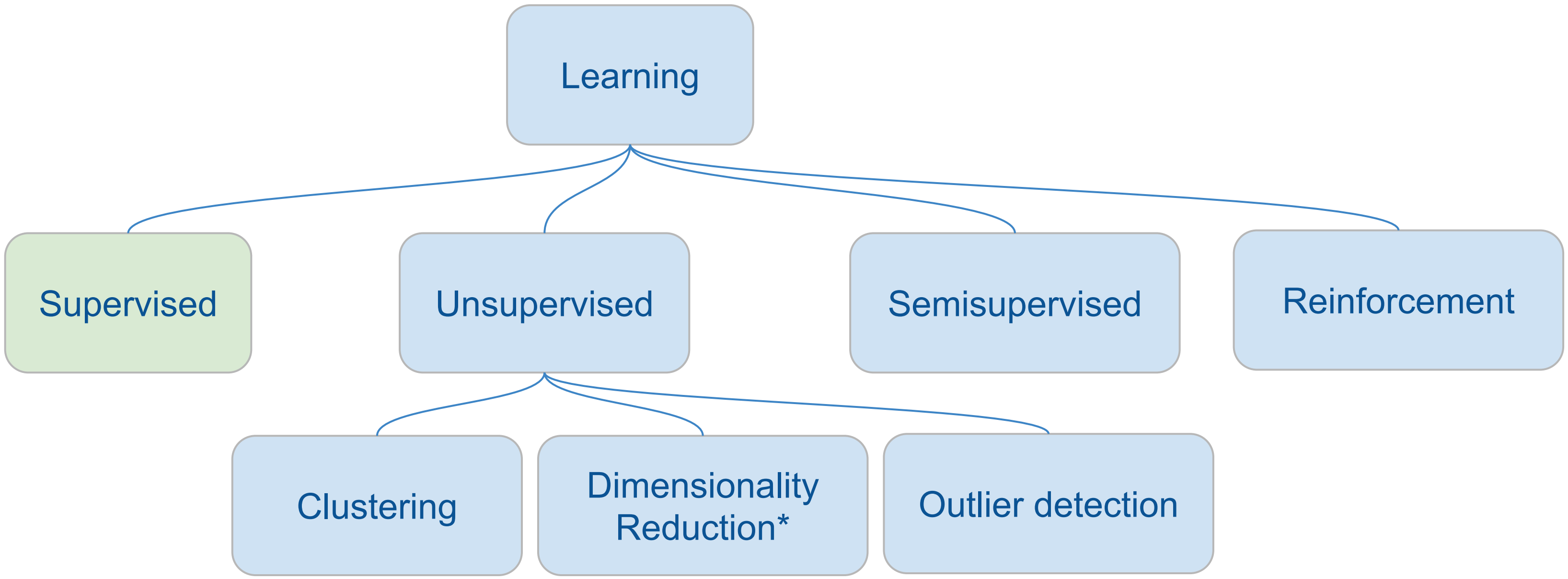
MSc in AI
NCSR Demokritos - University of Piraeus

Course: **Machine Learning**

Lesson 2
Linear Regression

Theodoros Giannakopoulos

Types of learning



Supervised Learning

- Given data and correct output, try to find a relationship between data and output

$$\{(x_i, y_i)\}_{i=1}^{N}$$

- N labelled examples or instances or feature vectors
- xi feature vector $\in \mathbb{R}^D$
- D: dimensionality
- xi(j) j-th feature, j=1, ..., D
- yi label $\in$ {w1, w2, ..., wc} set of **classes** OR $\mathbb{R}$ (regression)

