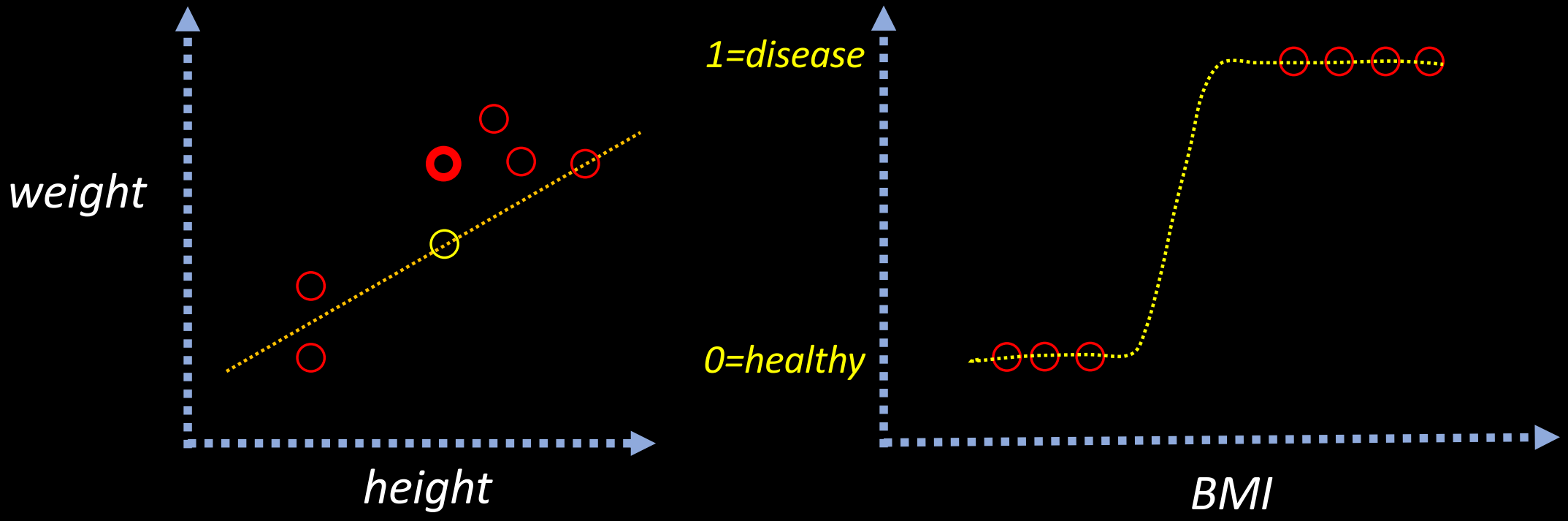
Linear & Logistic Regression

Regression vs Classification

Regression: Predicts continuous outcomes by finding the best-fit line through data points. Used for forecasting and understanding relationships between variables.

Classification: Predicts the probability of an instance belonging to a particular class.

Regression vs Classification



Regression:

Predicting drug dosage based on patient weight
Estimating bone density changes over time in osteoporosis patients
Analyzing the relationship between blood pressure and salt intake

Classification:

Predicting the likelihood of heart disease based on cholesterol levels
Determining the probability of remission in cancer patients given certain treatment
Classifying medical images as showing presence or absence of a tumor

```
# Linear Regression: Predicting drug dosage based on patient weight
from sklearn.linear_model import LinearRegression
import numpy as np

# Sample data: patient weights (kg) and corresponding drug dosages (mg)

weights = np.array([[60], [70], [80], [90], [100]])
dosages = np.array([100, 115, 130, 145, 160])
```

Code Block 1

```
model = LinearRegression()

model.fit(weights, dosages)

# Predict dosage for a new patient weighing 75 kg
new_weight = np.array([[75]])
predicted_dosage = model.predict(new_weight)

print(f"Predicted dosage for 75 kg: {predicted_dosage[0]:.2f} mg")

# Output: ```Predicted dosage for 75 kg: 122.50 mg ```
```

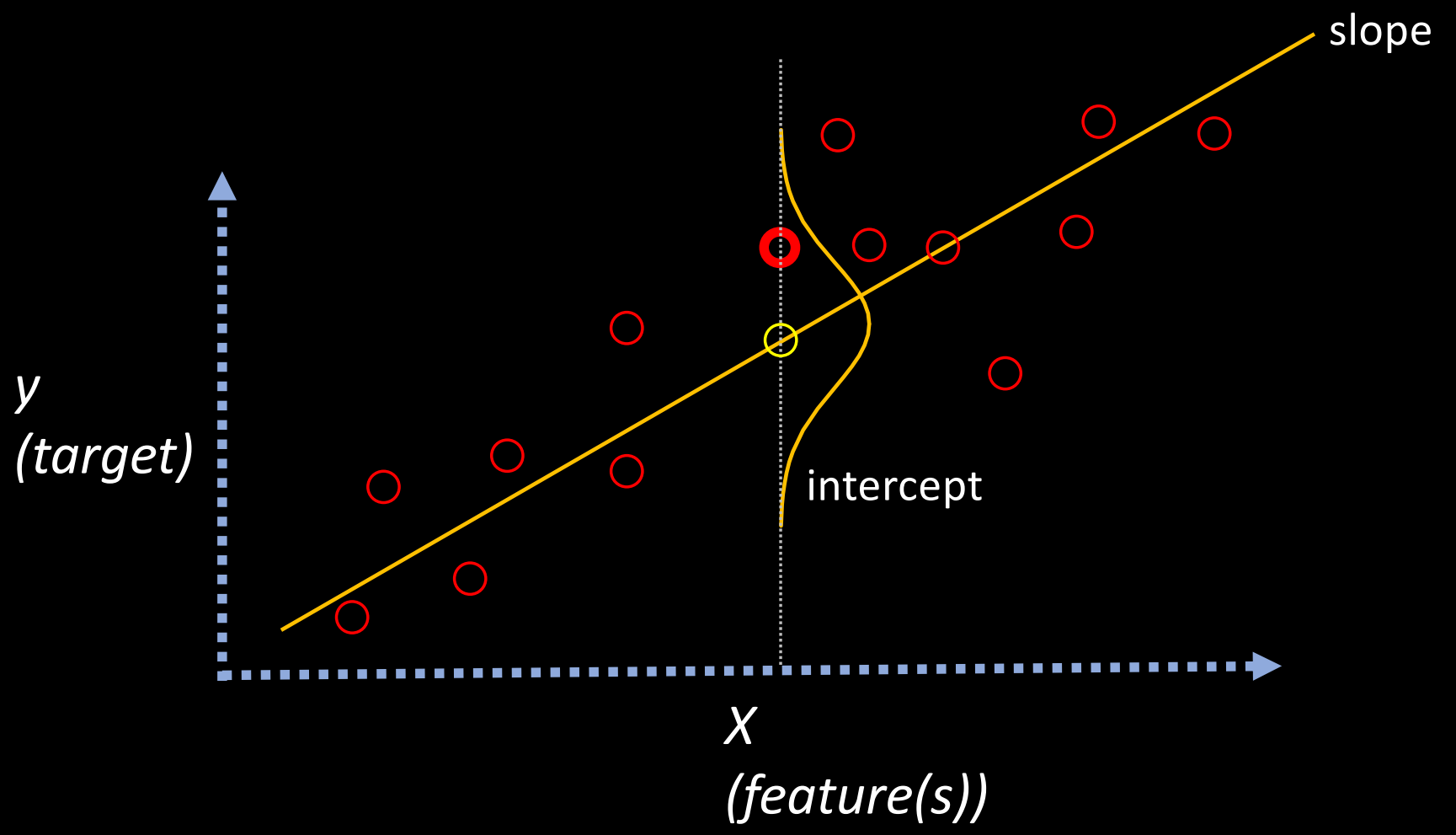
Code Block 1

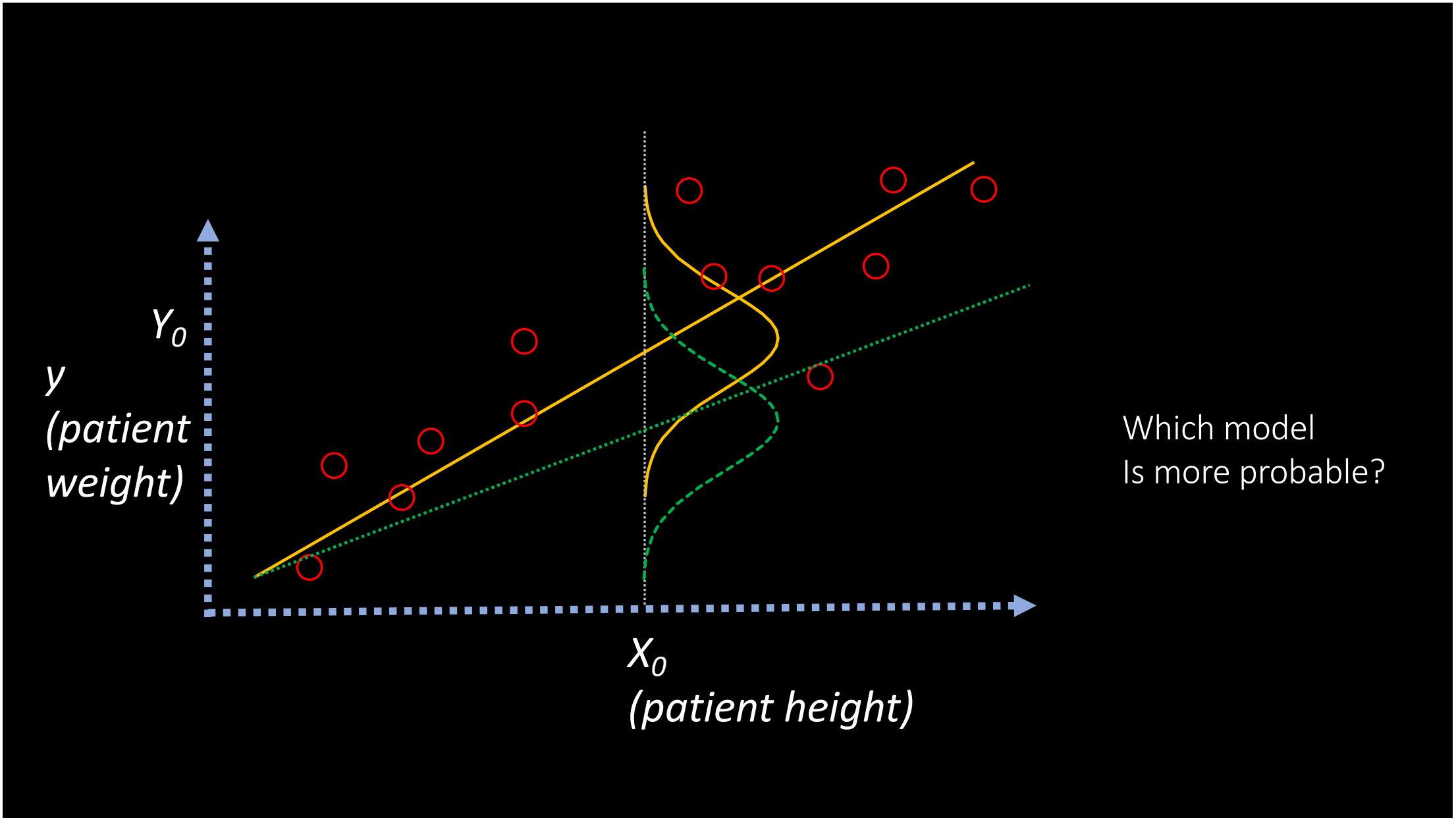
```
# Logistic Regression: Predicting presence of heart disease

from sklearn.linear_model import LogisticRegression
# Sample data: patient cholesterol levels and heart disease
presence (0: No, 1: Yes)

cholesterol = np.array([[150], [200], [250], [300], [350]])
heart_disease = np.array([0, 0, 1, 1, 1])

log_model = LogisticRegression()

log_model.fit(cholesterol, heart_disease)
```

