Introduksjon og lineær regresjon

Ole Christian Eidheim

Institutt for informatikk og e-læring,
NTNU

17. august 2020

Oversikt

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

Tradisjonell programmering vs maskinlæring

- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(x)$ der input x gir resultatet y:
    - $y = f(x)$

Tradisjonell programmering vs maskinlæring

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

Tradisjonell programmering vs maskinlæring

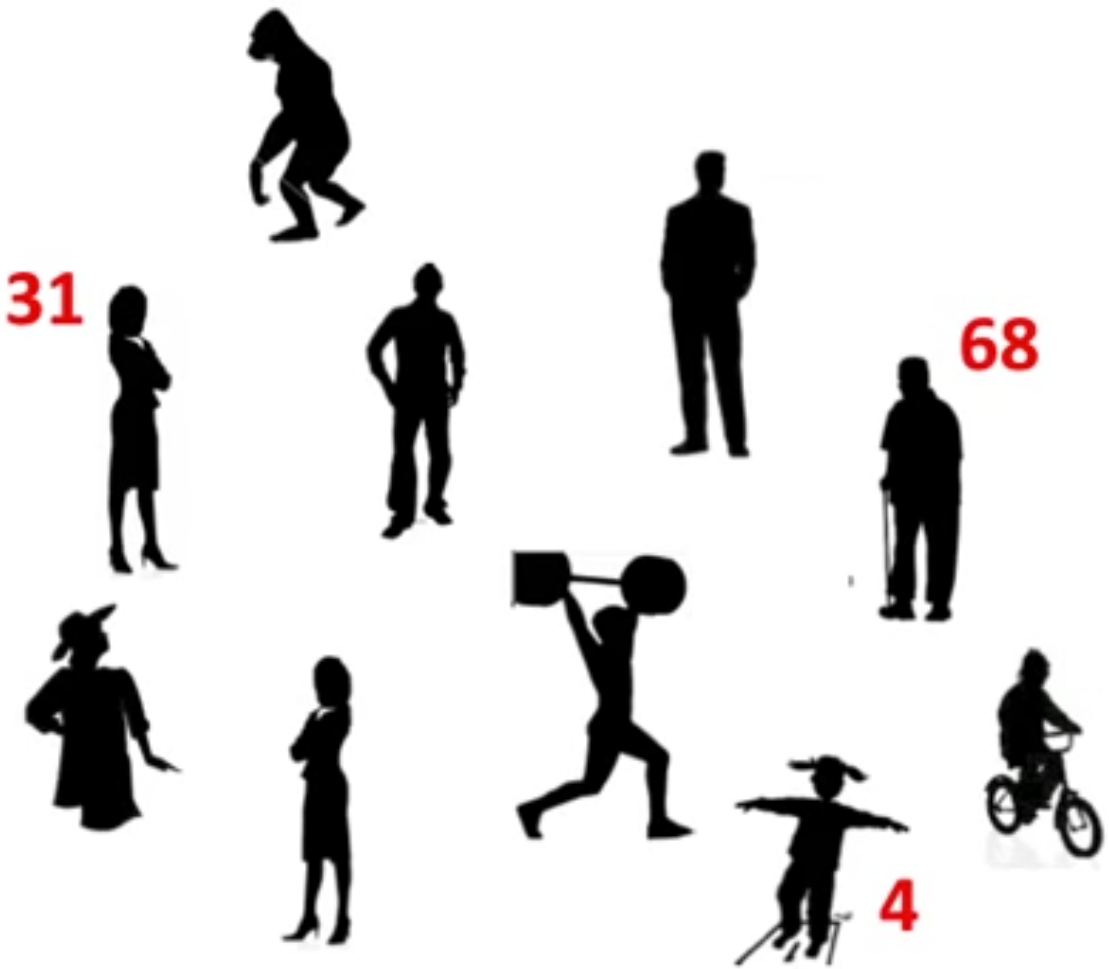
- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(x)$ der input x gir resultatet y:
    - $y = f(x)$
- Maskinlæring (regresjon og klassifikasjon):
  - Gitt observasjoner $(\hat{x}, \hat{y})$, bygger vi opp en modell $f(x)$ ved hjelp av utvalgte maskinlæringsmetoder.
  - Gjennom en automatisk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

