Supervised learning → data set and already know what the correct output should look like; there is a relationship between the input and the output

- **Classification problem** → predict in a discrete output
  "1 or 0"

- **Regression problem** → predict within a continuous value
  "1000's of values"

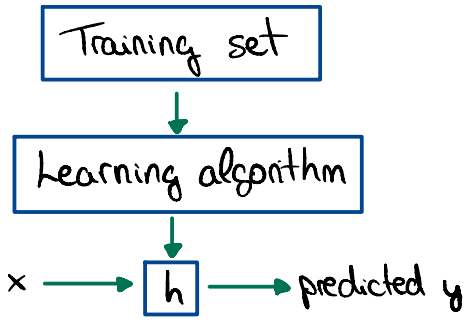
Unsupervised learning → problems with little or not idea what the result should look like; can derive structure from data where we don't necessarily know the effect of the variables.

- **Clustering** → derive the data based on relationships among the variables in the data.

- **Cocktail party** → find structure in a chaotic environment

Model representation (2D idea)

m → number of training examples
h → hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$
θ_0, θ_1 } parameters (to determine)

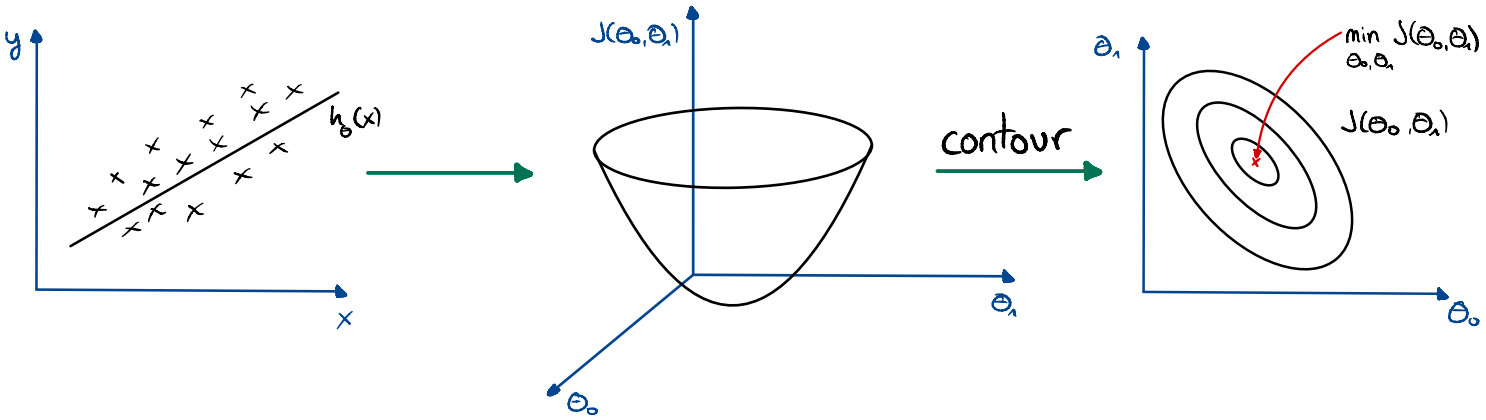


Cost function

Is used to measure the accuracy of the hypothesis function.
Takes an average difference of all results of the hypothesis with inputs from x's and the actual output y's

$\min_{\theta_0,\theta_1} J(\theta_0,\theta_1)$, where $J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$

Squared error function
Mean squared function



Parameter learning

Gradient descent → take the derivative (the tangential line to a function) of the cost function

Gd algorithm:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0,\theta_1)$$ ← repeat until convergence

α → "learning rate" → size of each step
$\alpha \downarrow\downarrow$ → small step
$\alpha \uparrow\uparrow$ → large step

At each iteration j, one should simultaneously update the parameters θ_1, θ_2. Updating a specific parameter prior to calculating another one on the $j^{(th)}$ iteration would yield to a wrong implementation

$$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0,\theta_1)$$
$$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0,\theta_1)$$
$$\theta_0 := temp0$$
$$\theta_1 := temp1$$

At the specific case of linear regression:

$$temp0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$
$$temp1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_i$$
$$\theta_0 := temp0$$
$$\theta_1 := temp1$$

Multivariate linear regression

$x_j^{(i)}$ → value of feature j in the i^{th} training example
$x^{(i)}$ → input (features) of the i^{th} training example
m → the number of training examples
n → the number of features

