} 0.7 & 0.3 \end{bmatrix} \approx F$

$f($  $) \approx \begin{bmatrix} 0.4 & 0.6 \end{bmatrix} \approx M$

$f($  $) \approx \begin{bmatrix} 0.3 & 0.7 \end{bmatrix} \approx M$

# Typer maskinlæringsmetoder
# - clustering

# Oversikt

- Introduksjon
- **Lineær regresjon**
- Øving 1

# Lineær regresjon i PyTorch

- Modell variabler (også kalt modell parametre):
  $\boldsymbol{W}$ og $\boldsymbol{b}$
- Modell prediktor:
  $f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$
- N observasjoner (treningsdata):
  $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}), i = 1, ..., N$
- Tapsfunksjon (Mean squared error):
  $loss = \frac{1}{N}\sum_{i=1}^{N}(f(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)})^2$
- Optimalisering:
  Justering av $\boldsymbol{W}$ og $\boldsymbol{b}$, for å minske *loss*, gjennom *Gradient descent*
  - Hyperparametre (verdier satt av utvikleren), for eksempel:
    - Læringsrate (*learning rate*): hvor store steg som skal tas i *Gradient descent* metoden i optimaliseringen
    - Epoker (*epochs*): antall ganger hele treningsdatasettet skal brukes i optimaliseringen

# Oppsett av modell med tapsfunksjoner

```python
class LinearRegressionModel:
    def __init__(self):
        # requires_grad enables calculation of gradients
        self.W = torch.tensor([[0.0]], requires_grad=True)
        self.b = torch.tensor([[0.0]], requires_grad=True)

    # Predictor
    def f(self, x):
        return x @ self.W + self.b

    # Uses Mean Squared Error
    def loss(self, x, y):
        return torch.mean(torch.square(self.f(x) - y))
```

# Lineær regresjon - eksempler

- Interaktive visualiseringer for å bedre forståelse:
  - https://gitlab.com/ntnu-tdat3025/regression/visualize
  - **Ikke se på den rotete kildekoden!**
- Optimalisering med PyTorch og visualisering gjennom Matplotlib:
  - https://gitlab.com/ntnu-tdat3025/regression/linear-2d
  - **Les README filen først**
  - Denne kildekoden er grei, og kan fungere som utgangspunkt til øvingen.

# Oversikt

- Introduksjon
- Lineær regresjon
- **Øving 1**

# PyTorch tips knyttet til øvingen

- Hvis du får feilaktige verdier av $\boldsymbol{W}$ og $\boldsymbol{b}$, prøv:
  - Minske læringsraten i `torch.optim.SGD`
  - Øke antall *epoker*
  - Endre tapsfunksjonen, `LinearRegressionModel.loss`, til å bruke innebygd PyTorch funksjon i stedet:`torch.nn.functional.mse_loss`
    - Innebygde PyTorch funksjoner kan være mer numerisk stabile

# Installasjon av nødvendige Python pakker

- Forutsetning: Linux eller MacOS
  - Noen brukte Windows i fjor, men eksempler og løsningsforslag blir ikke testet på Windows
    - Windows er lite brukt i dette fagfeltet
  - Anbefaler Manjaro Linux som er basert på Arch Linux
    - Nyeste biblioteker og programvare
    - Svært god dokumentasjon: https://wiki.archlinux.org/
- Installasjon av nødvendige Python3 pakker:
  - Arch Linux basert distribusjon
    - `sudo pacman -S python-numpy python-matplotlib python-pytorch`
  - MacOS eller andre Linux distribusjoner:
    - `pip3 install numpy matplotlib torch torchvision`
- Spesielt MacOS: hvis python2 er *default*, kjør `.py` filene med `python3`

# Oppsett av Python IDE/Jupyter Notebook

- Bruk et valgfritt IDE eller Jupyter Notebook
- Donn skal senere vise dere Jupyter Notebook

# Øving 1

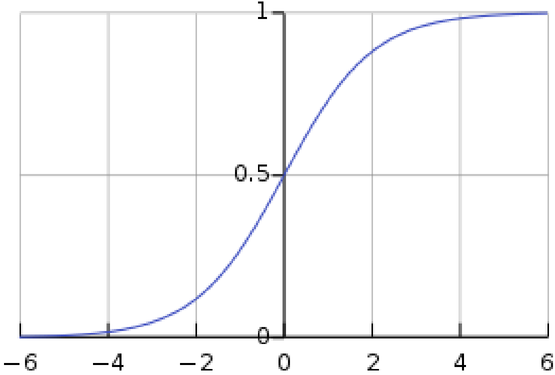
Ta gjerne utgangspunkt i linear-2d.
Datasettene i deloppgavene inneholder observasjoner om nyfødte barn. Det kan være til hjelp å visualisere observasjonene først.
**For alle deloppgavene**: du skal visualisere modellen etter optimalisering sammen med observasjonene, og skrive ut tapsverdien (*loss*) for modellen.

a) Lineær regresjon i 2 dimensjoner:
   - Lag en lineær modell som predikerer vekt ut fra lengde gitt observasjonene i length_weight.csv

b) Lineær regresjon i 3 dimensjoner:
   - Lag en lineær modell som predikerer alder (i dager) ut fra lengde og vekt gitt observasjonene i day_length_weight.csv
   - 3D plotting kan være litt uvant, men se eksempler på matplotlib.org.

c) Ikke-lineær regresjon i 2 dimensjoner (**se neste side**):
   - Lag en ikke-lineær modell som predikerer hodeomkrets ut fra alder (i dager) gitt observasjonene i day_head_circumference.csv
     - Bruk følgende modell prediktor: $f(x) = 20\sigma(xW + b) + 31$, der $\sigma$ er sigmoid funksjonen som definert på neste slide.

# Øving 1
# - sigmoid funksjonen




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