Tradisjonell programmering vs maskinlæring

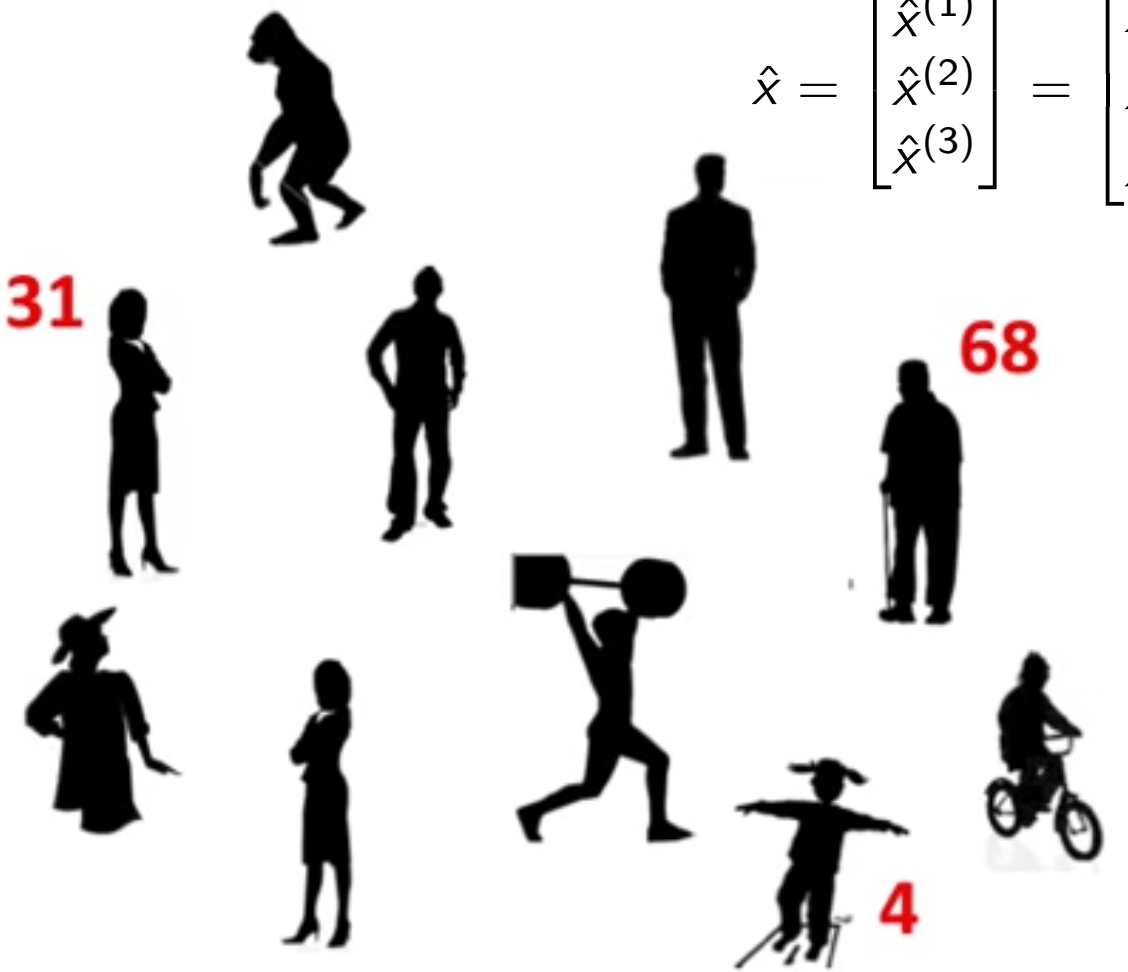
- Tradisjonell programmering:
  - Vi programmerer en funksjon $f(x)$ der input x gir resultatet y:
    - $y = f(x)$
- Maskinlæring (regresjon og klassifikasjon):
  - Gitt observasjoner $(\hat{x}, \hat{y})$, bygger vi opp en modell $f(x)$ ved hjelp av utvalgte maskinlæringsmetoder.
  - Gjennom en automatisk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