(Supervised) learning of the target function from training data

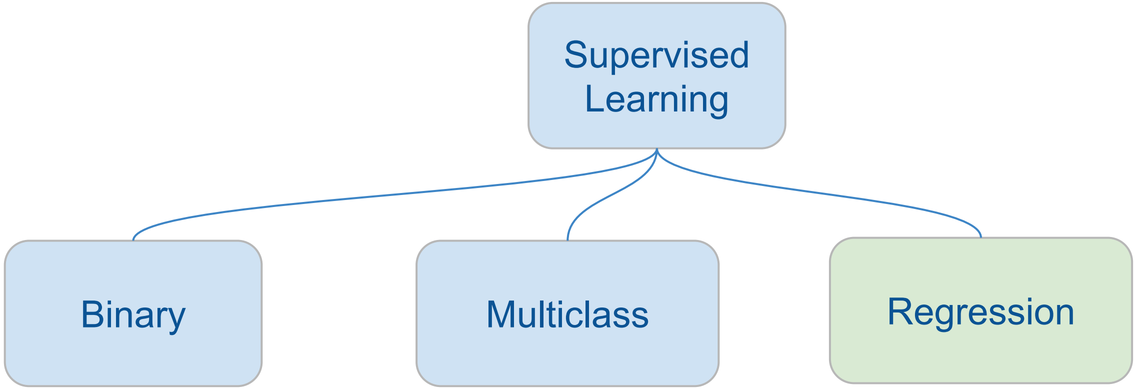
Instance Space → Target Function y=f(x) → **Label Space**

The learning algorithm does not know f(x), it only "sees" some examples of it, though the training data

$$\{(x_i, y_i)\}_{i=1}^{N}$$ → **Learning Algorithm** → g(x)

Training dataset

Types of supervised learning (1/3)

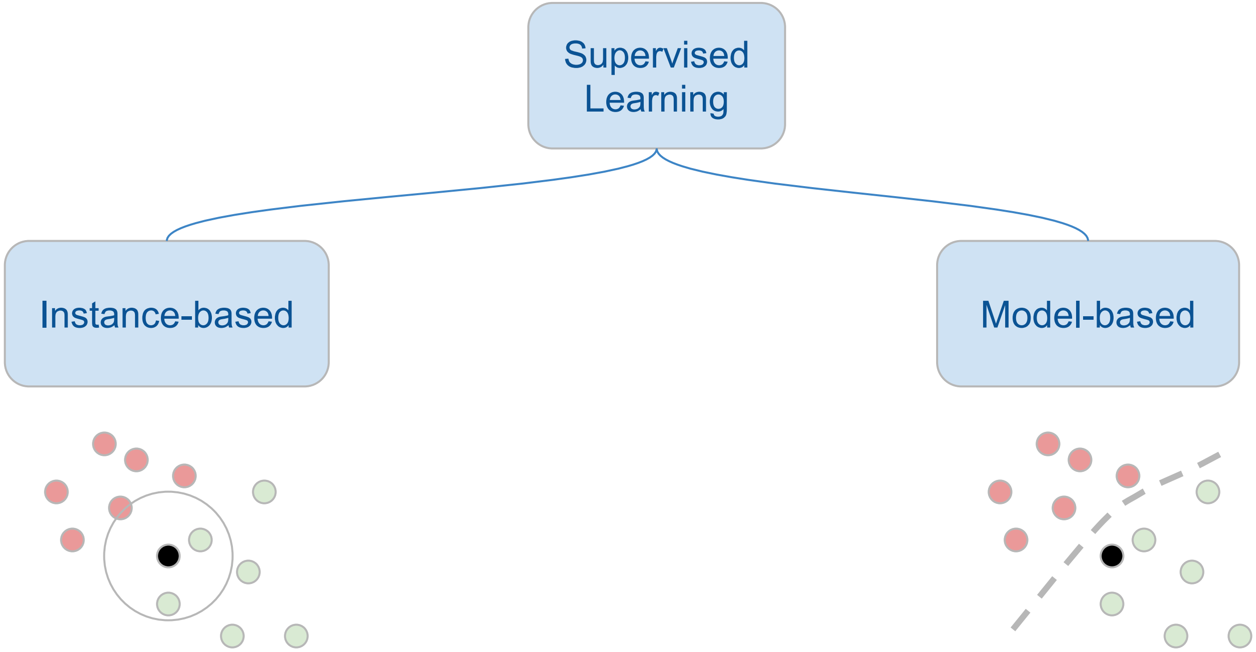


sound is either music or speech?

song is jazz, pop, electro or rock?