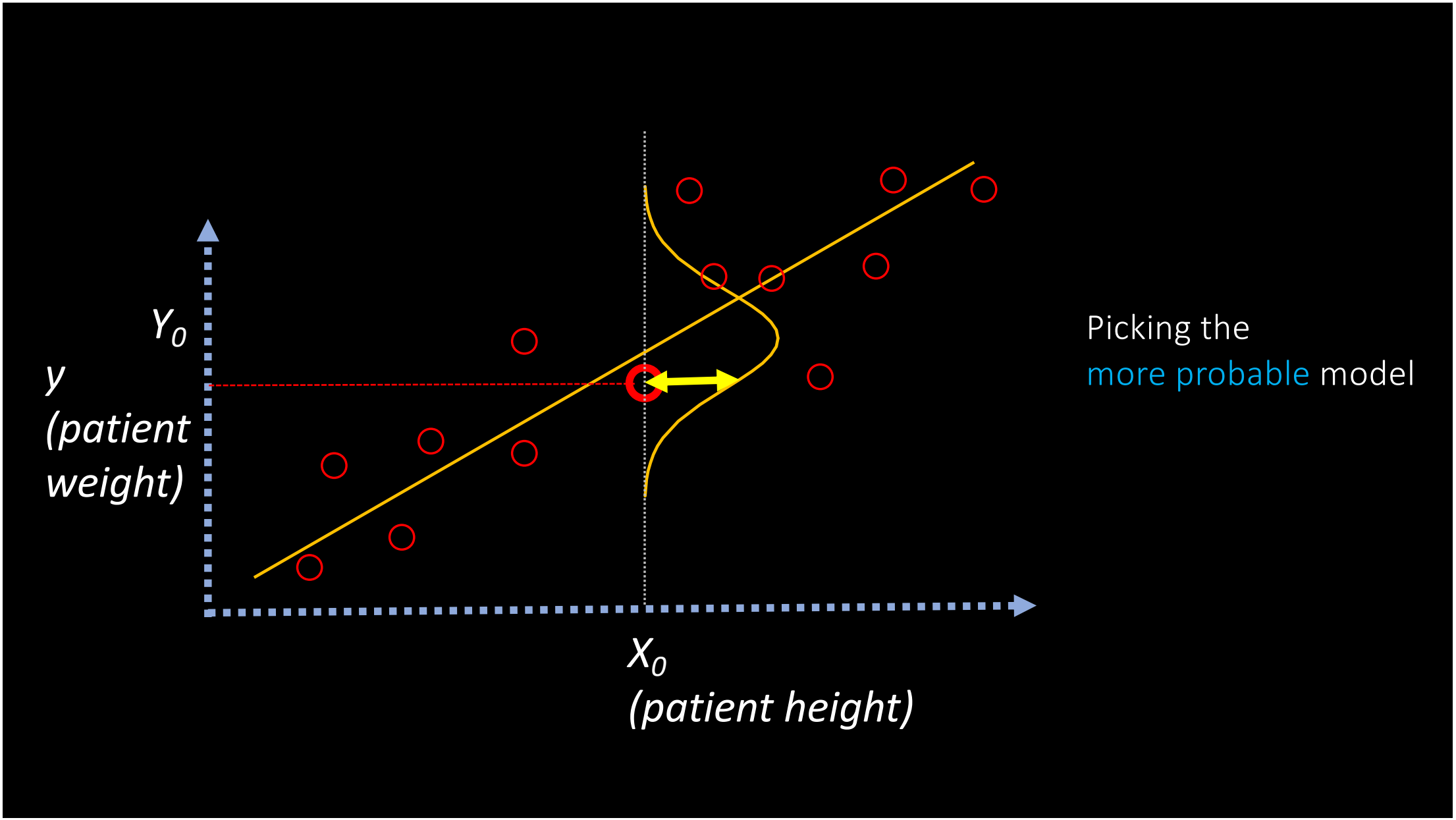
Code Block 1

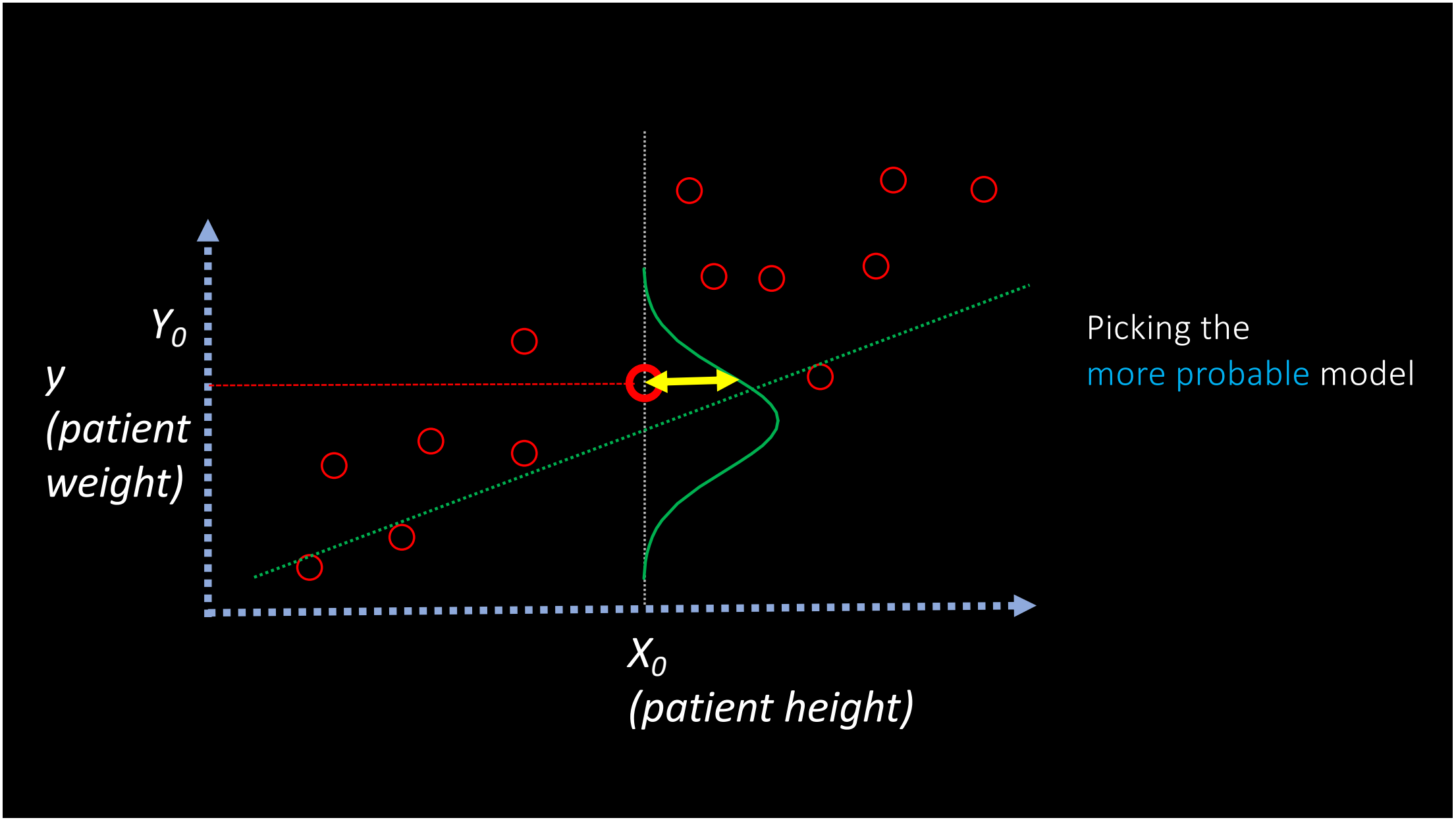
```
# Predict heart disease for a new patient with cholesterol level
275
new_cholesterol = np.array([[275]])
predicted_prob =
log_model.predict_proba(new_cholesterol)[0][1]

print(f"Probability of heart disease for 275 cholesterol:
{predicted_prob:.2f}")

# Output: ```Probability of heart disease for 275 cholesterol: 0.73
```
```

**Code Block 1**
```

Linear Regression: Predicts continuous outcomes




Y_0
y
(patient
weight)
X_0
(patient height)
Which model
Is more probable?


Y_0
y
(patient
weight)
X_0
(patient height)
Picking the
more probable model


Y_0
y
(patient
weight)
X_0
(patient height)
Picking the
more probable model

Maximum Likelihood Estimation (MLE)

$$\theta_{ML} = argmax_{\theta} P(y|X;\theta) = argmin_{\theta}(-P(y|X;\theta))$$

MLE is a method for estimating the parameters of a statistical model by maximizing the likelihood function.

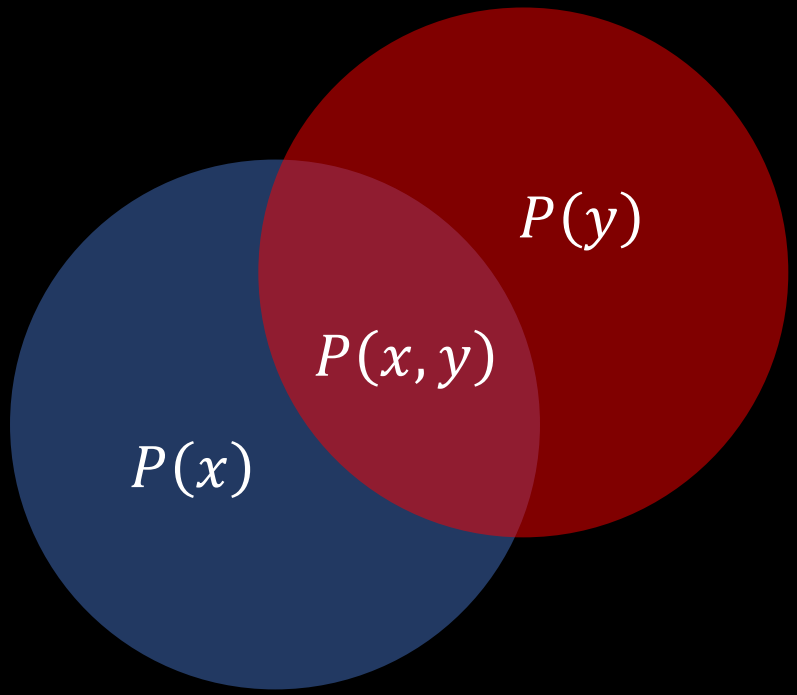
Key Concept: It finds the parameter values that make the observed data most probable.

Review probability

$P(x)$ Probability

$P(x, y)$ Joint Probability

$P(y|x) = P(x, y) \,/\, P(x)$ Conditional Probability



Probability:

The likelihood of an event occurring. Example: The probability of a patient responding to a specific drug treatment.

Joint Probability:

The probability of two or more events occurring together. Example: The probability of a patient having both diabetes and hypertension.

Conditional Probability:

The probability of an event occurring, given that another event has already occurred. Example: The probability of a patient having a heart attack, given that they have high blood pressure.

Maximum Likelihood Estimation (MLE)

$$\theta_{MLE} = argmax_{\theta} P(y|X;\theta))$$

$P(y\|X)$	Probability of seeing y given X (true probabilistic distribution)
$P(y\|X;\theta)$	Probability of seeing y predicted X from model with parameters θ
$argmax_{\theta}$	Find the parameters θ to maximize the given probability $P(y\|X,\theta)$