Typer maskinlæringsmetoder
- regresjon

Typer maskinlæringsmetoder
- regresjon



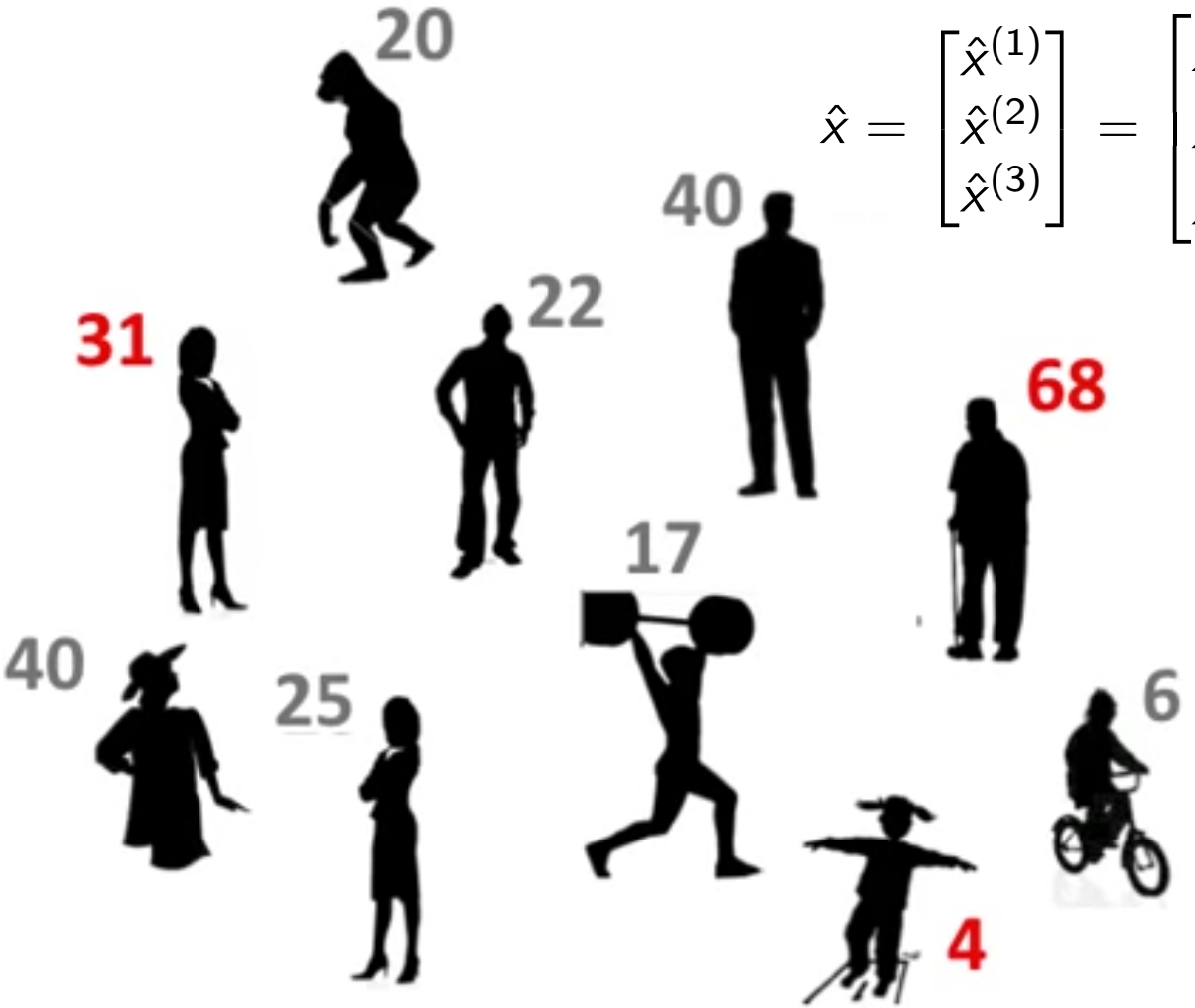
Typer maskinlæringsmetoder
- regresjon



$$\hat{x} = \begin{bmatrix} \hat{x}^{(1)} \\ \hat{x}^{(2)} \\ \hat{x}^{(3)} \end{bmatrix} = \begin{bmatrix} \hat{x}_1^{(1)} & \hat{x}_2^{(1)} & \hat{x}_{\dots}^{(1)} & \hat{x}_n^{(1)} \\ \hat{x}_1^{(2)} & \hat{x}_2^{(2)} & \hat{x}_{\dots}^{(2)} & \hat{x}_n^{(2)} \\ \hat{x}_1^{(3)} & \hat{x}_2^{(3)} & \hat{x}_{\dots}^{(3)} & \hat{x}_n^{(3)} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

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