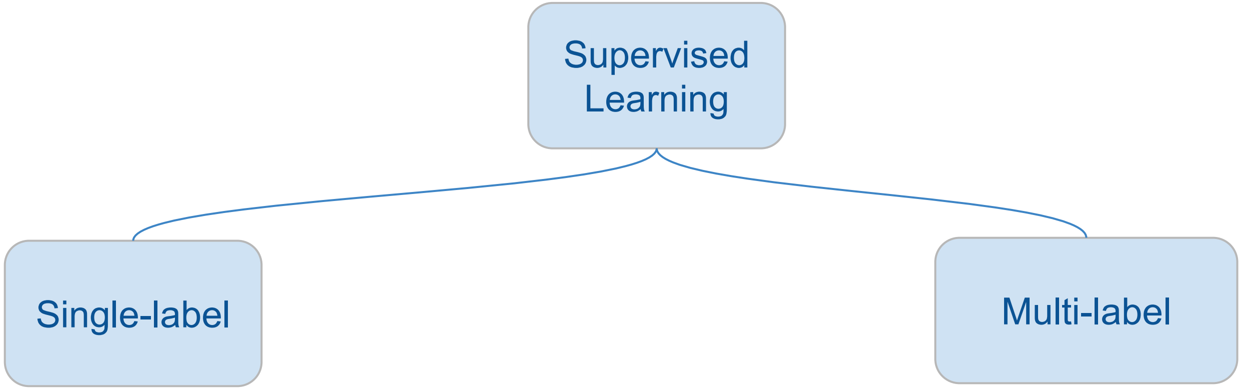
predict the value of this product next year

Types of supervised learning (2/3)



class 1
class 2
unknown instance

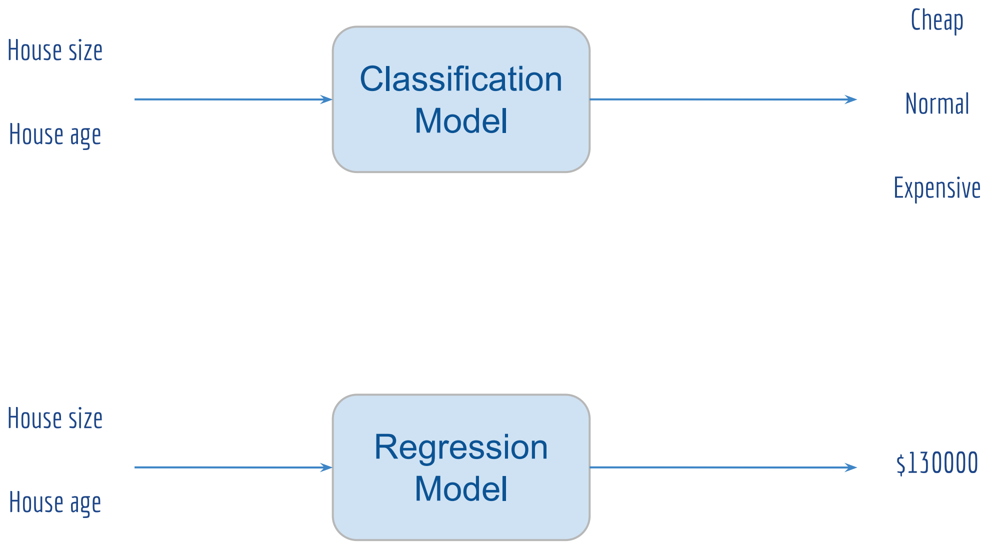
Types of supervised learning (3/3)



