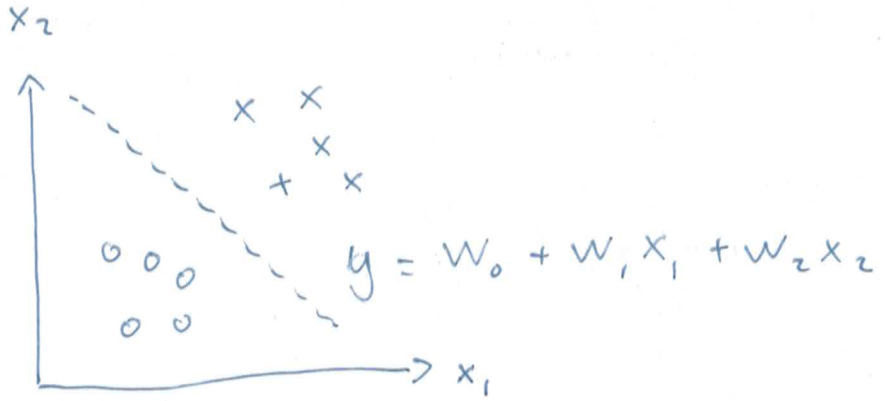
Oppgavesett 2

3) Logistisk regresjon: Finne separasjon mellom to klasser $y = \{0, 1\}$



Parameter vektor:

$\underline{w} = [w_0, w_1, w_2]$

$\underline{x} = [1, x_1, x_2]$

Ligning: $y = \underline{w} \cdot \underline{x}$

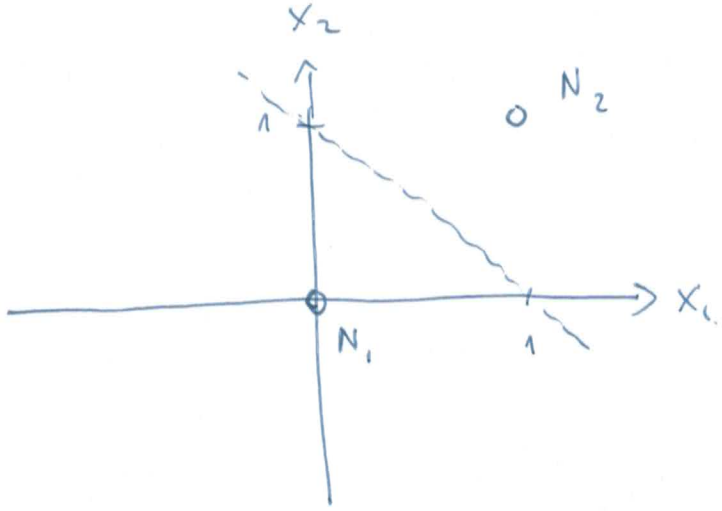
$\underline{w}$ trenes f.eks gjennom gradient descent.

Etter trent $\underline{w}$: Predikerer $\hat{y} = \sigma(\underline{w} \cdot \underline{x})$

$\sigma(y) = \frac{1}{1 + e^{-y}}$: Logistiske funksjon.

$\hat{y}$ angir sannsynlighet for enten $y=0$ eller $y=1$ (grensen går på 0,5).

a)



Mest naturlig å tro at linja $y = 1 - x$ er den beste skille linja...

Problem 4, lab 2

N samples, d features:

$$\underline{\underline{X}} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_d^1 \\ \vdots & & & \\ x_1^N & x_2^N & \cdots & x_d^N \end{bmatrix}$$

$$\underline{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

$\underline{y} = \underline{\underline{X}} \cdot \underline{w}$ kan da skrives som `y = np.dot(X, w)` eller `y = X @ w` i python.

a) Logistisk regresjon, se Ethem Alpaydin, kap 10.7, fig 10.6.

Generelt for gradient descent algoritme:

repeat {

$$w_j := w_j - \alpha \frac{\partial J(\underline{w})}{\partial w_j}$$

}

α: Læringsrate

J: Kostnadsfunk.

↳ Gjøres for alle w_j til konvergens.

$$\frac{\partial J(w)}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} \left(\sigma(\underline{x}^i) - y^i \right) x_j^i$$

$$\sigma(\underline{x}^i) = \frac{1}{1 + e^{-\underline{w} \cdot \underline{x}}}$$

Bias, Representeres som 1 i første kolonne av X

Husk: $z = w_0 1 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d$

Notater i forhold til implementasjon:

* X.shape bør være (N, d)
  - Legg til bias ved `np.concatenate((b, X), axis=1)`
    der `b = np.ones()`
* y.shape = $(N, 1)$
* W.shape = $(d+1, 1)$
* For hvert sample: $z = X(:, i) @ w$ $N \times 1$

$h = \text{sigmoid}(z)$ $N \times 1$

$\nabla = \frac{1}{N} \sum_i (h^i - y^i) \cdot x_j^i$

$\frac{1}{N} X^T @ (h - y)$

$d \times N \cdot N \times 1 = d \times 1$

* Trenger å sette en læringsrate η (kan eksperimentere)

* Trenger å sette #iterasjoner.

* Kan lage en kost-funksjon også for å se at den avtar pr. iterasjon

$$J = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(\sigma(x_i)) + (1-y_i) \log(1-\sigma(x_i))$$

```
def cost(X, y, w):
    N = y.shape[0]
    J = 0
    for i in range(N)
        h = sigmoid(X[i,:] @ w)
        J = J - 1/N (y @ log(h) + (1-y) log(1-h
```

* NB: Bruk np.log og np.exp, ikke math.log, math.exp.