Probability

$$P(A|B)$$

Chance of seeing data A given fixed condition B

$:=$

Likelihood

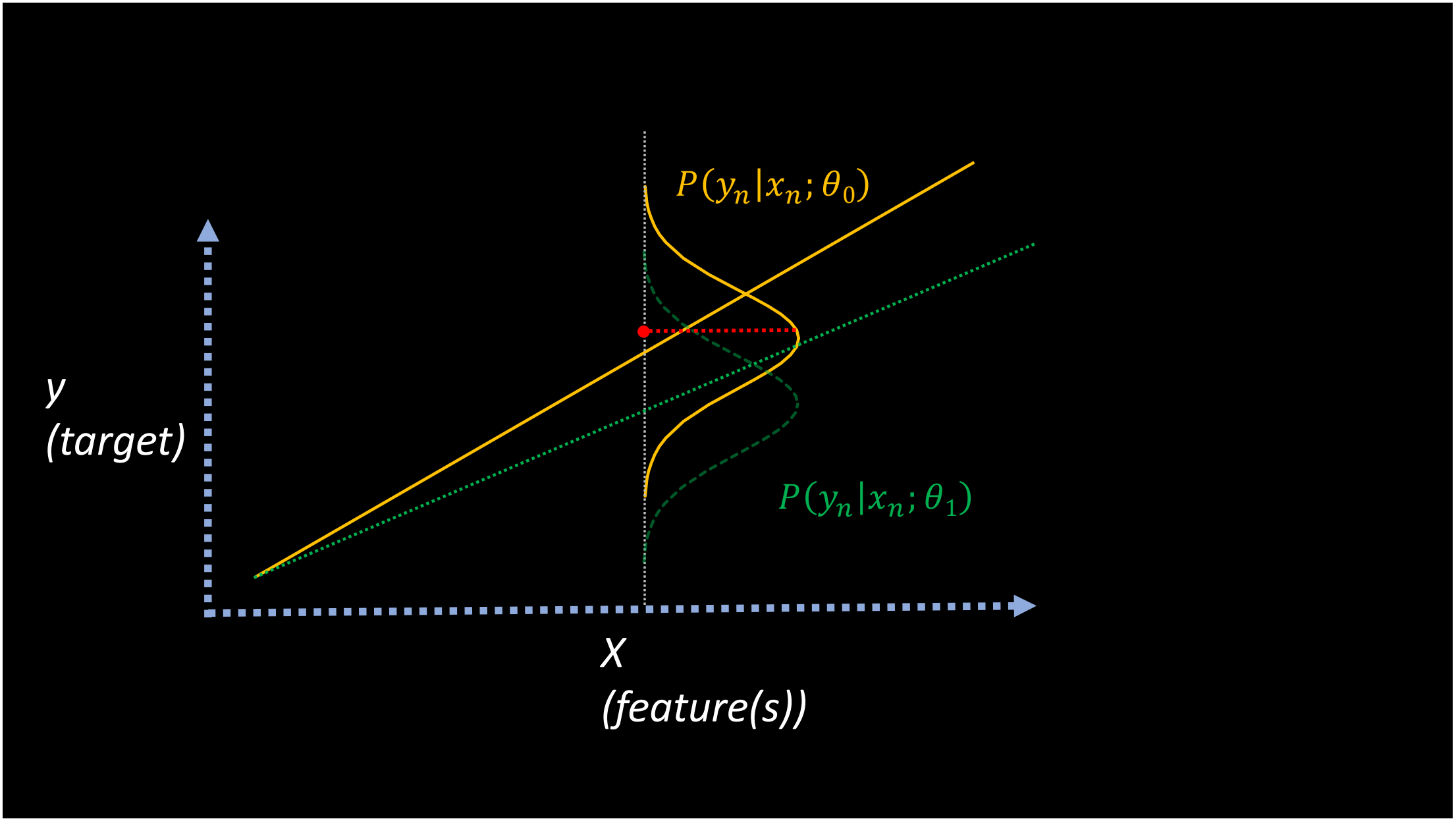
$$L(B|A)$$

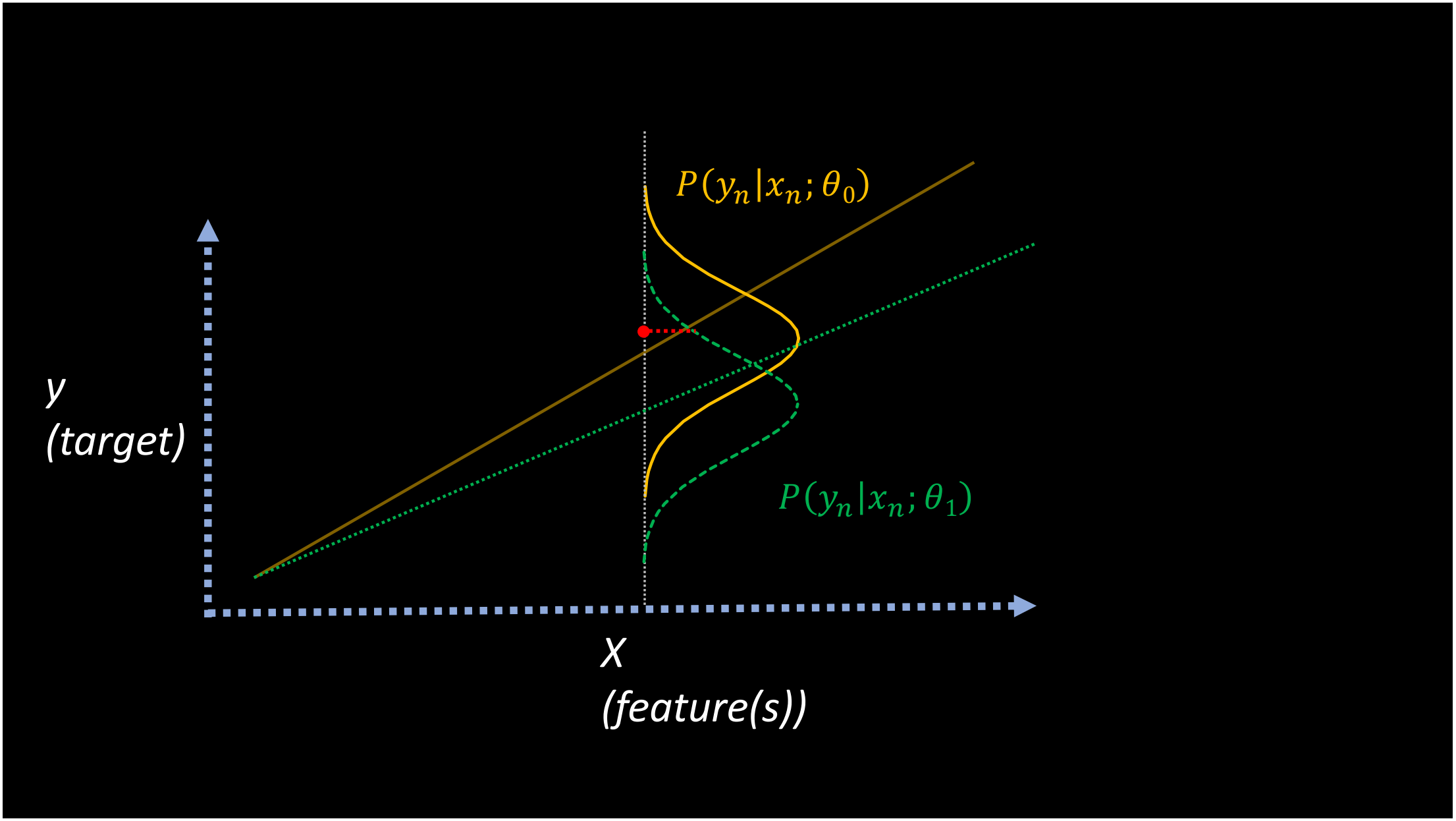
Chance of condition B given seeing data A

Maximum Likelihood Estimation (MLE)

$$\theta_{MLE} = argmax_{\theta} P(y|X;\theta))$$

$P(y\|X)$	(true) probability of seeing stroke given blood pressure
$P(y\|X;\theta)$	Probability of seeing stroke given blood pressure from model with θ
$argmax_{\theta}$	Find the parameters θ to maximize the given probability $P(y\|X,\theta)$


$P(y_n|x_n;\theta_0)$
$P(y_n|x_n;\theta_1)$
y
(target)
X
(feature(s))

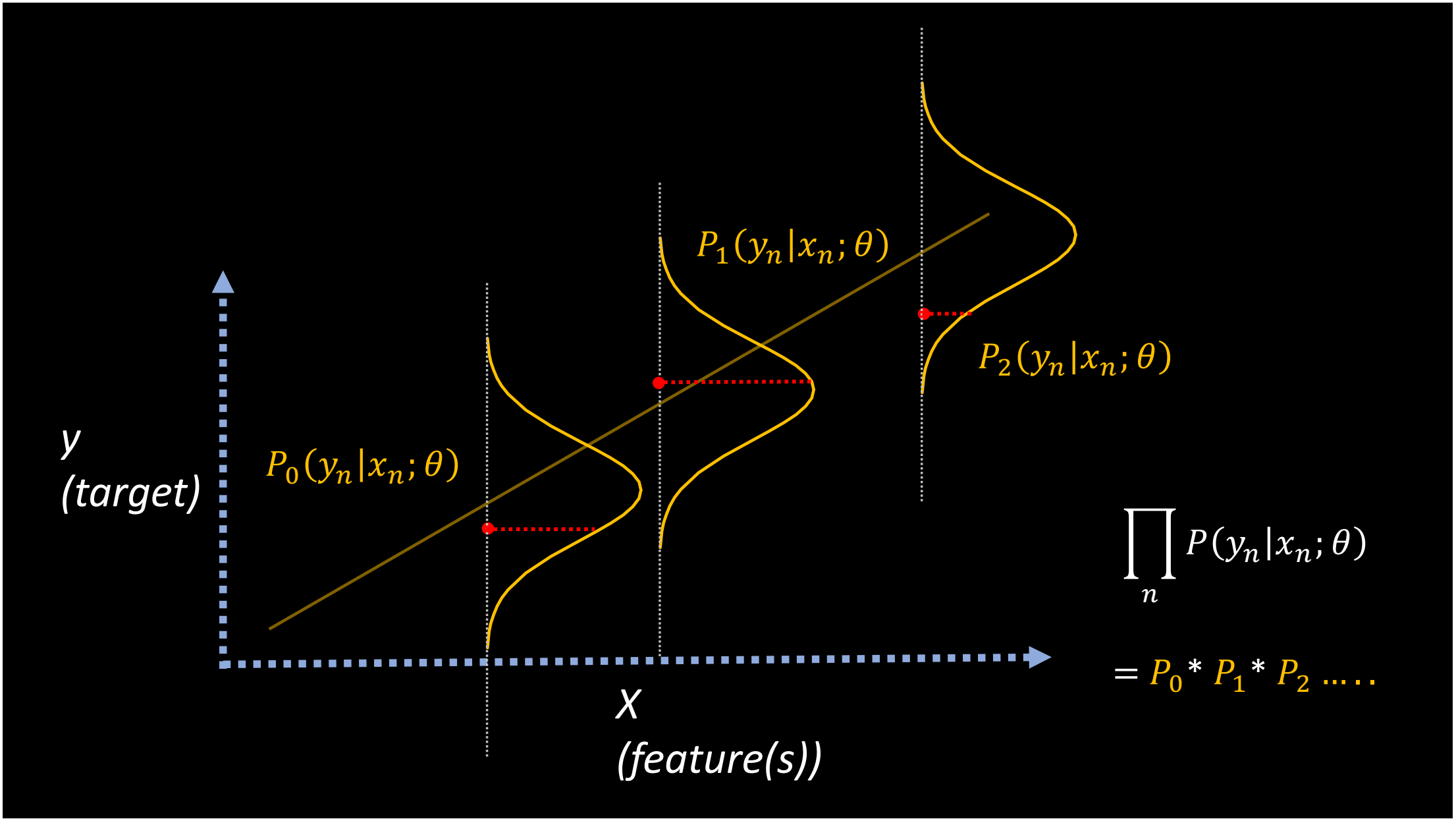

$P(y_n|x_n;\theta_0)$
$P(y_n|x_n;\theta_1)$
y
(target)
X
(feature(s))

MLE in Linear Regression

Assumption: In linear regression, we assume errors are normally distributed around the true regression line.

Likelihood Function: Measures the probability of observing our data given a particular set of regression parameters (slope and intercept).

MLE Process: Finds the regression line that maximizes this likelihood function, making our observed data most probable, equivalent to minimizing the sum of squared errors (least squares method).


$P_1(y_n|x_n;\theta)$
$P_2(y_n|x_n;\theta)$
$P_0(y_n|x_n;\theta)$
y
(target)
X
(feature(s))
$\prod_n P(y_n|x_n;\theta)$
$= P_0 * P_1 * P_2 \ldots\ldots$

$$P(A, B) = P(A) * P(B)$$

If A and B are independent
(often assumed that data are **independent and identically distributed (IID))**

For $y = \{y_1, y_2 \dots y_i\}$, $X = \{x_1, x_2 \dots x_n\}$

Features and targets for all the observations

$P(y|X; \theta)$ = Likelihood of all the observations

$$P(y_1|x_1; \theta) * P(y_2|x_2; \theta) \dots * P(y_n|x_n; \theta) = \prod_n P(y_n|x_n; \theta)$$

```
# Data points
weights = np.array([[60], [70], [80], [90], [100]])
dosages = np.array([100, 115, 130, 145, 160])

# Create and train the good model
model_good = LinearRegression()
model_good.fit(weights, dosages)

# Create a bad model with random coefficients
model_bad = LinearRegression()
model_bad.coef_ = np.array([1.53])
model_bad.intercept_ = np.array([9.5])
```

Code Block 2

```
def calculate_probabilities(predictions, actual, std_dev):
        return norm.pdf(actual - predictions, loc=0, scale=std_dev)

# Predict dosages
predictions_good = model_good.predict(weights)
predictions_bad = model_bad.predict(weights)


probs_good = calculate_probabilities(predictions_good, dosages, 1)
probs_bad = calculate_probabilities(predictions_bad, dosages, 1)

print(probs_good) print(probs_bad)
```

Code Block 2

prob of the bad model:

[0.17136859 0.11092083 0.06561581 0.03547459 0.0175283]

prob of the good model:

[0.39894228 0.39894228 0.39894228 0.39894228 0.39894228]

Code Block 2

Joint Probability

$$P(y_1|x_1;\theta) * P(y_2|x_2;\theta) \ \ldots * P(y_n|x_n;\theta) = \prod_n P(y_n|x_n;\theta)$$

$$\prod_n P(y_n|x_n;\theta) \qquad (0.1 * 0.1 * 0.1 * 0.1 \ldots) \qquad 0.000000001$$

VS

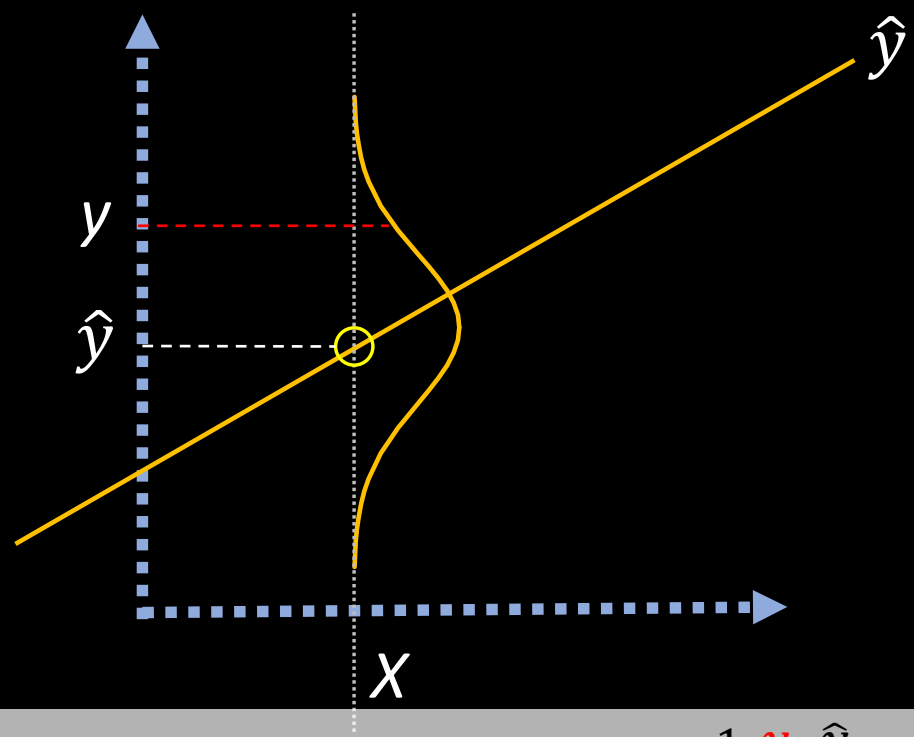
$$\sum_i logP(y_n|x_n;\theta) \quad \text{(-1 -1 -1 -1 -1.....)} \qquad -9$$

Log Probability

```python
log_probs_good = np.log(probs_good)
log_probs_bad = np.log(probs_bad)
print("\nLog probabilities for good model:")
print(log_probs_good)print(log_probs_bad)


# Calculate sum of log probabilities (log-likelihood)
log_likelihood_good = np.sum(log_probs_good)
log_likelihood_bad = np.sum(log_probs_bad)
print(f"\nLog-likelihood for good model: {log_likelihood_good:.2f}")
print(f"Log-likelihood for bad model: {log_likelihood_bad:.2f}")
```

Code Block 3

```
Log probabilities for good model:
[-0.91893853 -0.91893853 -0.91893853 -0.91893853 -0.91893853]

Log probabilities for bad model:
[-1.76393853 -2.19893853 -2.72393853 -3.33893853 -4.04393853]

Log-likelihood for good model: -4.59
Log-likelihood for bad model: -14.07
```



Picking the most probable parameters (w_i and b)
for the linear regression mode, or

the model which has the most **LIKELIHOOD** based on the present observation.

aka Maximum Likelihood Estimation (MLE)

$$P(y|x) = N(\hat{y}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{y-\hat{y}}{\sigma})^2} \qquad \sigma\ (variance)$$

$$\hat{y} = wx + b$$

or = $w_1x_1 + w_2x_2 + \cdots + w_nx_n + b = \Sigma_i w_i x_i +$ b for multi-variables

$$\sum_n logP(y_n|x_n;\theta)$$

$$=\text{n}\sum \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_n-wx_n-b)2}{2\sigma^2}\right)\right)$$

Linear Regression picked from Normal Distribution

$$= \text{-n} * log\frac{1}{\sqrt{2\pi\sigma^2}} - \sum_i \frac{(y_n-wx_n-b)2}{2\sigma^2}$$

Assuming constant variance

$$argmax(\sum_i logP(y_n|x_n;\theta)) = argmin(\sum_n(y_n - wx_n - b)^2)$$

$$= argmin(\sum_n(y_n - \hat{y})^2)$$

Mean Square Error (MSE)

Linear Regression is optimized by minimize mean square error (MSE)

Linear Regression and Maximum Likelihood Estimation (MLE)

Keys:

- Likelihood: Probability of observing the data given the model parameters.
- Log-Likelihood: Sum of log probabilities, used for numerical stability and easier optimization.
- Assumptions: Errors are normally distributed around the true regression line.

MLE Process:

-Define the likelihood function based on the probability distribution of errors.
-Find the parameters that maximize the log-likelihood.