Typer maskinlæringsmetoder
- regresjon



$$\hat{x} = \begin{bmatrix} \hat{x}^{(1)} \\ \hat{x}^{(2)} \\ \hat{x}^{(3)} \end{bmatrix} = \begin{bmatrix} \hat{x}_1^{(1)} & \hat{x}_2^{(1)} & \hat{x}_{\dots}^{(1)} & \hat{x}_n^{(1)} \\ \hat{x}_1^{(2)} & \hat{x}_2^{(2)} & \hat{x}_{\dots}^{(2)} & \hat{x}_n^{(2)} \\ \hat{x}_1^{(3)} & \hat{x}_2^{(3)} & \hat{x}_{\dots}^{(3)} & \hat{x}_n^{(3)} \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix}$$

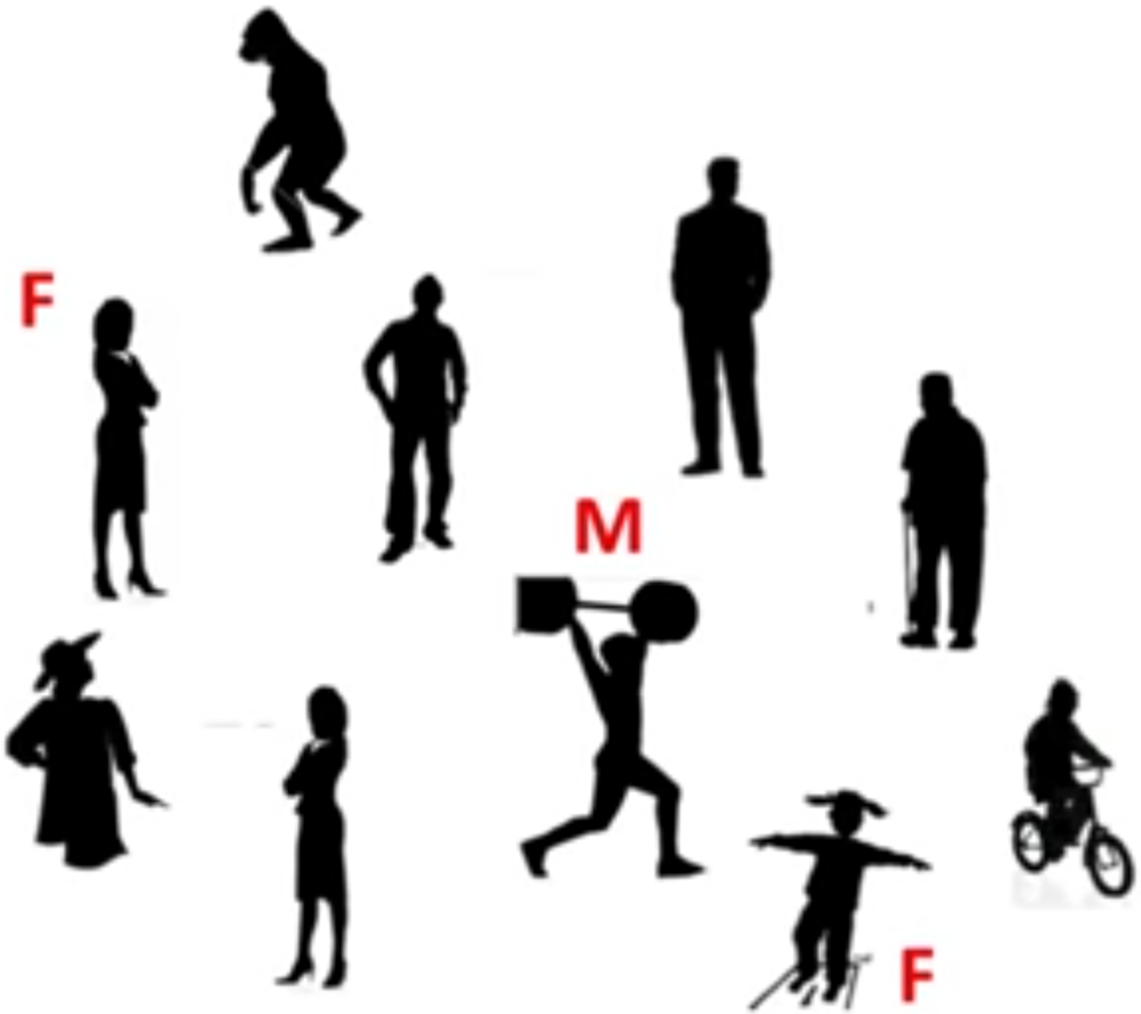
$$\hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \end{bmatrix} = \begin{bmatrix} 31 \\ 68 \\ 4 \end{bmatrix}$$