Multivariate form of the hypothesis function:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

Using matrix multiplication:

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \ldots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

≠ Note that $x_0^{(i)} = 1$ for $(i \in 1, \ldots, m)$

Gradient descend algorithm:

repeat until convergence

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \text{for } j := 0 \ldots n$$

Feature scaling → divide the input values by the range of the input variable → $-1 \leq x_{(i)} \leq 1$

↳ max value − min value

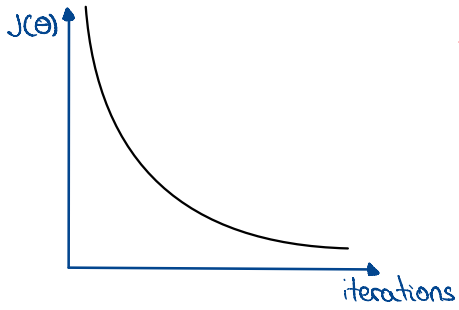
$$x_i := \frac{x_i}{s_i}$$

Mean normalization → substract the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero → $-0.5 \leq x_{(i)} \leq 0.5$

$$x_i := \frac{x_i - \mu_i}{s_i}$$

μ_i → average of all the values for feature (i)
s_i → range of values (max − min) (or standard deviation)

Debugging gradient descend



Declare convergence if $J(\theta) < \varepsilon$, where $\varepsilon = 10^{-3}$

$\alpha \downarrow\downarrow \rightarrow$ slow convergence
$\alpha \uparrow\uparrow \rightarrow$ may not converge

Linear regression improvement → combine multiple features into one

$$x_3 = x_1 x_2$$

Polynomial regression → change the behavior of the curve

$$h_\theta(x) = \theta_0 + \theta_1 x_1 \begin{cases} h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 \\ h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1} \end{cases}$$

Normal equation → minimize J by explicitly taking its derivatives with respect to the θ_j's, and setting them to zero. This allows to find the optimum tetha without iteration

$$\theta = (X^T X)^{-1} X^T y$$

There is not need to do feature scaling

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$, need to calculate $(X^T X)^{-1}$
Works well when n is large	Slow if n is very large
$n > 10^5$	$n < 10^5$

Use `pinv` rather than `inv` → this gives a value of θ even if $X^T X$ not invertible

Common $X^T X$ not invertible causes
- Redundant features → remove the dependency
- Too many features ($m \leq n$) → eq < features

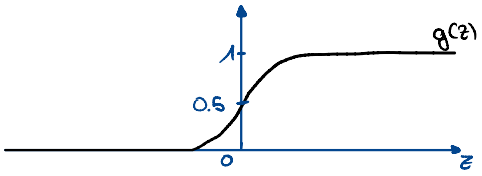
Classification problem

Linear regression doesn't work well, with the hypothesis $0 \leq h_\theta(x) \leq 1 \rightarrow \theta^T x$ can be expresed with the Sigmoid function or Logistic function:

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

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



$\Rightarrow \theta^T x \geq 0 \rightarrow y = 1$
$\theta^T x < 0 \rightarrow y = 0$

in this way $h_\theta(x)$ gives the probability that the output is 1.

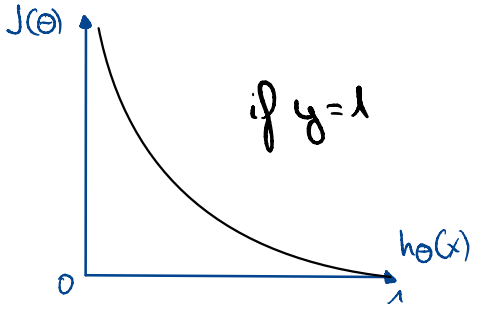
$$h_\theta(x) = P(y=1|x;\theta) = 1 - P(y=0|x;\theta)$$

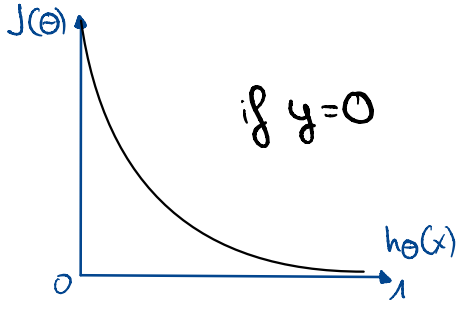
Decision boundary → the line that separates the area where $y=0$ and where $y=1$ created by the hypothesis function.