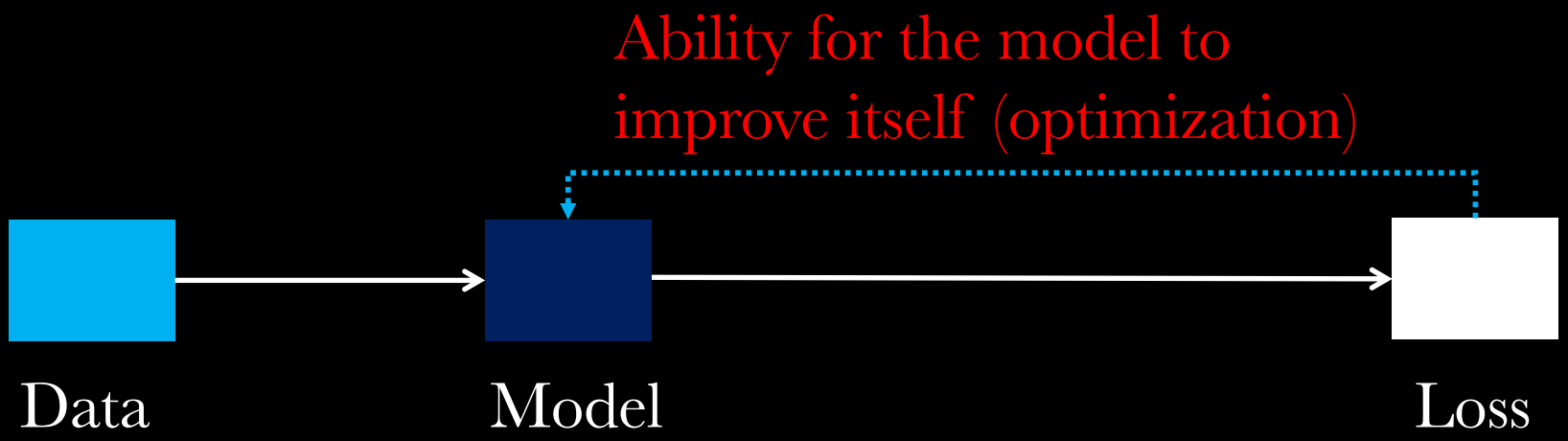
Advantages of MLE:

- Provides a principled way to estimate model parameters.
- Allows for comparison between different models using log-likelihoods

Regression by just called a library (Scipy)

```
from sklearn.linear_model import Regression

Regr = Regression()

Regr.fit(x_train, y_train)
```

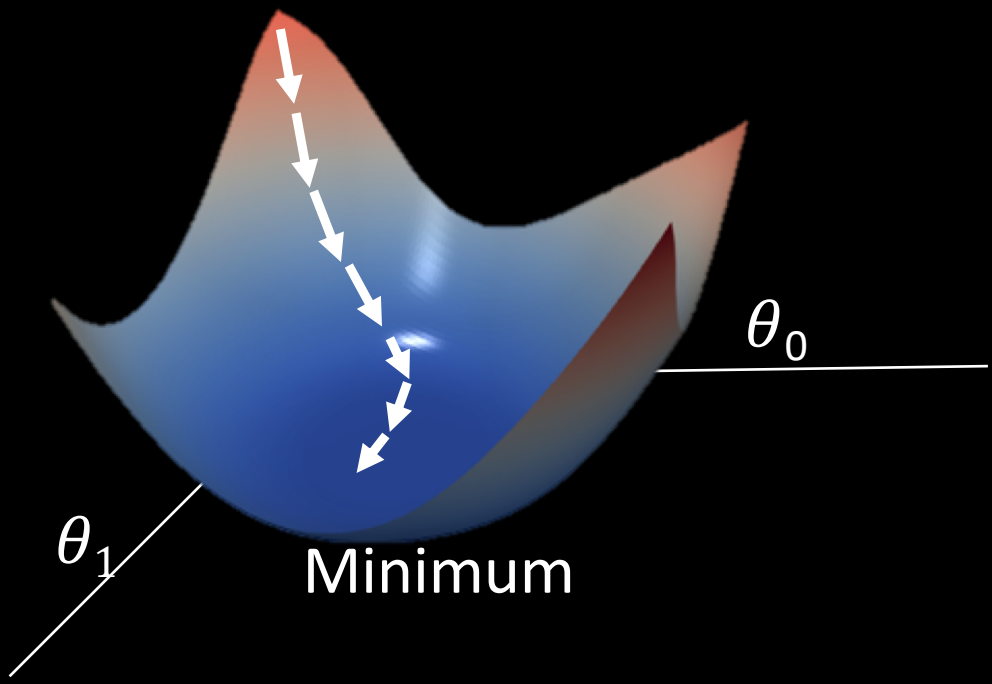
Behind the Scene Regr.fit(x_train, y_train)



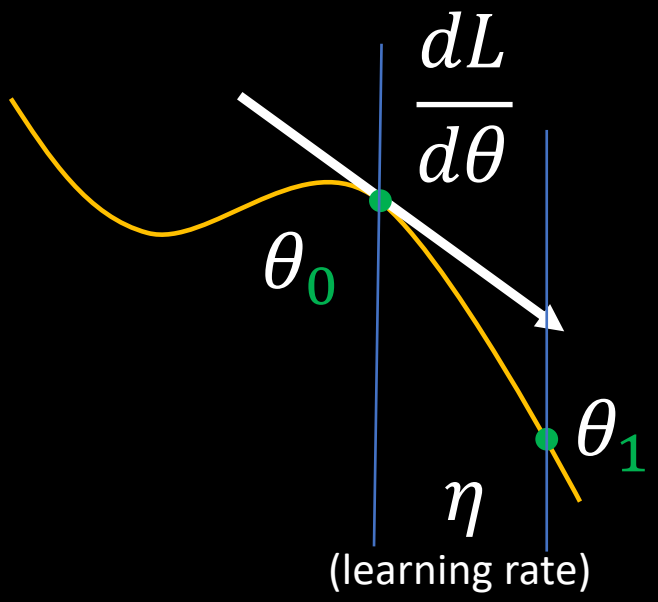
Gradient Descent

Moves in the direction of steepest descent (negative gradient) to minimize the objective function.

Steps:
1. Start with initial parameter values
2. Calculate the gradient of the objective function
3. Update parameters in the opposite direction of the gradient
4. Repeat steps 2-3 until convergence



Gradient Descent



$$\theta = argmin(\sum_n (y_n - \hat{y})^2) =$$

$$argmin(\sum_n (y_n - wx_n - b)^2)$$

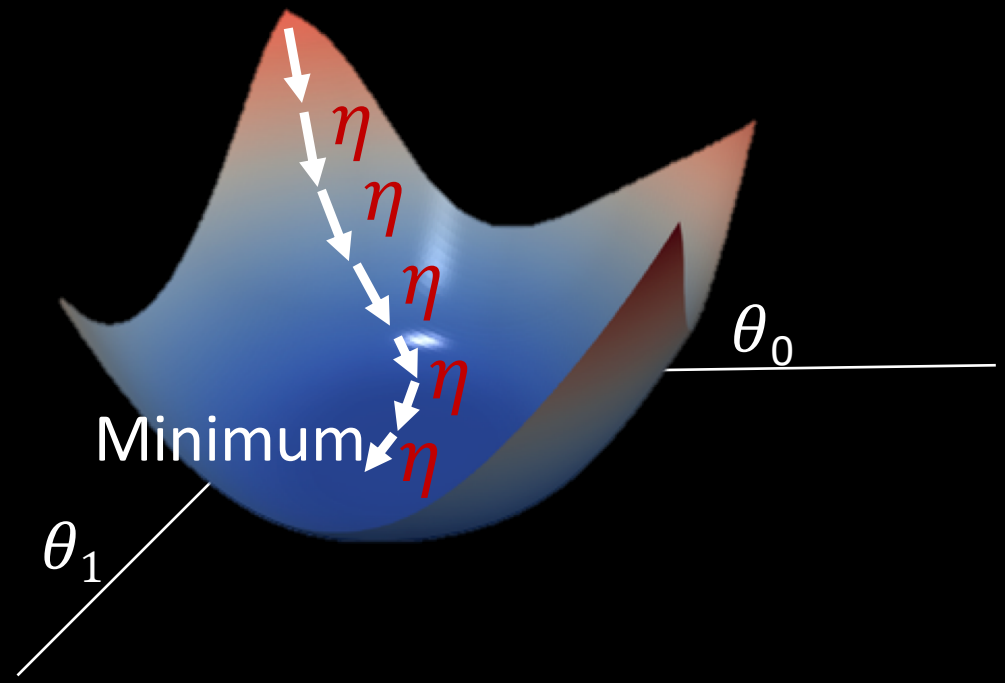
$$L = \sum_n (y_n - wx_n - b)^2$$

Loss function

$$\frac{dL}{dw} = \frac{}{dw}\sum_n (y_n - wx_n - b)^2$$

$$\frac{dL}{db} = \frac{}{db}\sum_n (y_n - wx_n - b)^2$$

Gradient of L to parameters



$$\frac{dL}{dw} = \frac{}{dw}\sum_n (y_n - wx_n - b)^2 \quad = -\sum_n 2x_n(y_n - wx_n - b)$$

$$\frac{dL}{db} = \frac{}{db}\sum_n (y_n - wx_n - b)^2 = -\sum_n 2(y_n - wx_n - b)$$

Remember chain rules?

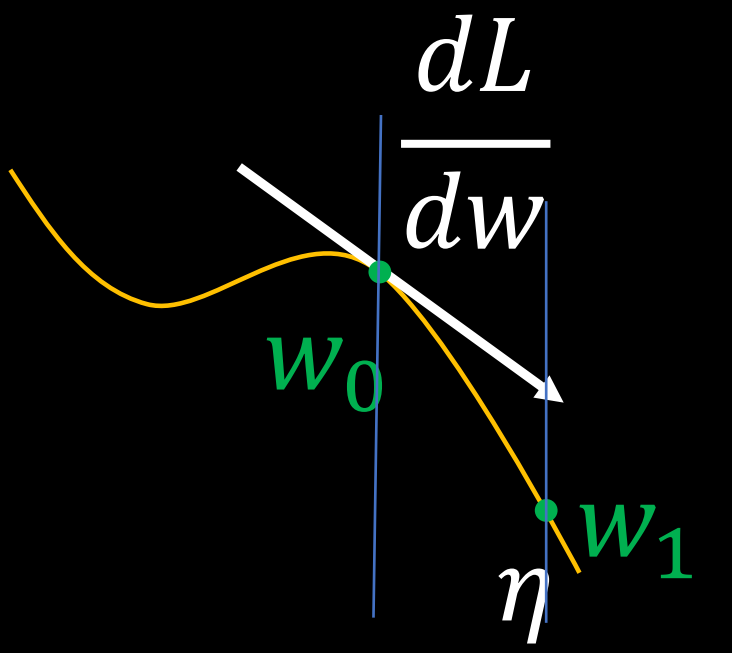
$$F'(x) = f'(g(x))\, g'(x)$$

```python
import numpy as np

# Sample data
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Two parameter sets
w1, b1 = 1.5, 1.0
w2, b2 = 1.5, 0.5

# Function to calculate predictions
def predict(X, w, b):
    return w * X + b

# Function to calculate Mean Squared Error (MSE) loss
def mse_loss(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)


# Function to calculate gradient of loss with respect to b
def gradient_b(y_true, y_pred):
    return -2 * np.mean(y_true - y_pred)
```

```python
# Calculate predictions, loss, and gradient for both parameter sets
y_pred1 = predict(X, w1, b1)
loss1 = mse_loss(y, y_pred1)
grad_b1 = gradient_b(y, y_pred1)


y_pred2 = predict(X, w2, b2)
loss2 = mse_loss(y, y_pred2)
grad_b2 = gradient_b(y, y_pred2)
```

Parameter set 1: w = 1.5, b = 1.0
Loss: 0.7500
Gradient dL/db: -1.0000

Parameter set 2: w = 1.5, b = 0.5
Loss: 1.5000
Gradient dL/db: -2.0000



$$w_1 = w_0 - \frac{dL}{dw}\eta$$

$$b_1 = b_0 - \frac{dL}{db}\eta$$

$$(w_0, b_0) \longrightarrow (w_1, b_1)$$

```python
import numpy as np
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Initial parameters
w, b = 1.5, 1.0

# Function to calculate predictions
def predict(X, w, b):
    return w * X + b

def mse_loss(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def gradient_w(X, y_true, y_pred):
    return -2 * np.mean(X * (y_true - y_pred))

def gradient_b(y_true, y_pred):
    return -2 * np.mean(y_true - y_pred)

# Calculate initial predictions, loss, and gradients
y_pred = predict(X, w, b)
initial_loss = mse_loss(y, y_pred)
grad_w = gradient_w(X, y, y_pred)
grad_b = gradient_b(y, y_pred)

# Perform one step of gradient descent
learning_rate = 0.01
w_new = w - learning_rate * grad_w
b_new = b - learning_rate * grad_b

# Calculate new predictions and loss
y_pred_new = predict(X, w_new, b_new)
new_loss = mse_loss(y, y_pred_new)
```

```python
print(f"Initial parameters: w = {w}, b = {b}")
print(f"Initial loss: {initial_loss:.4f}")
print(f"Gradient dL/dw: {grad_w:.4f}")
print(f"Gradient dL/db: {grad_b:.4f}")


print(f"\nAfter one gradient descent step:")
print(f"New parameters: w = {w_new:.4f}, b =
{b_new:.4f}")
print(f"New loss: {new_loss:.4f}")


print(f"\nLoss reduction: {improvement:.2f}%")
```