$f(\,) = 6$

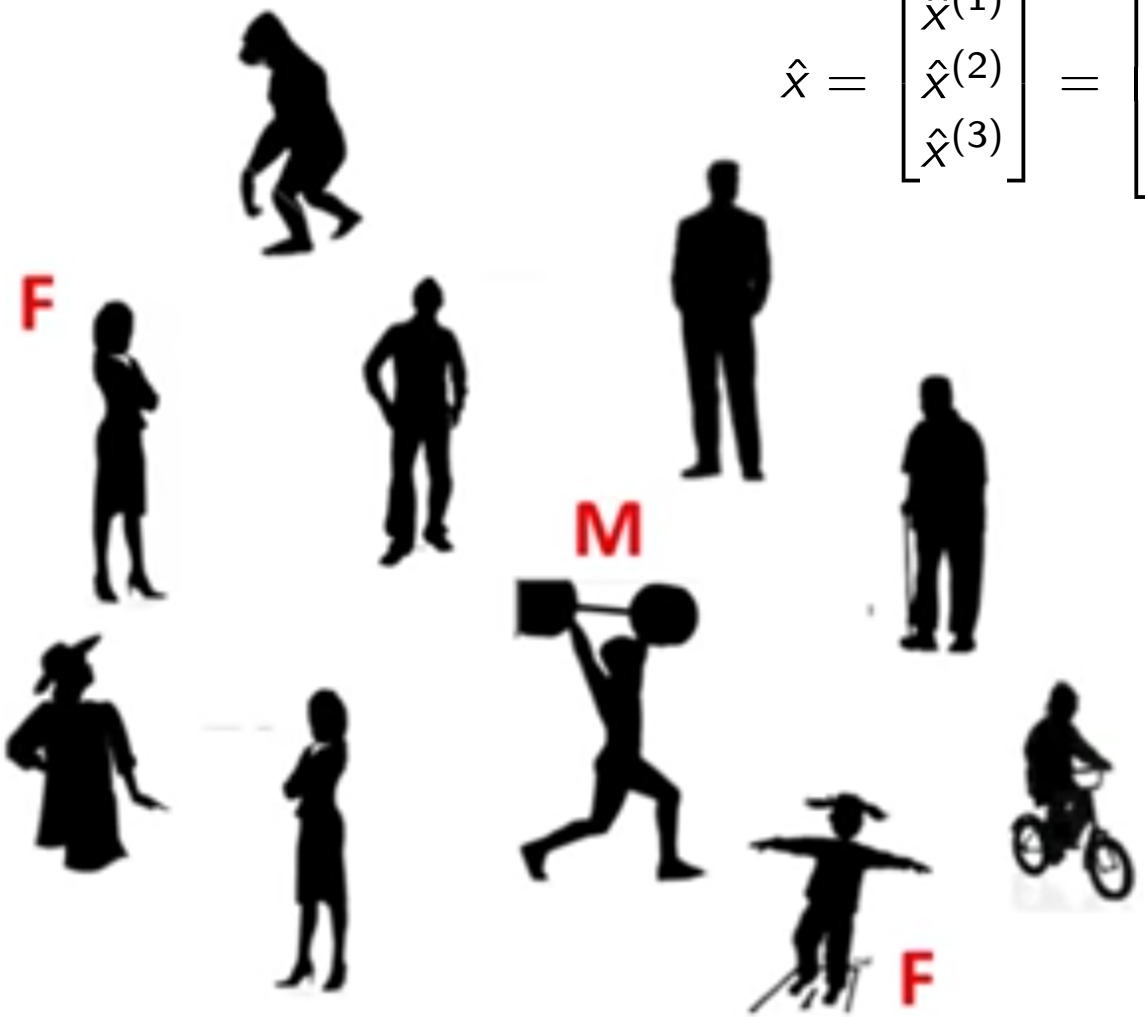
$f(\,) = 17$

$f(\,) = 20$

Typer maskinlæringsmetoder
