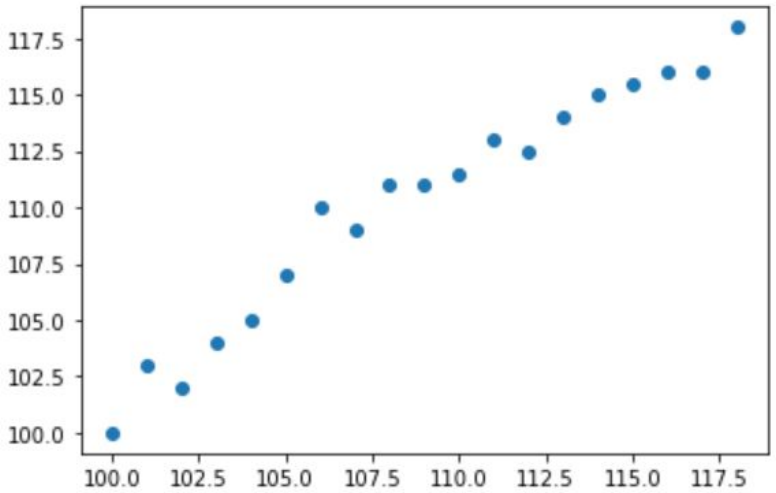
Regression

AMC spring 1400
Presenter : Daniel Azimi

Dealing with continuous values

- Our output or goal is a continuous value
- Examples : house price , currency value ...

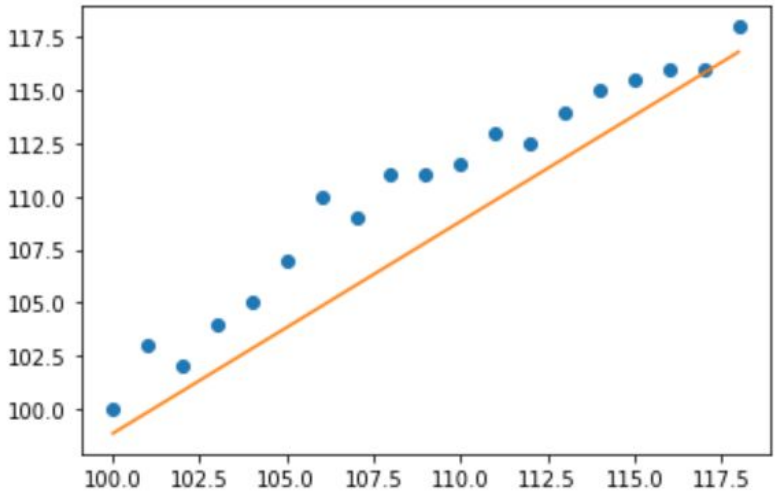


Predicting Linear Data

- We have two main learnable parameters that we train in our algorithm
  - Weight or slope
  - Bias or line intercept
- If you remember from calculus :
  - $y = a * x + b$ or $y = w * x + b$

Fitting a line on Data

- With learning we can fit a line on our data
- Our parameters are what is being learnt
- We need to find a way to update our parameters



Defining Cost

- For us to be able to have measure of how good our model is we need too define a cost function.
- We must update our parameters based on our cost function
- Our goal is to minimize the cost function

Squared Error Loss

- To measure our loss for each sample
- It is defined as such to give us an absolute value that can still have its derivative taken.

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

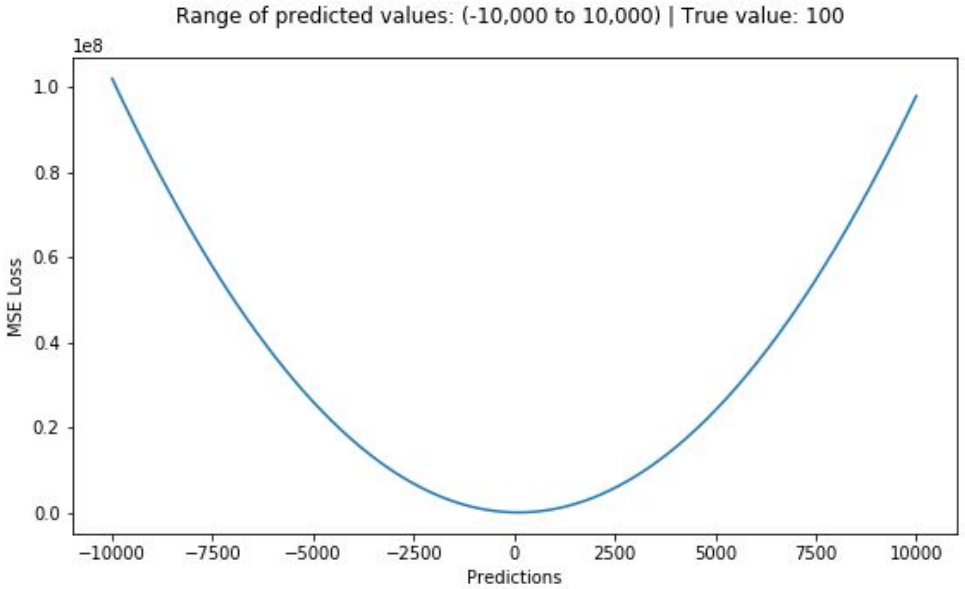
Mean Squared Error

- We measure the difference between our prediction and actual result
- The average loss over our entire training set is defined as our cost
- Our goal is to minimize this function

$$\mathrm{MSE} = \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - t^{(i)} \right)^2$$