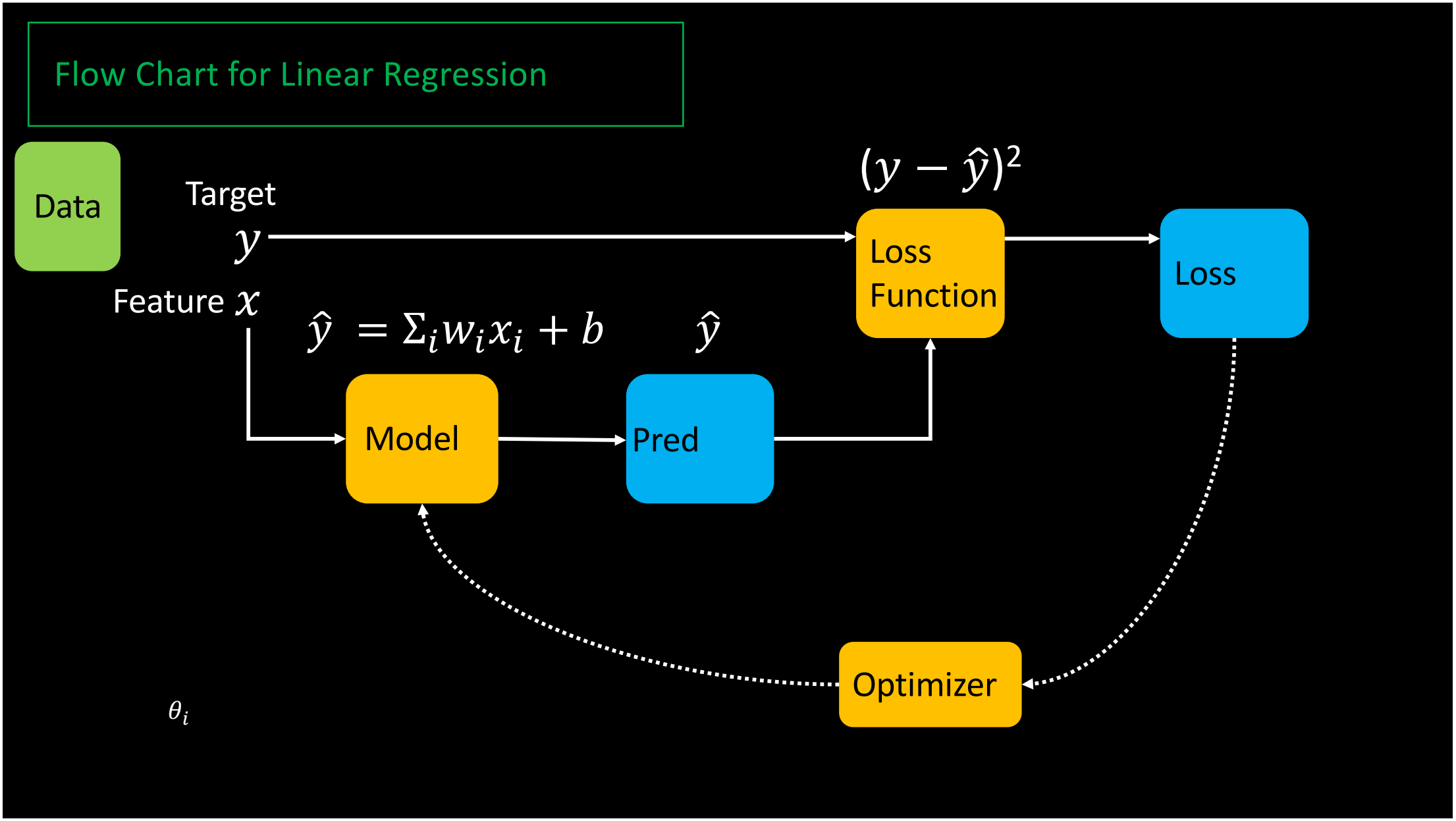
```
Initial parameters: w = 1.5, b = 1.0
Initial loss: 0.7500
Gradient dL/dw: -5.0000
Gradient dL/db: -1.0000

After one gradient descent step:
New parameters: w = 1.5500, b = 1.0100
New loss: 0.5206
```

Gradient Direction:

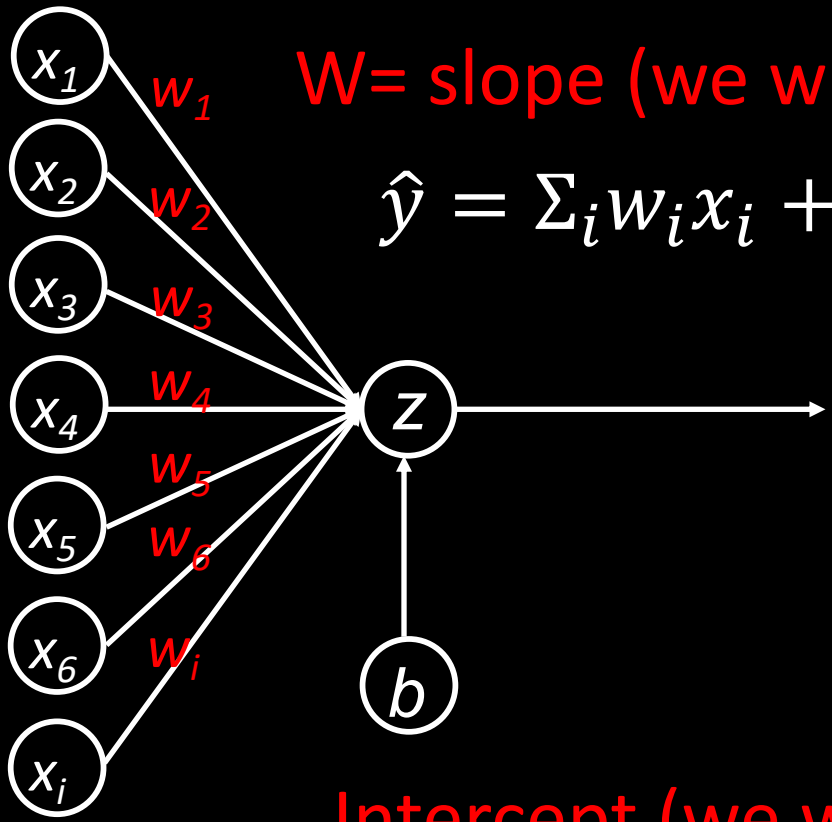
- Gradient ∇L points towards the steepest increase in loss
- We move in the opposite direction ($-\nabla L$) to minimize loss
- For each parameter θ: $\theta_{new} = \theta_{old} - \eta * \partial L/\partial \theta$

Learning Rate (η):

- Controls the step size in each iteration
- Too large: May overshoot the minimum, causing divergence
- Too small: Slow convergence, may get stuck in local minima


Flow Chart for Linear Regression
Data
Target
y
Feature x
$\hat{y} = \Sigma_i w_i x_i + b$
$\hat{y}$
$(y - \hat{y})^2$
Loss Function
Loss
Model
Pred
Optimizer
θ_i

Graph of (Multi-variable)Linear Regression



W= slope (we will call it weight by some point)

$$\hat{y} = \Sigma_i w_i x_i + b$$

Outcome

Intercept (we will call it bias by some point)

Classification: Logistic Regression

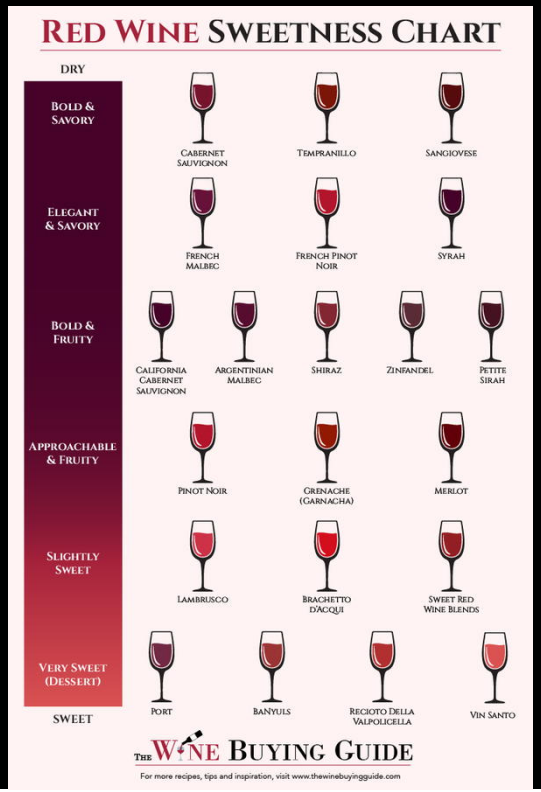
Classification vs Regression

TNM 8th - Primary tumor characteristics	
T_x	Tumor in sputum/bronchial washings but not be assessed in imaging or bronchoscopy
T_0	No evidence of tumor
T_{is}	Carcinoma in situ
T_1	≤ 3 cm surrounded by lung/visceral pleura, not involving main bronchus
$T_{1a(mi)}$	Minimally invasive carcinoma
T_{1a}	≤ 1 cm
T_{1b}	> 1 to ≤ 2 cm
T_{1c}	> 2 to ≤ 3 cm
T_2	> 3 to ≤ 5 cm *or* involvement of main bronchus without carina, regardless of distance from carina or invasion visceral pleural *or* atelectasis or post obstructive pneumonitis extending to hilum
T_{2a}	>3 to ≤4cm
T_{2b}	>4 to ≤5cm
T_3	>5 to ≤7cm in greatest dimension *or* tumor of any size that involves chest wall, pericardium, phrenic nerve *or* satellite nodules in the same lobe
T_4	> 7cm in greatest dimension *or* any tumor with invasion of mediastinum, diaphragm, heart, great vessels, recurrent laryngeal nerve, carina, trachea, oesophagus, spine *or* separate tumor in different lobe of ipsilateral lung
N_1	Ipsilateral peribronchial and/or hilar nodes and intrapulmonary nodes
2	Ipsilateral mediastinal and/or subcarinal nodes
3	Contralateral mediastinal or hilar; ipsilateral/contralateral scalene/ supraclavicular
M_1	Distant metastasis
M_{1a}	Tumor in contralateral lung or pleural/pericardial nodule/malignant effusion
M_{1b}	Single extrathoracic metastasis, including single non-regional lymphnode
M_{1c}	Multiple extrathoracic metastases in one or more organs

Human has a tendency (or need) to categorize things for our own perception

Table 2. Examples of the training data set (values are normalized between 0 and 1)

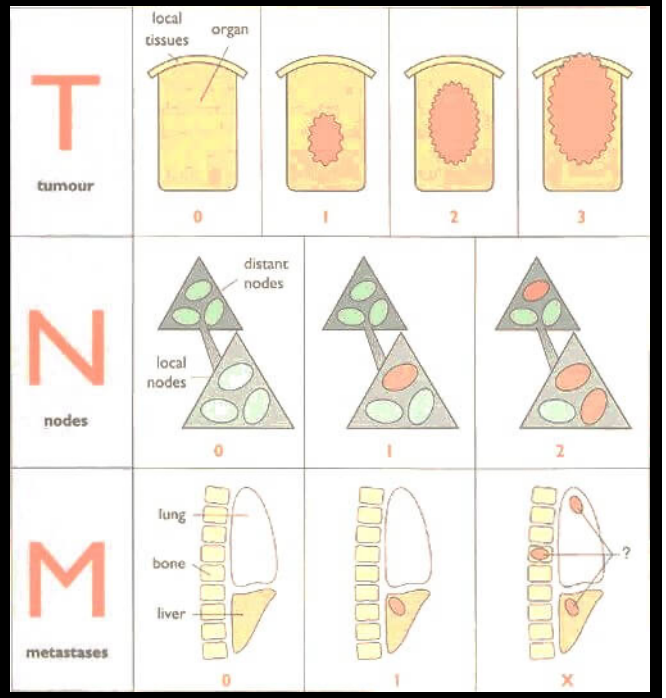
Fruit image	Input of the network		Output
	Mean	Variance	Fruit Grading
	0.43	0.11	'Extra'
	0.22	0.05	'Type 1'
	0.27	0.23	'Type 2'
	0.26	0.31	'Rejected'
	0.28	0.29	'Rejected'



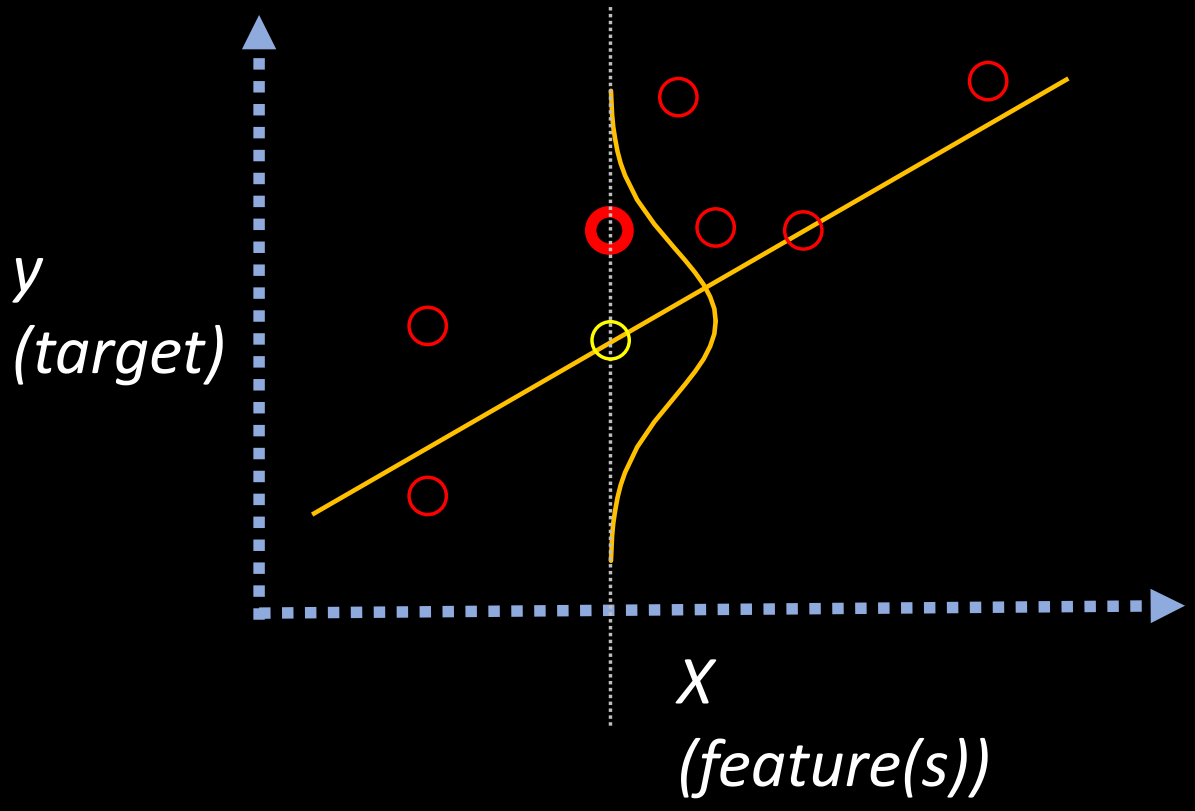
2023 Federal Tax Brackets

TAX BRACKET/RATE	SINGLE	MARRIED FILING JOINTLY	HEAD OF HOUSEHOLD
10%	$0 - $11,000	$0 - $22,000	$0 - $15,700
12%	$11,001 - $44,725	$22,001 - $89,450	$15,701 - $59,850
22%	$44,726 - $95,375	$89,451 - $190,750	$59,851 - $95,350
24%	$95,376 - $182,100	$190,751 - $364,200	$95,351 - $182,100
32%	$182,101 - $231,250	$364,201 - $462,500	$182,101 - $231,250
35%	$231,251 - $578,125	$462,501 - $693,750	$231,251 - $578,100
37%	$578,126+	$693,751+	$578,101+