- klassifikasjon



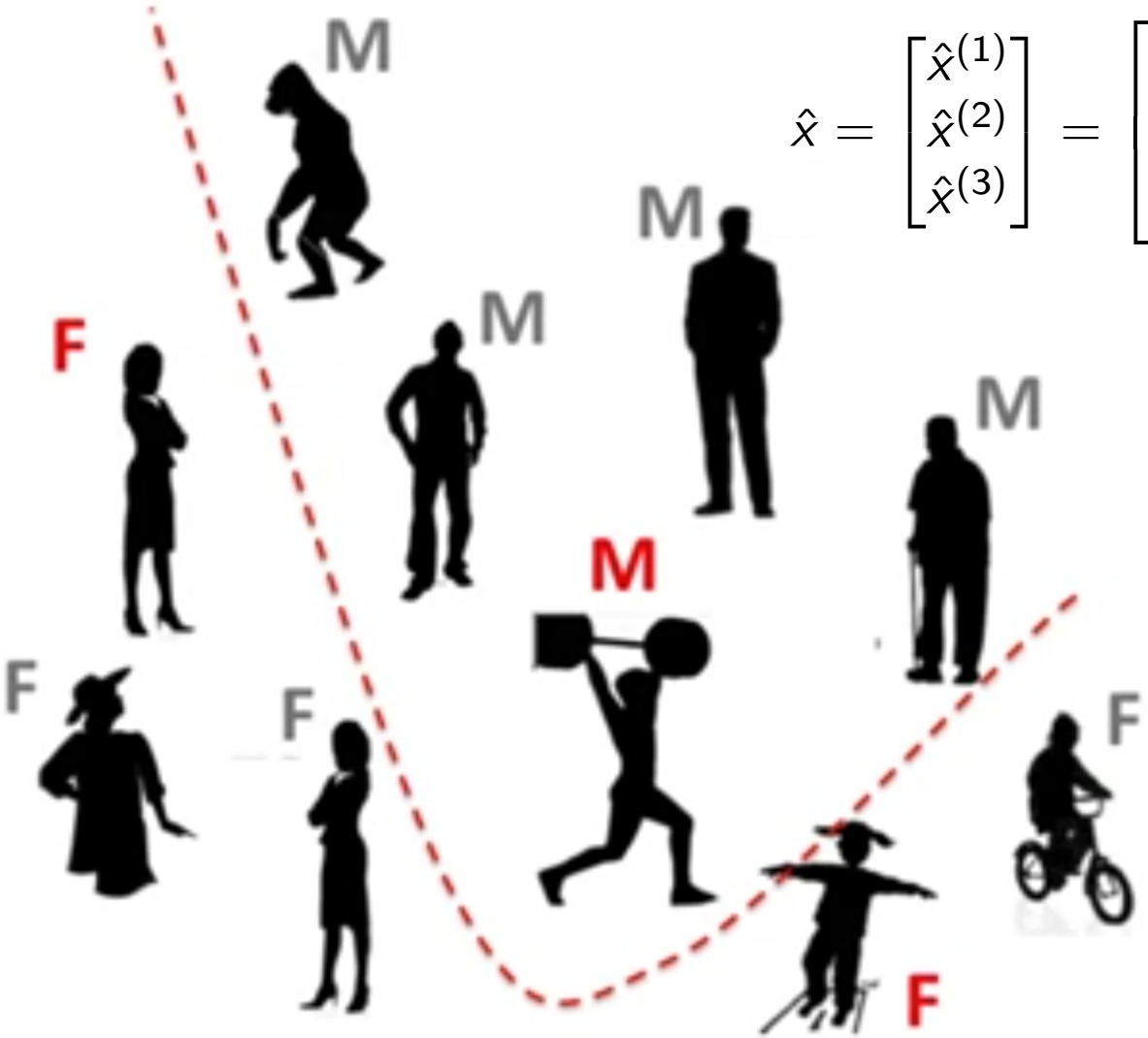
Typer maskinlæringsmetoder
- klassifikasjon