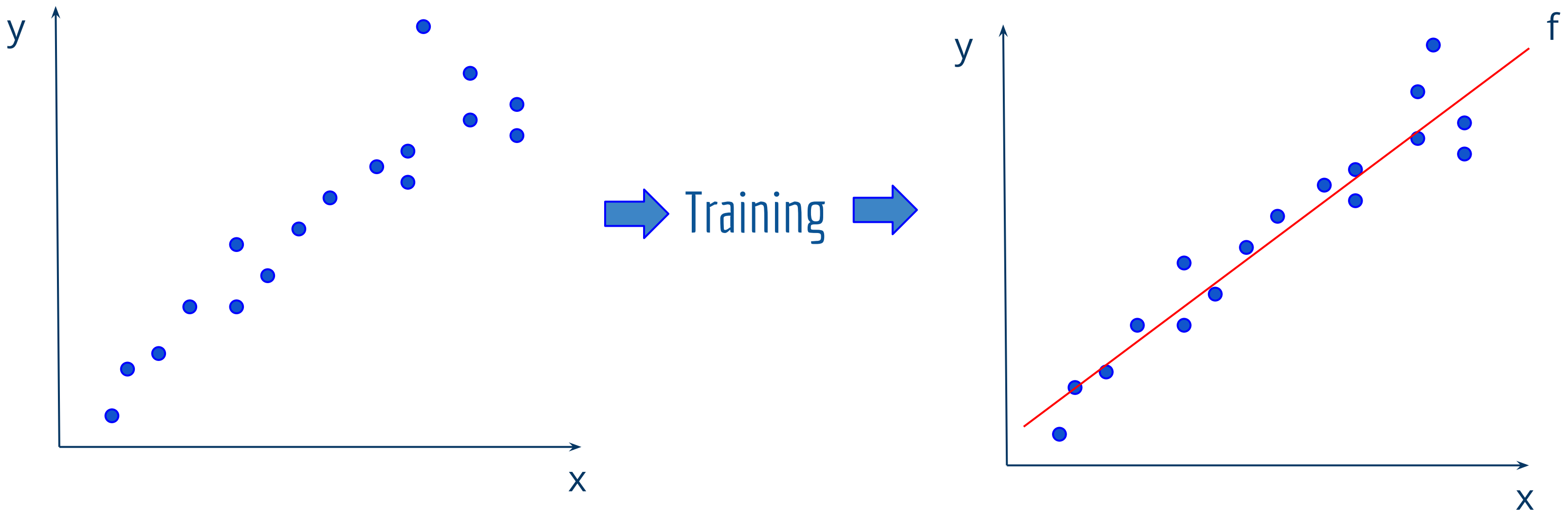
song1: jazz
song2: jazz
song3: electronic
song4: ...

song1: electro-swing, hip-hop
song2: acid-jazz, funk
song3: detroit-techno
song4: ...

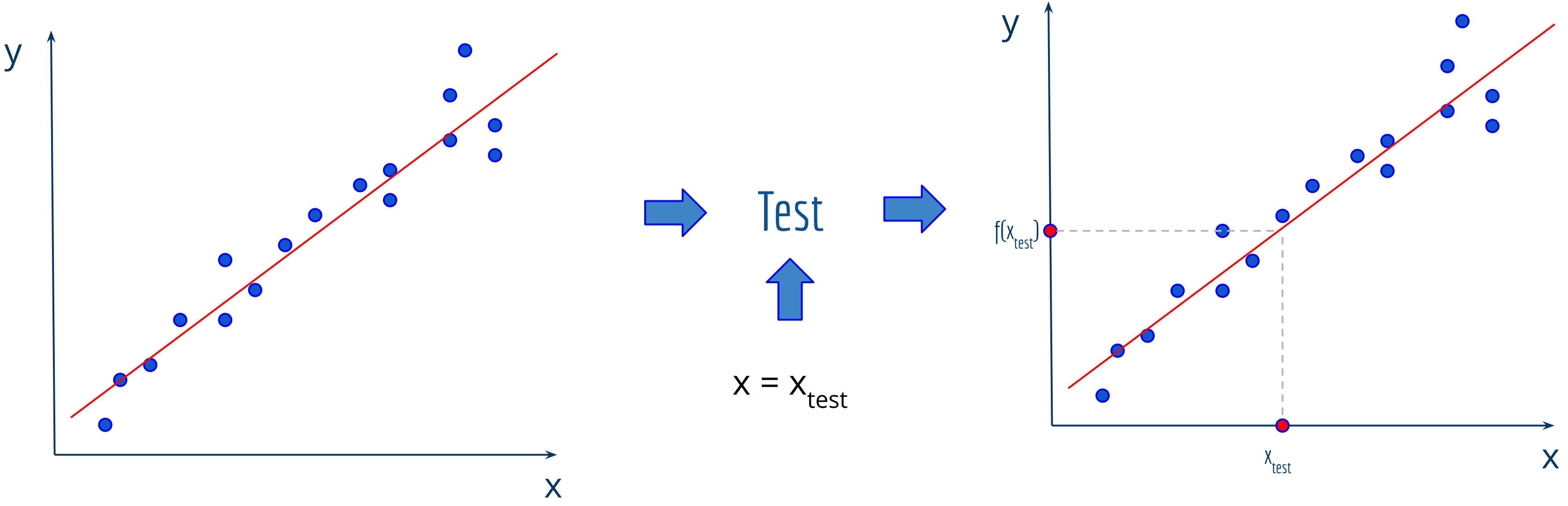
Classification Vs Regression



Linear Regression: train (1D)