Cost function → it is not possible to use the same cost function as linear regression because the Logistic Function will cause the output to be wavy, causing many local optima. Instead, the cost function for logistic regression looks like:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}), \text{ where } Cost(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}(h_\theta(x_i) - y_i)^2$$

$$Cost(h_\theta(x), y) = -\log(h_\theta(x)) \quad \text{if } y=1$$
$$Cost(h_\theta(x), y) = -\log(1 - h_\theta(x)) \quad \text{if } y=0$$





fminunc

Properties

$Cost(h_\theta(x), y) = 0$ if $h_\theta(x) = y$ $<$ $y=0$ & $h_\theta(x)=0$ / $y=1$ & $h_\theta(x)=1$ $>$ $Cost = 0$

$Cost(h_\theta(x), y) \to \infty$ if $y=0$ and $h_\theta(x) \to 1$ $\quad y=0, h_\theta(x)=1 \to cost = \infty$

$Cost(h_\theta(x), y) \to \infty$ if $y=1$ and $h_\theta(x) \to 0$ $\quad y=1, h_\theta(x)=0 \to cost = \infty$

Simplified version of the cost function

$$Cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y) \log(1 - h_\theta(x))$$

then:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1-y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

where, a vectorized implementation is:

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} \left[-y^T \log(h) - (1-y)^T \log(1-h) \right]$$

Gradient descent

repeat until convergence

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \text{for } j := 0 \dots n$$

\# in this case $h_\theta = \frac{1}{1+e^{\theta^T x}}$

where, a vectorized implementation is:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - y)$$

Advanced optimization:

Conjugate gradient, BFGS, L-BFGS more sophisticated and faster ways to optimize θ. To compute this, there are two steps:

1) Provide a function that evaluates $J(\theta)$ and $\frac{\partial}{\partial \theta_j} J(\theta)$

```
function [jVal, gradient] = costFunction(theta)
  jVal = [... code to compute J(theta) ...]
  gradient = [... code to compute d/dtheta_j J(theta) ...]
end
```

2) Use function fminc()

```
options = optimset('GradObj', 'on', 'MaxIter', 100)
initialTheta = zeros(2,1)
   [optTheta, functionVal, exitFlag] =
         = fminunc(@costFunction, initialTheta, options)
```

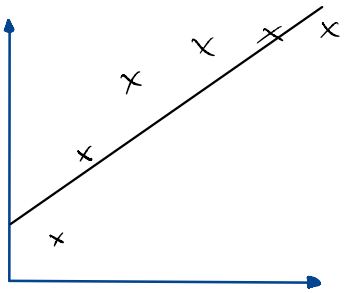
Multiclass classification: One-vs-all

Since $y = \{0, 1, \dots n\}$, we divide our problem into $n+1$ binary classification problems. In each one, we predict the probability that y is a member of one of our classes.

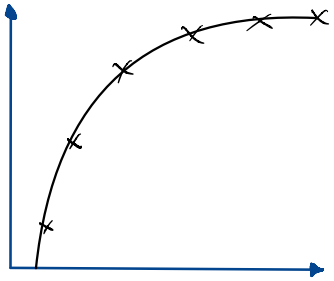
$$y \in \{0, 1, \dots n\}$$
$$h_\theta^{(0)}(x) = P(y=0 \mid x;\theta)$$
$$h_\theta^{(1)}(x) = P(y=1 \mid x;\theta)$$
$$\vdots$$
$$h_\theta^{(n)}(x) = P(y=n \mid x;\theta)$$

} probability that one of them ($y=0$; $y=1$; ...; $y=n$) compared to the rest is $h_\theta^{(n)}(x) = 1$

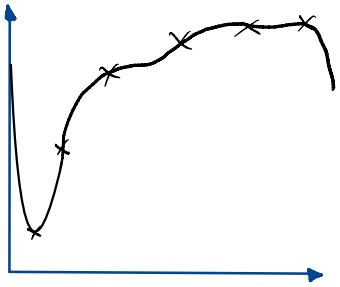
Overfitting → even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor



Underfitting
"high bias"



It's right



Overfitting
"high variance"

Therminology for linear and logistic regression.

Main options to adress the overfitting:

1) Reduce the number of features
   * Manually select which features to keep
   * Use a model selection algorithm
2) Regularization
   * Keep all the features, but reduce the magnitude of θ_j
   * Works well when there are a lot of slightly useful features.