$$\hat{x} = \begin{bmatrix} \hat{x}^{(1)} \\ \hat{x}^{(2)} \\ \hat{x}^{(3)} \end{bmatrix} = \begin{bmatrix} \hat{x}_1^{(1)} & \hat{x}_2^{(1)} & \hat{x}_{\dots}^{(1)} & \hat{x}_n^{(1)} \\ \hat{x}_1^{(2)} & \hat{x}_2^{(2)} & \hat{x}_{\dots}^{(2)} & \hat{x}_n^{(2)} \\ \hat{x}_1^{(3)} & \hat{x}_2^{(3)} & \hat{x}_{\dots}^{(3)} & \hat{x}_n^{(3)} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

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

Typer maskinlæringsmetoder
- klassifikasjon



$$\hat{x} = \begin{bmatrix} \hat{x}^{(1)} \\ \hat{x}^{(2)} \\ \hat{x}^{(3)} \end{bmatrix} = \begin{bmatrix} \hat{x}_1^{(1)} & \hat{x}_2^{(1)} & \hat{x}_{\dots}^{(1)} & \hat{x}_n^{(1)} \\ \hat{x}_1^{(2)} & \hat{x}_2^{(2)} & \hat{x}_{\dots}^{(2)} & \hat{x}_n^{(2)} \\ \hat{x}_1^{(3)} & \hat{x}_2^{(3)} & \hat{x}_{\dots}^{(3)} & \hat{x}_n^{(3)} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

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

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

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

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

Typer maskinlæringsmetoder
- clustering

Oversikt

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

Lineær regresjon i PyTorch

- Modell variabler:
  W og b
- Modell prediktor:
  $f(x) = xW + b$
- Observasjoner:
  $(\hat{x}, \hat{y})$
- Tapsfunksjon (Mean squared error):
  $loss = \frac{1}{n} \sum_{i=1}^{n} (f(\hat{x}^{(i)}) - \hat{y}^{(i)})^2$
- Optimalisering:
  Justering av W og b, for å minske *loss*, gjennom Gradient descent