Gradient Descent

- A method to iteratively update our parameters by the derivative of our cost
- We try to move to the Global minimum of our cost function
- We may not get to the exact minimum but we try to get as close as possible



Derivatives and the chain rule

- In calculus we learnt of the chain rule a method to more easily compute derivatives of a composite function
- We define the chain rule as the following :

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Derivatives of our parameters

- We define the derivative for our weight(slope) as follows :

$$\frac{\partial \mathcal{E}}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \, x_j^{(i)}$$

- And for b or our intercept line as :

$$\frac{\partial \mathcal{E}}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} y^{(i)} - t^{(i)}$$

Updating our Parameters

- Updating our weights on each iteration :

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{w}}$$

- The same is repeated for b

Python code: Vectorized Cost

- Cost function implementation in python
- Vectorized using numpy library
- t : real value
- y : computed value based on parameters

```
def cost(x,t,w,b):
  N = len(x)
  y = np.multiply(w,x) + b
  err = np.sum((y-t)**2)
  return err/float(N*2)
```

Python code: Updating derivatives

```
alpha = 0.01

#derivative of cost with respect to w
dw = np.sum(y - t)/m

#derivative of cost with respect to b
db = np.sum(y-t)

#updating slope by its derivative
w = w - alpha * dw

#updating intercept by its derivative
b = b - alpha * db
```

Binary Classification using regression

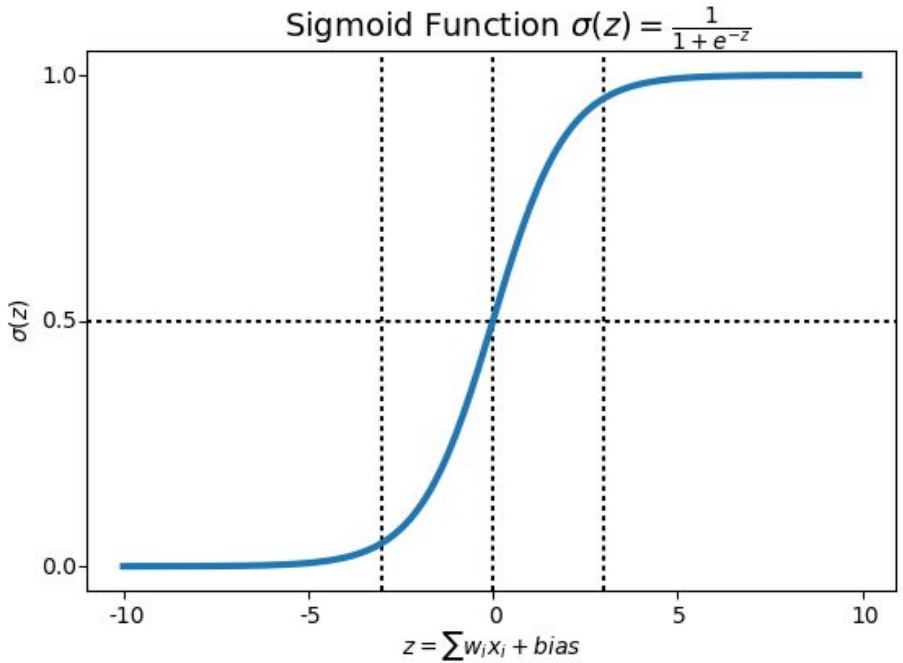
- If our data is in a form that can be separated by using a line or a hyperplane then we will be able to use regression
- To be able to do this we will need to consider a new function to change our prediction to a form suitable for classification

The Sigmoid function : Definition

- This function brings our values between 0 and 1
- If the value of our variable z is very large we are closer to 1 and if the value of z is very small then we are closer to 0

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

The Sigmoid function : Graph



The Sigmoid function : Derivative

- The computed Derivative of the sigmoid function is as follows

$$\frac{d\big(\sigma(\mathrm{x})\big)}{dx} = \sigma(\mathrm{x})\big(1 - \sigma(\mathrm{x})\big)$$

- It is encouraged that you compute the derivative as practice to gain better insight

Cross entropy loss

- As can be seen our values will all range between 1 and 0
- MSE is ill suited to calculate such a cost because of how small the values are
- We will use a different cost function known as cross entropy loss

$$\mathcal{L}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \frac{1}{n} \sum_{i=1}^{n} -y_i \log\left(\hat{y}_i\right) + \left(1 - y_i\right) \log\left(1 - \hat{y}_i\right)$$

Cross entropy loss : Derivative

- The computed derivative for the cross entropy loss is :

$$\frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} = \frac{\hat{y} - y}{(1-\hat{y})\hat{y}}$$

- Can also be vectorized using numpy

Python code：Logistic Regression

```
def sigmoid(z):
    return 1/(1+np.exp(-z))



def update(x,t,w,b):
    z = w * x + b

    a = sigmoid(z)

    #dw = dL/da * da/dz * dz/dw
    dw = ((a - t) / (1-a)*a) * (sigmoid(z)* (1-sigmoid(z))) * x

    #db = dL/da * da/dz * dz/db
    db = ((a - t) / (1-a)*a) * (sigmoid(z)* (1-sigmoid(z))) * 1

    w = w - alpha * dw

    b = b - alpha * db

    return w , b
```

Epilogue :

- Linear Regression is great but has some drawbacks:
  - Doesn't work for a lot of problems cause most problems are not be described linearly.
  - Can overfit data if not handled correctly and must be wary of such
  - Can be computationally intensive for large data sets when using classic GD
- Advantages :
  - It's very simple both conceptually and to implement
  - If our data can be described linearly then it is one of the best tools