THE COLLEGE INVESTOR Source: TheCollegeInvestor.com



Linear Regression: Predicts continuous outcomes

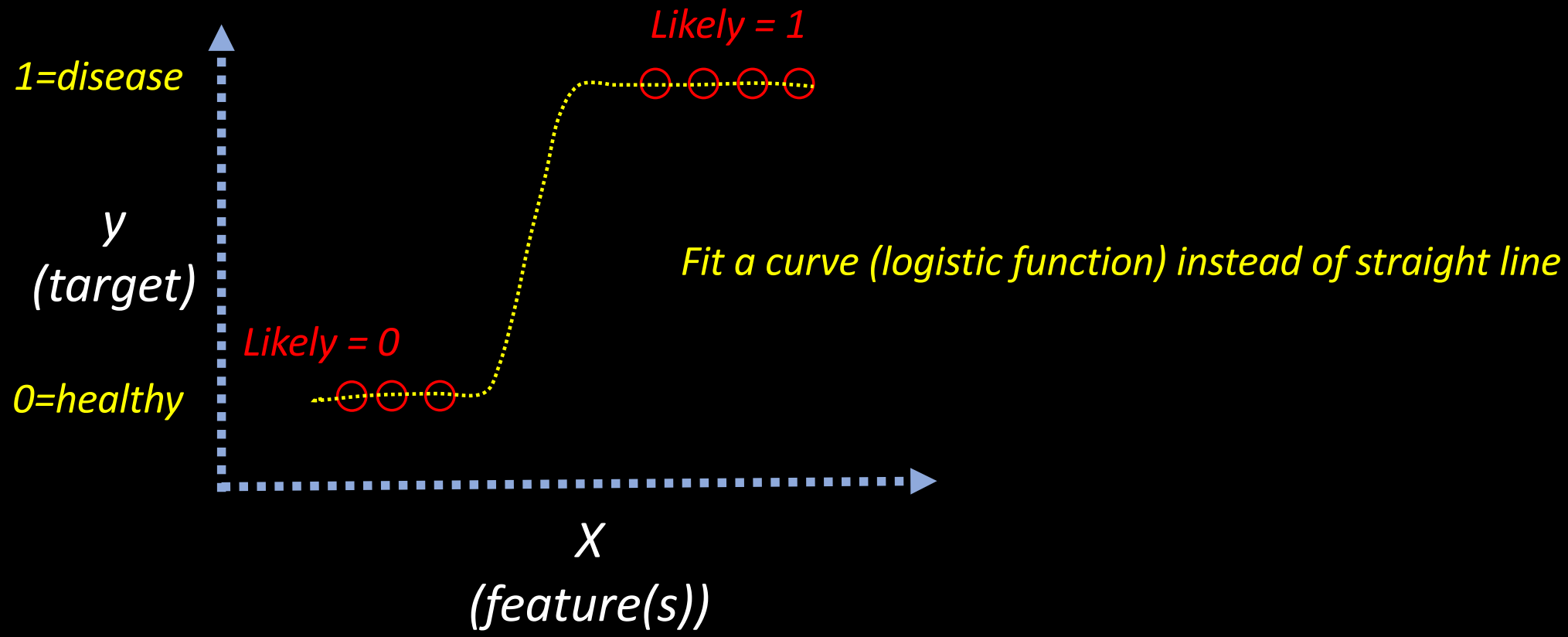


Linear regression is a method of predicting an outcome based on input data.

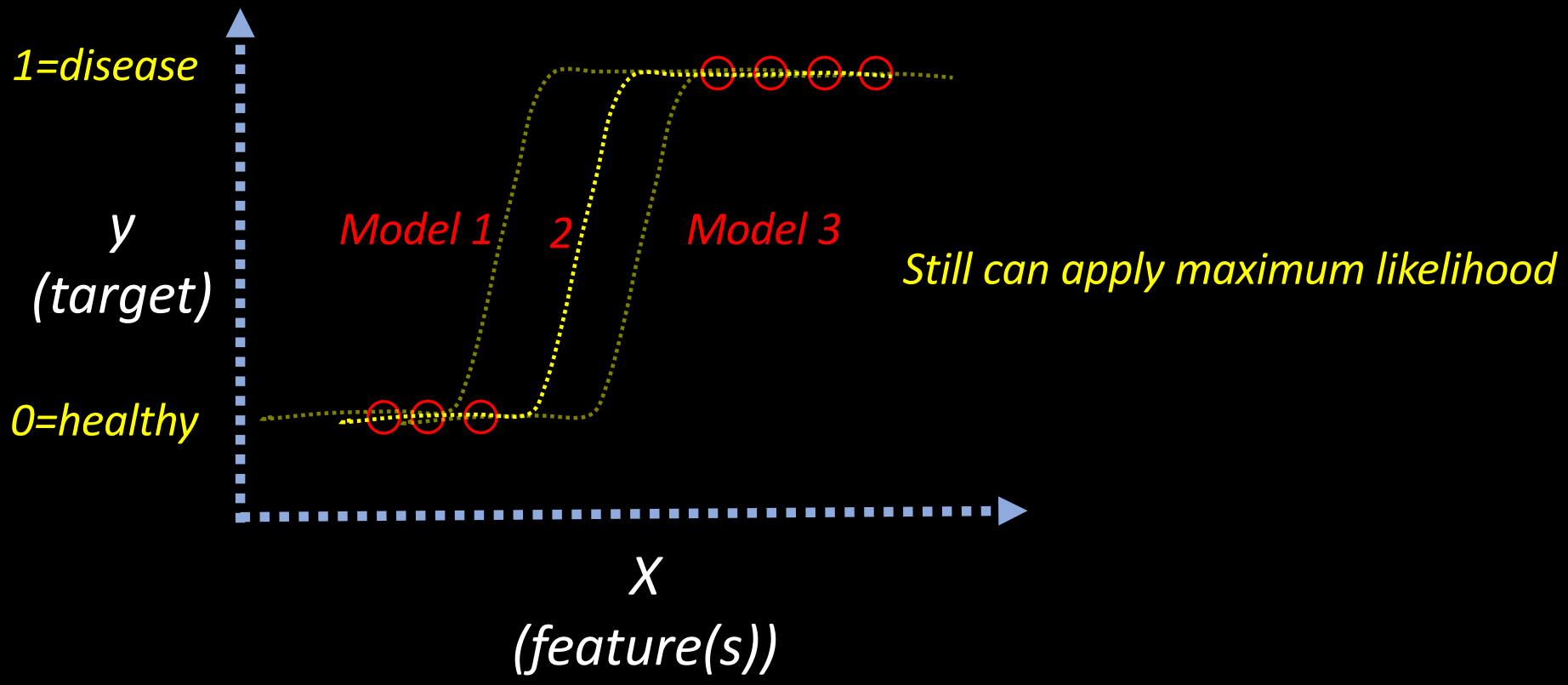
A line represents the best-fit linear relationship between weight and size.

This line can be used to make predictions: given a new weight value, we can estimate the corresponding size.

Logistic Regression: Predicts categorical outcomes



Logistic Regression: Predicts categorical outcomes



Regression -> classification, how?

regression $\Sigma_i w_i x_i + b \quad -\infty \sim \infty$ $y_i \; -\infty \sim \infty$

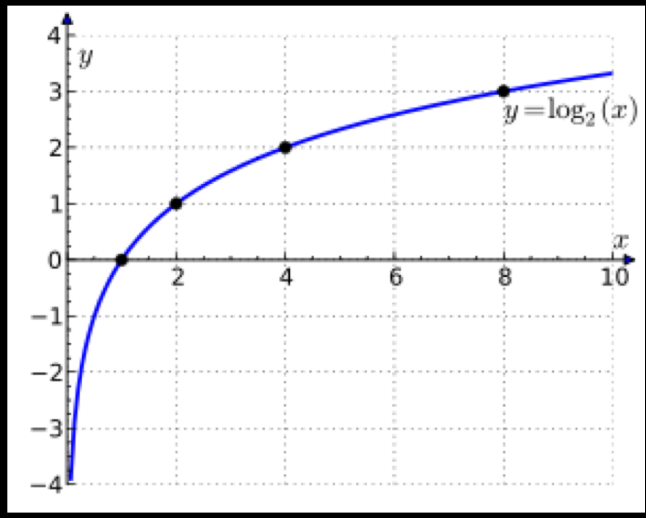
classification $0 < P(Y = truck) < 1$ $P(Y = car) = 1 - P(Y = truck)$

We want to find a way to map $-\infty \sim \infty$ to 0-1

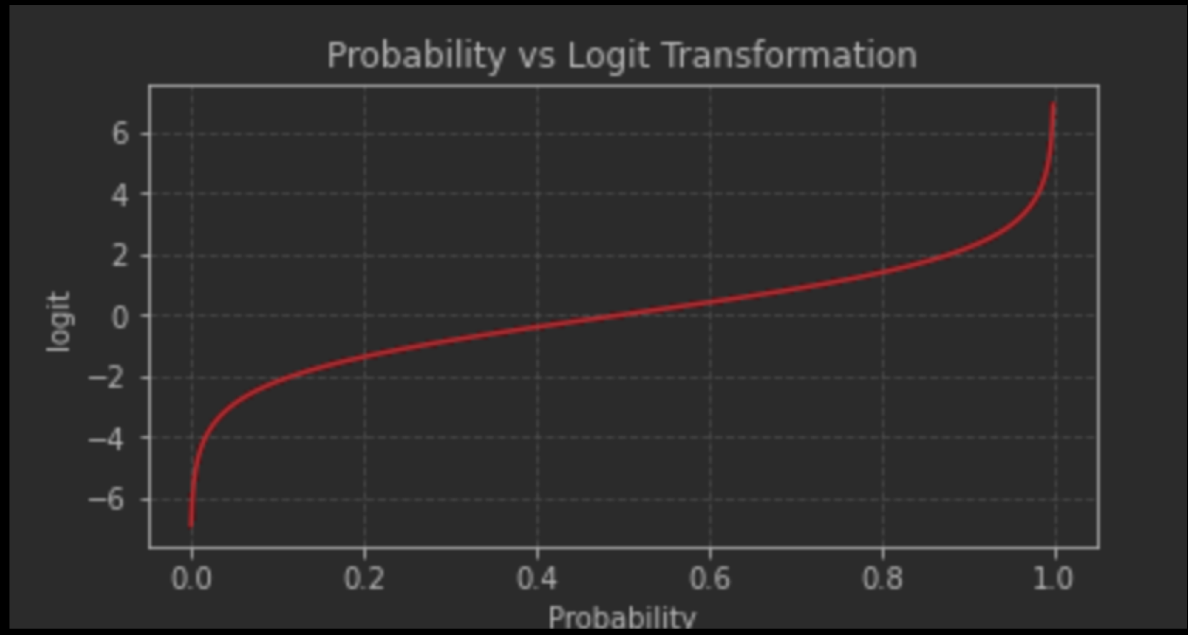
Probability p $0\sim1$

Odd Ratio $\frac{p}{1-p}$ $0\sim\infty$

$\Sigma_i w_i x_i + b$ $-\infty\sim\infty$ *Logit* $\log(\frac{p}{1-p})$ $-\infty\sim\infty$



```
# Create probability values from 0 to 1 (excluding 0 and 1 to avoid
infinity)
p = np.linspace(0.001, 0.999, 1000)

# Calculate logit values
logit = np.log(p / (1 - p))

# Create the plot
plt.figure(figsize=(6, 3))

# Plot logit
plt.plot(p, logit, label='Logit', color='red')

# Add labels and title
plt.xlabel('Probability')
plt.ylabel('logit')
plt.title('Probability vs Logit Transformation')

# Add grid for better readability
plt.grid(True, linestyle='--', alpha=0.7)
```



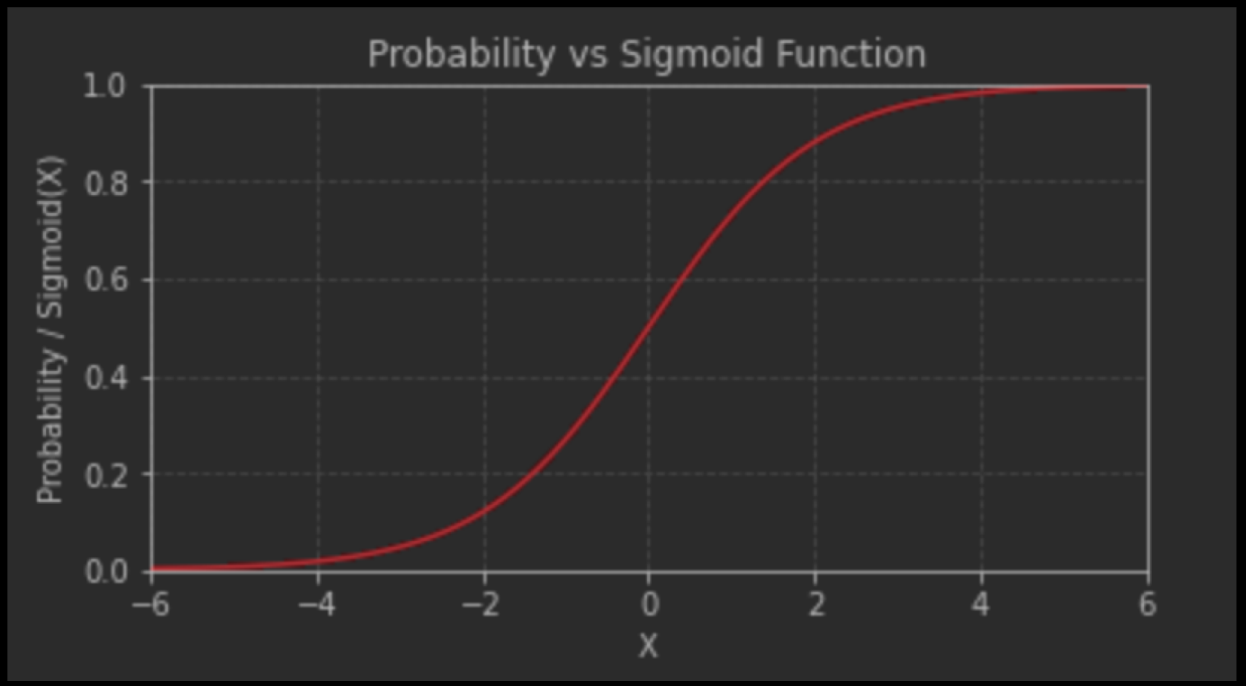
$\Sigma_i w_i x_i + b \; -\infty \sim \infty$ $\quad$ *Logit* $\quad \log(\frac{p}{1-p}) \quad -\infty \sim \infty$