Oppsett av modell med tapsfunksjoner

```
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
  - 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 W og b, prøv:
  - Minske læringsraten i `torch.optim.SGD`
  - Øk 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 Arch Linux eller Arch Linux baserte distribusjoner som Manjaro 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`
- Hvis python2 er *default* i systemet du bruker, kjør `.py` filene med `python3`

Oppsett av Python IDE/Jupyter Notebook

- Bruk et valgfritt IDE
- Ole's IDE oppsett: juCi++ (ikke anbefalt pga python-language-server/issues/823)
  - Installasjon
  - Oppsett av Python3 støtte
    - På linux må du skive `sudo` foran noen av kommandoene
  - Prosjektene som deles av Ole inneholder 2 ekstra filer:
    - `.python-format` - stil formatering ved lagring av python kode
    - `setup.cfg` - stil formatering oppsett
- Donn skal senere vise dere Jupyter Notebook

Øving 1

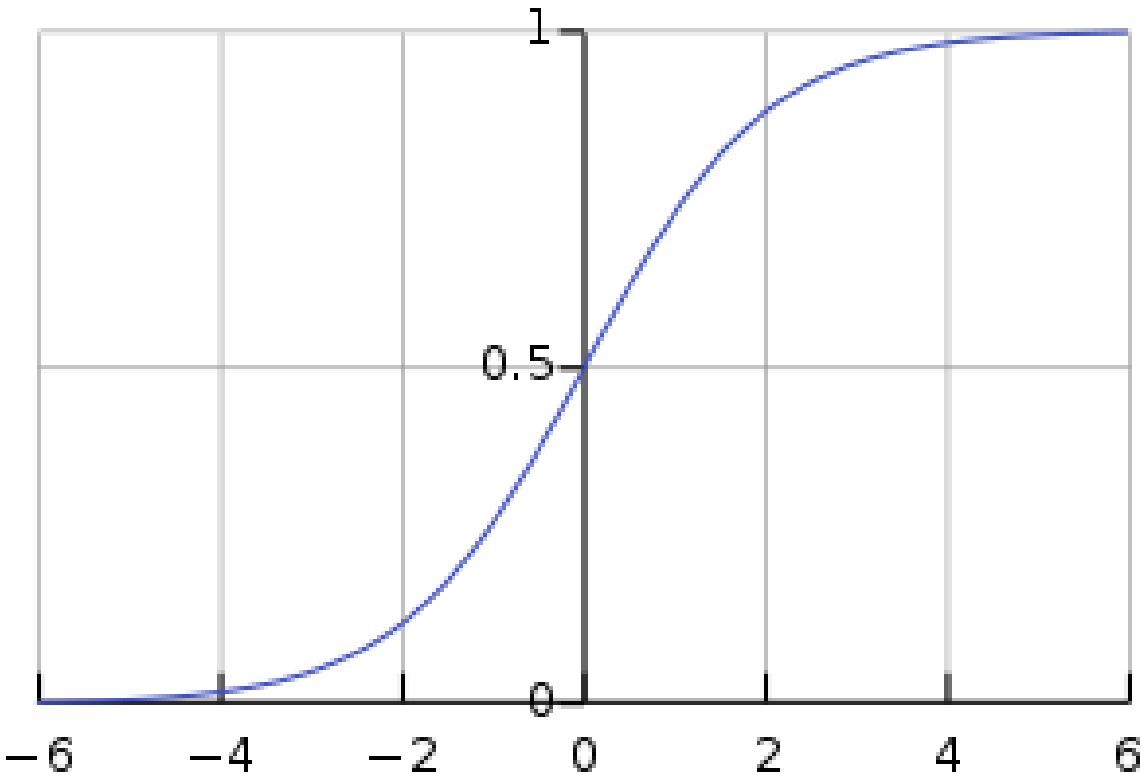
Ta gjerne utgangspunkt i linear-2d.
Datasettene i deloppgavene inneholder observasjoner om nyfødte barn. Det kan være til hjelp å visualisere observasjonene først.
For alle deloppgavene: du skal visualisere modellen etter optimalisering sammen med observasjonene, og skrive ut tapsverdien (*loss*) for 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: $f(x) = 20\sigma(xW + b) + 31$

Øving 1
- sigmoid funksjonen



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