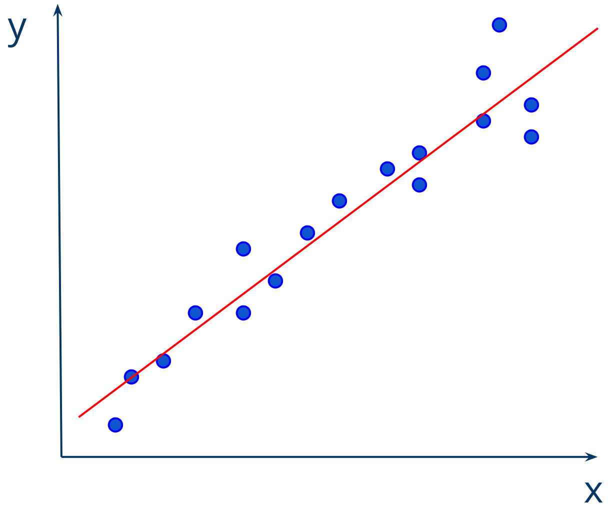
Linear Regression: test (1D)



Linear Regression: problem formulation

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + b$$

- **x**: D-dimensional feature vector
- **w**: D-dimensional vector
- b: bias
- **y**: target value (ground truth)



Regression: $y \in \mathbb{R}$

Linear Regression: problem formulation

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + b$$

M = 0

- **x**: D-dimensional feature vector
- **w**: D-dimensional vector
- b: bias
- **y**: target value (ground truth)
- Model f is parametrized wrt **w** and b
- Note on linearity:
  - Linear regression can also handle **non-linear** relationships between variables
  - f can also be nonlinear to x and still be linear to the w coefficients: $w_0+w_1x+w_2x^2+\ldots+w_Mx^M$
  - (here, we use b as w_0)
  - Don't be confused by the term "linear"!

Linear Regression: problem formulation

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + b$$

$f_{\mathbf{w},b}$ models relationship between features x and target value y

$$y = f + \varepsilon$$

Goal of a regression learning algorithm: estimate **parameters w** and b

(so that h makes a good prediction), i.e. such as:

$$y_i \simeq f(x_i)$$

Linear Regression: how to estimate w and b?

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + b$$

$$J(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{i=0}^{N} (f_{\boldsymbol{w},b}(x_i) - y_i)^2$$

Cost function: measures the error between true and predicted values

Loss function: a measure of penalty for misclassification of each example i

In linear regression, cost function is the average loss (also called empirical risk)

Linear Regression: how to estimate w and b?

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + b$$

$$J(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{i=0}^{N} (f_{\boldsymbol{w},b}(x_i) - y_i)^2$$

Goal:

minimize J to find w and b

Cost function: measures the error between true and predicted values

Loss function: a measure of penalty for misclassification of each example i

In linear regression, cost function is the average loss (also called empirical risk)

Linear Regression: how is J minimized?

$$J(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{i=0}^{N} (f_{\boldsymbol{w},b}(x_i) - y_i)^2$$

Gradient Descent:

- A generic optimization algorithm, not just for LR
- Start with some w, b
- Keep changing each param w (b, w1, w2, ...)

$$w := w - \alpha \frac{\partial J(w, b))}{\partial w}$$

Linear Regression: how is J minimized? (GD for 2 params)

$$J(w_1, b) = \frac{1}{2N} \sum_{i=0}^{N} (f_{w_1,b}(x_i) - y_i)^2 \qquad f_{w_1,b} = w_1 x + b$$

Gradient Descent:

- Select a random value for w1 and b
- Until convergence (or for a max number of epochs):

$$w_1 := w_1 - \alpha \frac{\partial J(w_1, b))}{\partial w_1}$$

$$b := b - \alpha \frac{\partial J(w_1, b))}{\partial b}$$

Linear Regression: how is J minimized? (GD for 2 params)

$$J(w_1, b) = \frac{1}{2N}\sum_{i=0}^{N}(f_{w_1,b}(x_i) - y_i)^2 \qquad f_{w_1,b} = w_1 x + b$$