Now we have a good target for regression:

$$\Sigma_i w_i x_i + b = \log(\frac{p}{1-p})$$

$$\exp(\Sigma_i w_i x_i + b) = \exp(\log(\frac{p}{1-p})) = \frac{p}{1-p}$$

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

# Create x values from -6 to 6
x = np.linspace(-6, 6, 1000)

# Calculate sigmoid values
y_sigmoid = sigmoid(x)

# Create the plot
plt.figure(figsize=(6, 3))

# Plot sigmoid
plt.plot(x, y_sigmoid, label='Sigmoid', color='red')

# Add labels and title
plt.xlabel('X')
plt.ylabel('Probability / Sigmoid(X)')
plt.title('Probability vs Sigmoid Function')
```



	No. of case patients	No. of control subjects	Odds ratio	95% confidence interval	*P* for trend*
Nonsmokers	117	1750	1.0	Referent	
Duration of tobacco use, y					
0.1–20.0	3	33	1.3	0.4–4.5	
20.1–32.0	7	33	3.4	1.4–8.0	
32.1–44.0	21	33	13.3	7.2–24.9	
⩾44.1	30	30	19.1	10.4–35.1	<.0001
Average consumption of tobacco, g/day					
0.1–3.5	2	10	2.2	0.5–10.4	
3.6–5.0	22	54	7.9	4.3–14.3	
5.1–10.7	6	18	4.8	1.9–12.6	
⩾10.8	31	47	12.4	7.2–21.4	.1
Cumulative consumption of tobacco, g/day × y					
1–71	3	31	1.3	0.4–4.4	
72–157	13	34	7.6	3.8–15.4	
158–382	15	32	8.0	4.0–15.7	
⩾383	30	32	18.3	10.2–32.8	.0006
Age at start of tobacco use, y					
⩽19	27	48	9.6	5.6–16.7	
20–26	20	52	6.3	3.5–11.2	
⩾27	14	29	8.2	4.1–16.3	.4

*Test for linear trend, two-sided *P* value (considered statistically significant for $P<.05$).

$$\Sigma_i w_i x_i + b = \log\left(\frac{p}{1-p}\right)$$

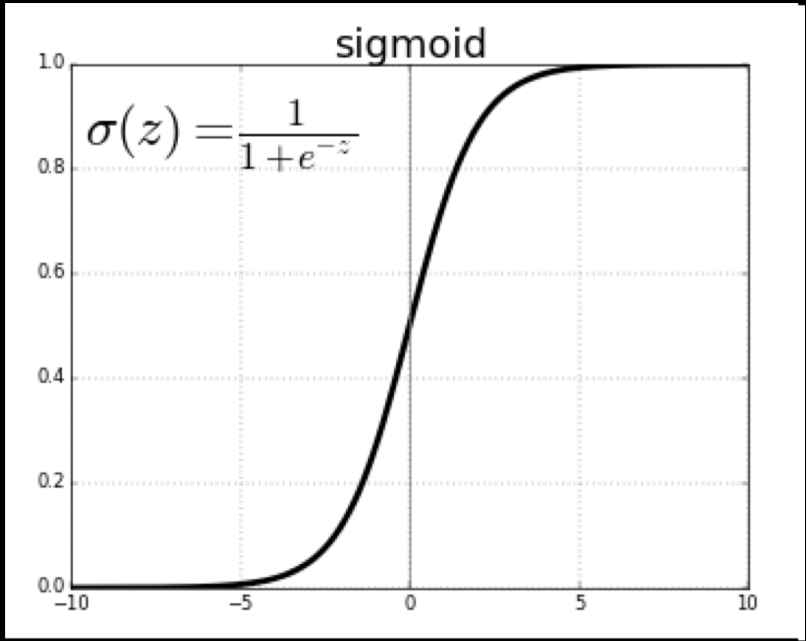
$$\exp(\Sigma_i w_i x_i + b) = \exp\left(\log\left(\frac{p}{1-p}\right)\right)$$

$$\exp(\Sigma_i w_i x_i + b) = \frac{p}{1-p}$$

$$p = \frac{exp(\Sigma_i w_1 x_i + b)}{1 + exp(\Sigma_i w_1 x_i + b)}$$

$$p = \frac{1}{1 + exp - (\Sigma_i w_i x_i + b)} = \text{Sigmoid}(w_i x_i + b) = S(\beta X)$$

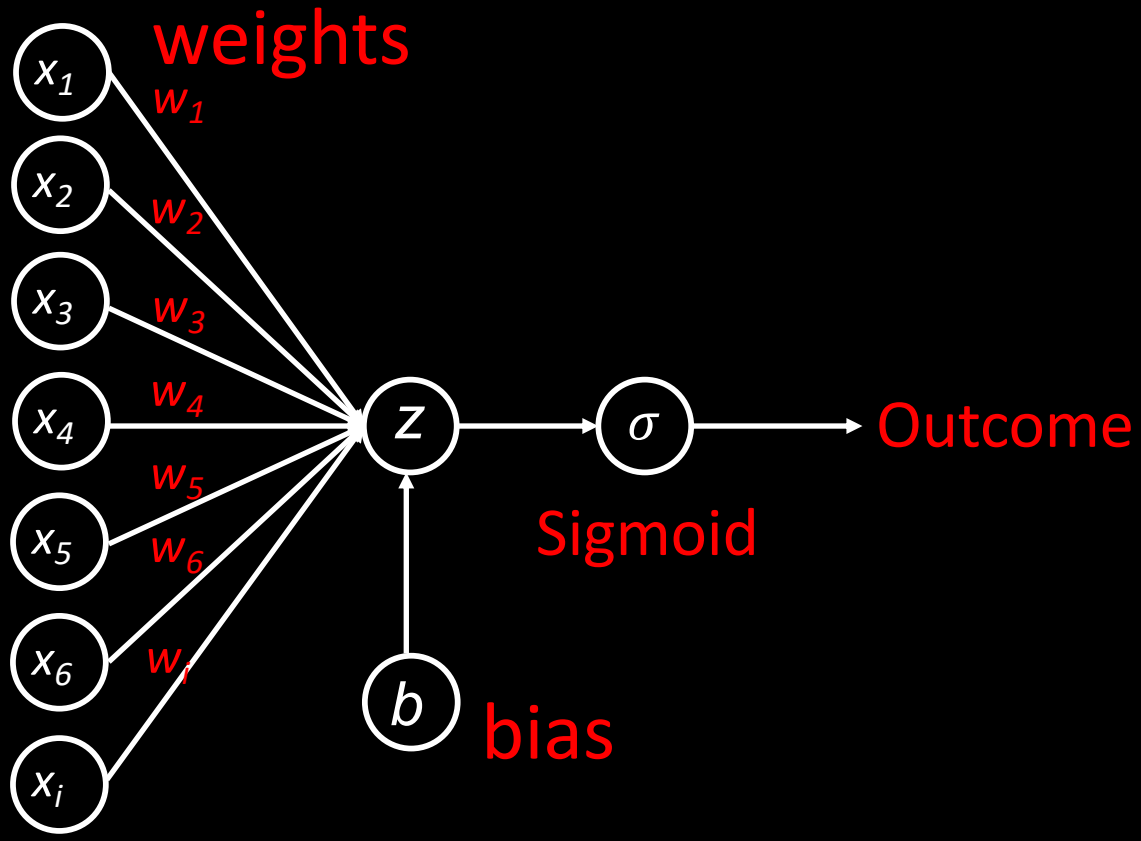
$$P = S(\beta X)$$



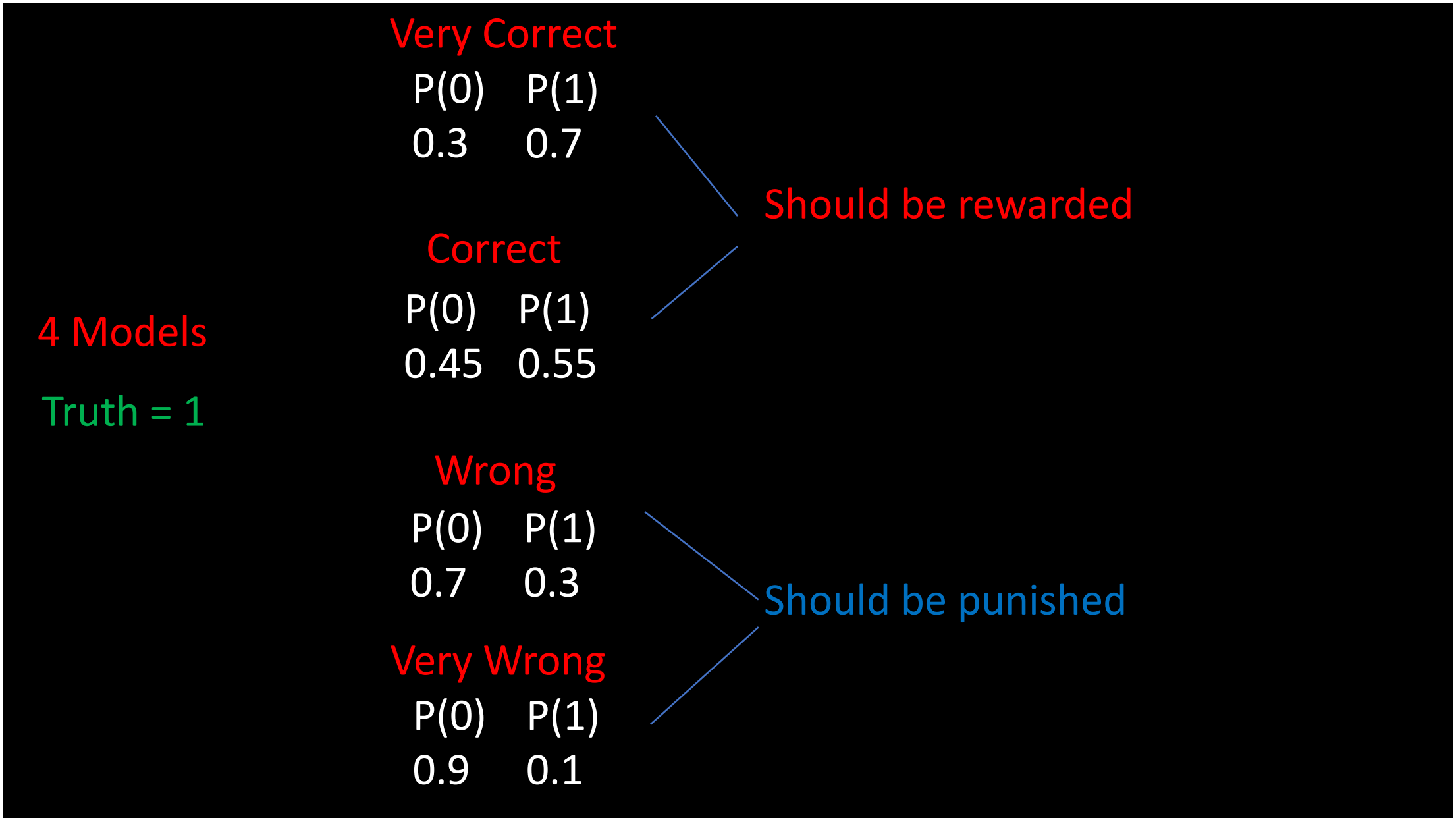
Logistic Regression = Regression Between Probability and Sigmoid of (weighted features)

Graph of Logistic Regression

$$p = \frac{1}{1+exp^{-(\Sigma_i w_i x_i + b)}} = \text{Sigmoid}(\Sigma_i w_i x_i + b) = S(WX)$$



Loss Function for Classification


Very Correct
P(0) P(1)
0.3 0.7
Correct
P(0) P(1)
0.45 0.55
Should be rewarded
4 Models
Truth = 1
Wrong
P(0) P(1)
0.7 0.3
Very Wrong
P(0) P(1)
0.9 0.1
Should be punished

4 Models

Truth = 1

Very Correct

P(0)	P(1)
0.3	0.7

Should be rewarded more

Correct

P(0)	P(1)
0.45	0.55

Should be rewarded less

Wrong

P(0)	P(1)
0.7	0.3

Should be punished less

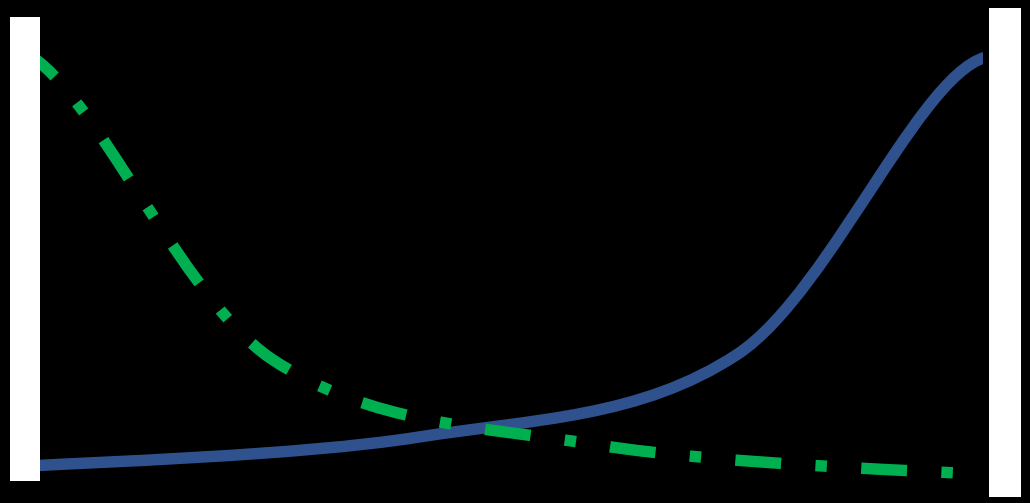
Very Wrong

P(0)	P(1)
0.9	0.1

Should be punished more

Maximum Likelihood Estimation (MLE)

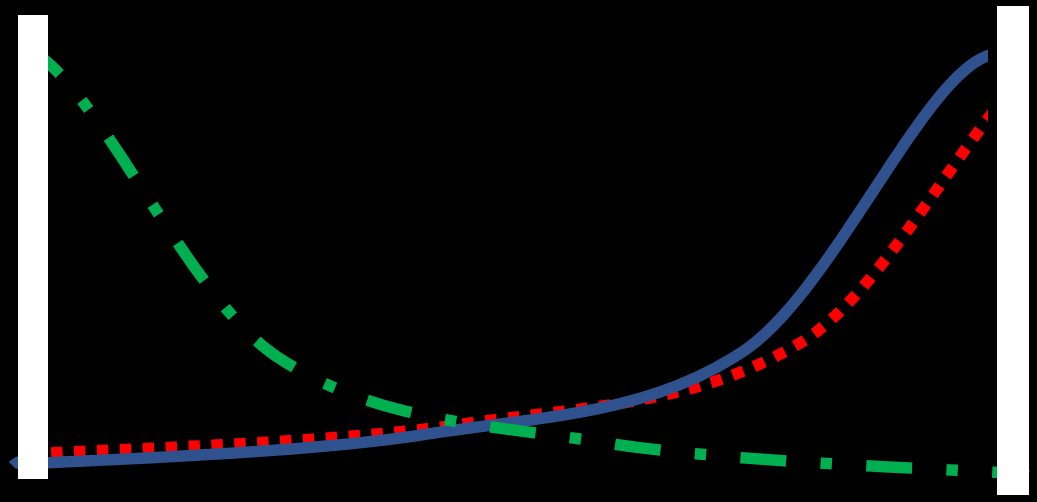
$$\theta_{ML} = argmax_{\theta} P(y|X;\theta)$$



Q, Real Probability

$P(y|X;\theta)$, prediction

Cross Entropy:
how different they are?



Q, Real Probability

$P(y|X;\theta)$, goodd prediction

$P(y|X;\theta)$, bad prediction

Cross Entropy:

how different they are?

$$H(Q,P) = -\sum_n Q_i \cdot \log(P_i)$$

Q	***P***
0.9	0.9
0.1	0.1
0.9	0.1
0.1	0.9

(0.09)
(0.23) → 0.32 *Good*

(2.07)
(0.01) → 2.08 *Bad*

(Binary Cross Entropy)

$$BCE =$$

$$-\sum_n [\mathrm{y} \cdot \log(p(\hat{y})) \quad + \quad (1-y) \cdot \log(p(1-\hat{y}))]$$

Loss if y=1

Loss if y=0

Or simply $-\sum_n [\mathrm{y} \cdot \log(p(\hat{y}))]$ for multi − classes

$$-\sum_n [\mathrm{y} \cdot \log(p(\hat{y})) + \quad (1-y) \cdot \log(p(1-\hat{y}))]$$

Truth = 1

Correct

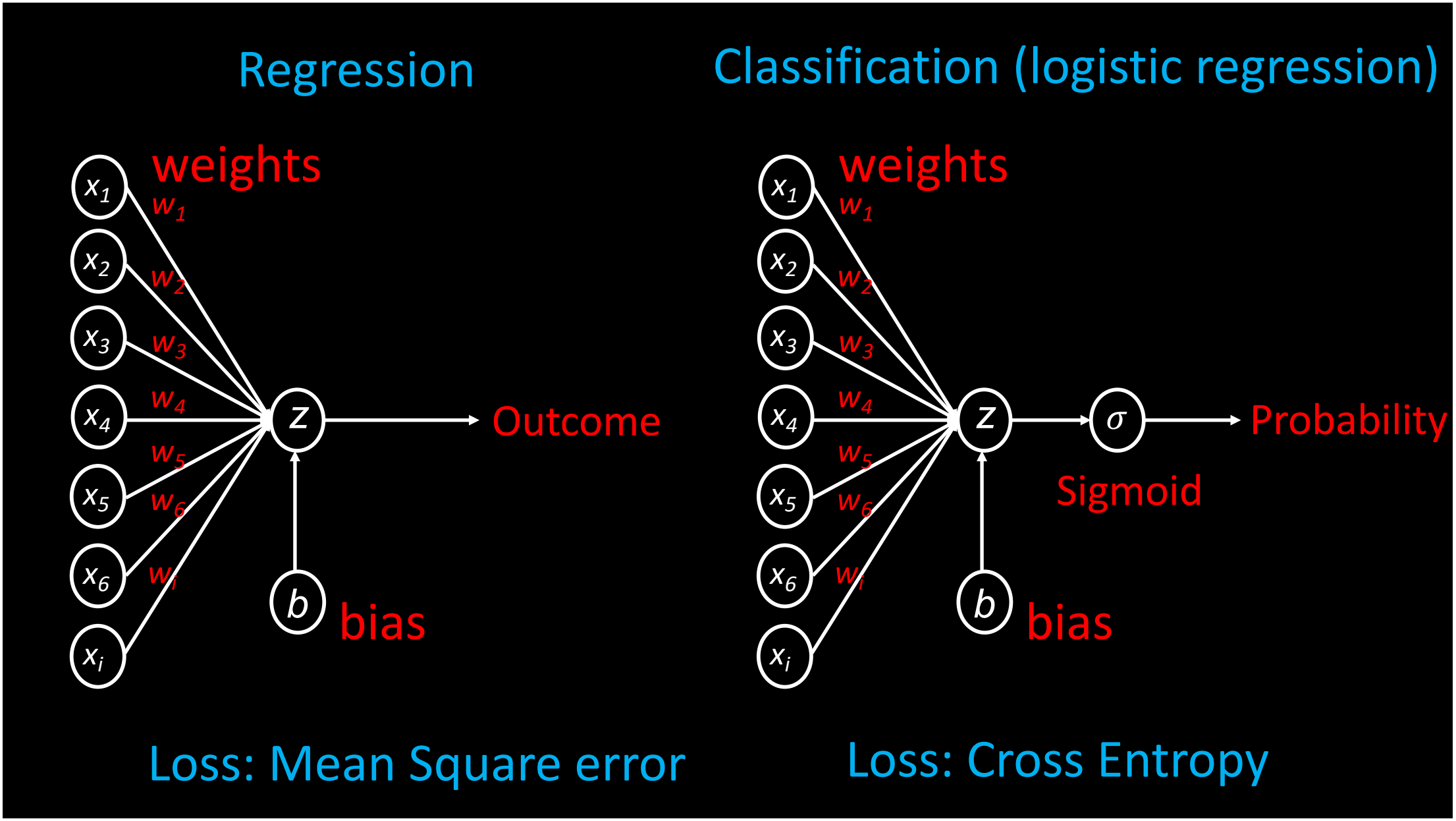
P(0)	P(1)
0.45	0.55

1 log(0.55) 0 log(0.45) =0.260

Very Correct

P(0)	P(1)
0.3	0.7

1 log(0.7) 0 log(0.3) =0.155


Regression
weights
x_1
x_2
x_3
x_4
x_5
x_6
x_i
w_1
w_2
w_3
w_4
w_5
w_6
w_i
z
b
bias
Outcome
Loss: Mean Square error
Classification (logistic regression)
weights
x_1
x_2
x_3
x_4
x_5
x_6
x_i
w_1
w_2
w_3
w_4
w_5
w_6
w_i
z
σ
Sigmoid
b
bias
Probability
Loss: Cross Entropy

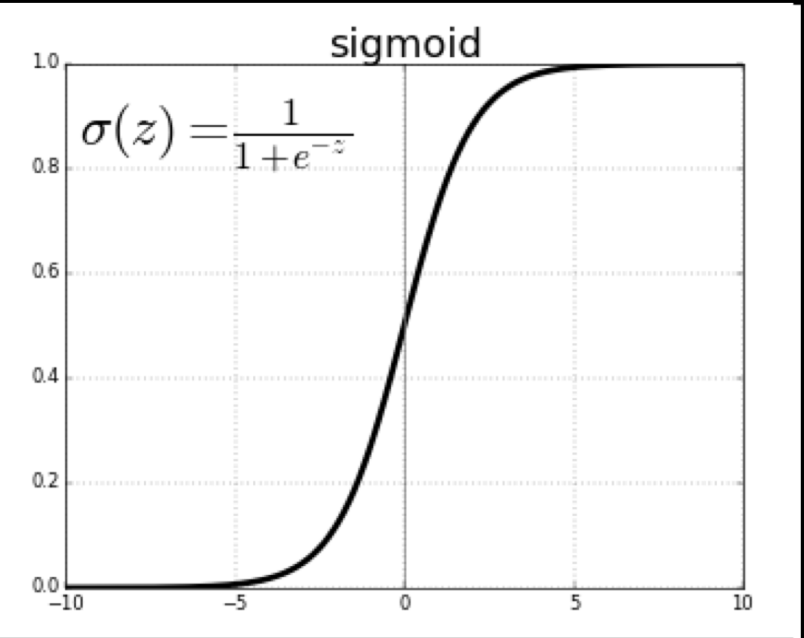
In practice!