Gradient Descent:

- Select a random value for w1 and b
- Until convergence (or for a max number of epochs):

$$w_1 := w_1 - \alpha\frac{\partial J(w_1, b))}{\partial w_1} = w_1 - \alpha\frac{1}{N}\sum_{i=0}^{N}(f_{w_1,b}(x) - y)x$$

$$b := b - \alpha\frac{\partial J(w_1, b))}{\partial b} = b - \alpha\frac{1}{N}\sum_{i=0}^{N}(f_{w_1,b}(x) - y)$$

Linear Regression: how is J minimized? (GD for 2 params)

$$J(w_1, b) = \frac{1}{2N}\sum_{i=0}^{N}(f_{w_1,b}(x_i) - y_i)^2 \qquad f_{w_1,b} = w_1 x + b$$

Gradient Descent:

- Select a random value for w1 and b
- Until convergence (or for a max number of epochs):

$$w_1 := w_1 - \alpha\frac{\partial J(w_1, b))}{\partial w_1} = w_1 - \alpha\frac{1}{N}\sum_{i=0}^{N}(f_{w_1,b}(x) - y)x$$

$$b := b - \alpha\frac{\partial J(w_1, b))}{\partial b} = b - \alpha\frac{1}{N}\sum_{i=0}^{N}(f_{w_1,b}(x) - y)$$

Why? Just forget Σ and:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}(f_{\theta,b}(x) - y)^2 = \\ &= 2\frac{1}{2}(f_{\theta,b}(x) - y)\frac{\partial}{\partial\theta_j}(f_{\theta,b}(x) - y) = \\ &= (f_{\theta,b}(x) - y)\frac{\partial}{\partial\theta_j}(f_{\theta,b}(x) - y) = \\ &= (f_{\theta,b}(x) - y)\frac{\partial}{\partial\theta_j}(\textstyle\sum_i \theta_i x_i - y) = \\ &= (f_{\theta,b}(x) - y)x_j\end{aligned}$$

Linear Regression: Gradient Descent Python Example

https://github.com/tyiannak/ml-python

See example 1-linear-regression.ipynb

Linear Regression: how is J minimized? (GD for 2 params)

$$J(w_1, b) = \frac{1}{2N}\sum_{i=0}^{N}(f_{w_1,b}(x_i) - y_i)^2 \qquad f_{w_1,b} = w_1 x + b$$

Gradient Descent:

- Select a random value for w1 and b
- Until convergence (or for a max number of epochs):

$$w_1 := w_1 - \alpha\frac{\partial J(w_1, b))}{\partial w_1} = w_1 - \alpha\frac{1}{N}\sum_{i=0}^{N}(f_{w_1,b}(x) - y)x$$

$$b := b - \alpha\frac{\partial J(w_1, b))}{\partial b} = b - \alpha\frac{1}{N}\sum_{i=0}^{N}(f_{w_1,b}(x) - y)$$

Batch gradient descent: at each step ALL data points are needed

(can be very slow for big data)

Linear Regression: Stochastic Gradient Descent

Stochastic gradient descent

Iteratively uses the derivative of one single example

Goes close to global minimum faster but with a more noisy "path"

Good for very large datasets

(never quite converges)

Batch gradient descent: at each step ALL data points are needed

(can be very slow for big data)

Linear regression: closed-form solution

- **Gradient descent** is an iterative algo that minimizes the cost, by gradually reducing it
- **Particularly** for **linear regression**, cost minimization can be computed through a closed-form solution
- Compute function's minimum $\longrightarrow$ set partial derivatives to zero (critical point): direct solution
- X: N x D matrix (rows represent training examples)
- y: target values (N-dimensional vector)

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

Linear regression: why iterative if there's a closed-form solution?

- **Gradient descent** is an iterative algo that minimizes the cost, by gradually reducing it
- Reason: computational complexity: X^TX and (especially) inversion can take a long time
- GD is much more easily parallelizable than matrix manipulation

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

Regression Evaluation Metrics

- (More details on train/val/test and other ML evaluation issues in later courses)

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$