```
model = LinearRegression()
model.fit(weights, dosages)

log_model = LogisticRegression()
log_model.fit(cholesterol, heart_disease)
```

But now you know!

Multi-class Logistic Regression

$$p = \frac{1}{1+exp^{-(\Sigma_i w_i x_i + b)}} = \text{Sigmoid}(w_i x_i + b) = S(WX)$$



Logistic Regression = Regression Between Probability and Sigmoid of (weighted features)

$$p = \frac{1}{1+exp-(\Sigma_i w_i x_i + b)} = \text{Sigmoid}(w_i x_i + b) = S(WX)$$

v s

$$p(y=0) = SX(W_0 X) = \frac{exp-(W_0 X)}{\sum_n exp-(W_n X)}$$

$$p(y=1) = SX(W_1 X) = \frac{exp-(W_1 X)}{\sum_n exp-(W_n X)}$$

•
•
•

$$p(y=N) = SX(W_N X) = \frac{exp-(W_N X)}{\sum_n exp-(W_n X)}$$

For multi-classes, you will have one *W* for each class

The summation for all the P Should be 1

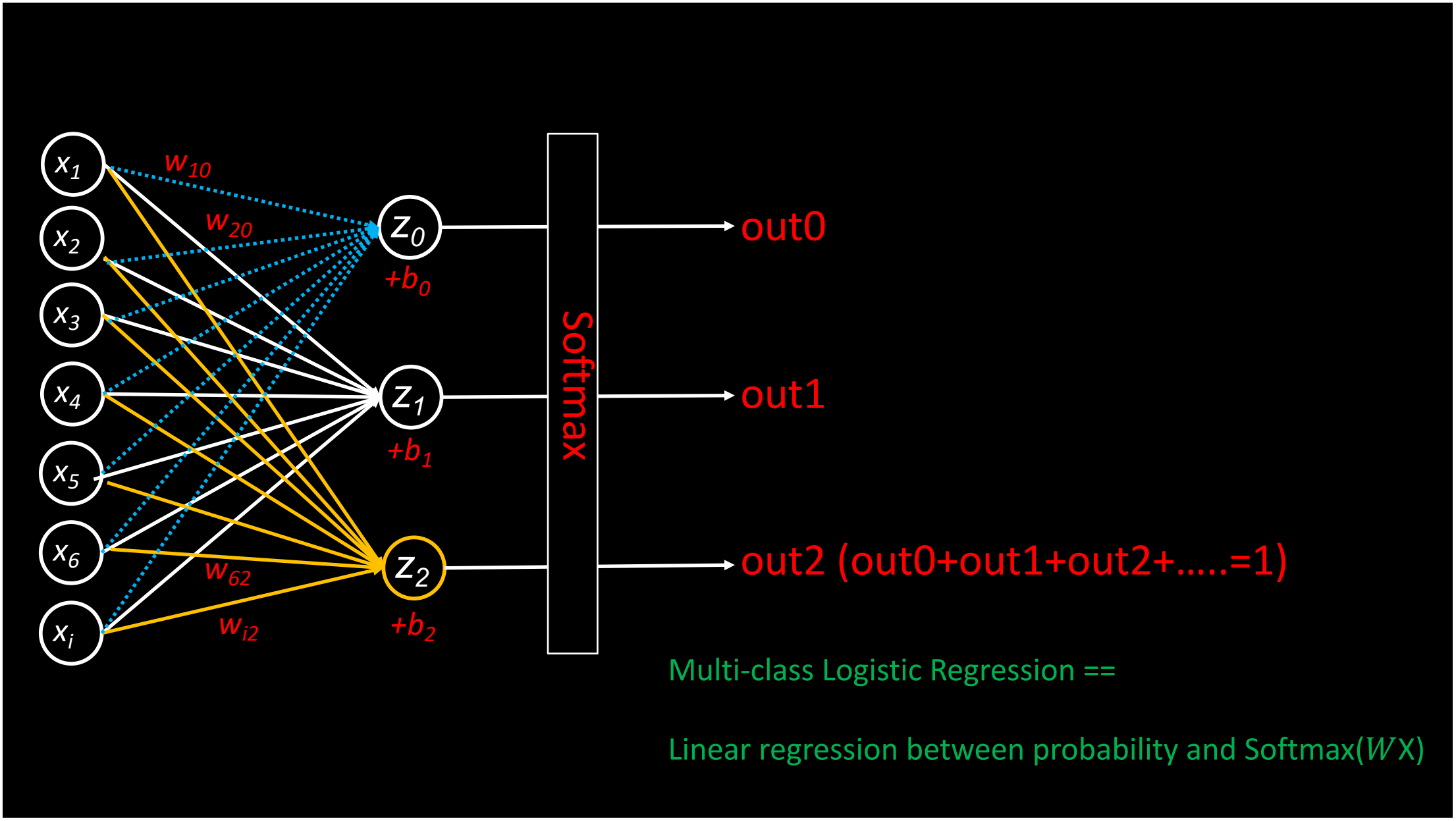
	Output (Wx)	exp(Wx)	Softmax exp(Wx) / sum(exp(Wx)))
1	3.08	21.76	0.24
2	2.87	17.63	0.19
3	-0.84	0.43	0.0047
4	2.69	14.73	0.16
5	1.76	5.81	0.064
6	-0.69	0.50	0.0055
7	3.03	20.697	0.23
8	-4.14	0.016	0.00017
9	2.19	8.93	0.098
10	-2.93	0.054	0.00005

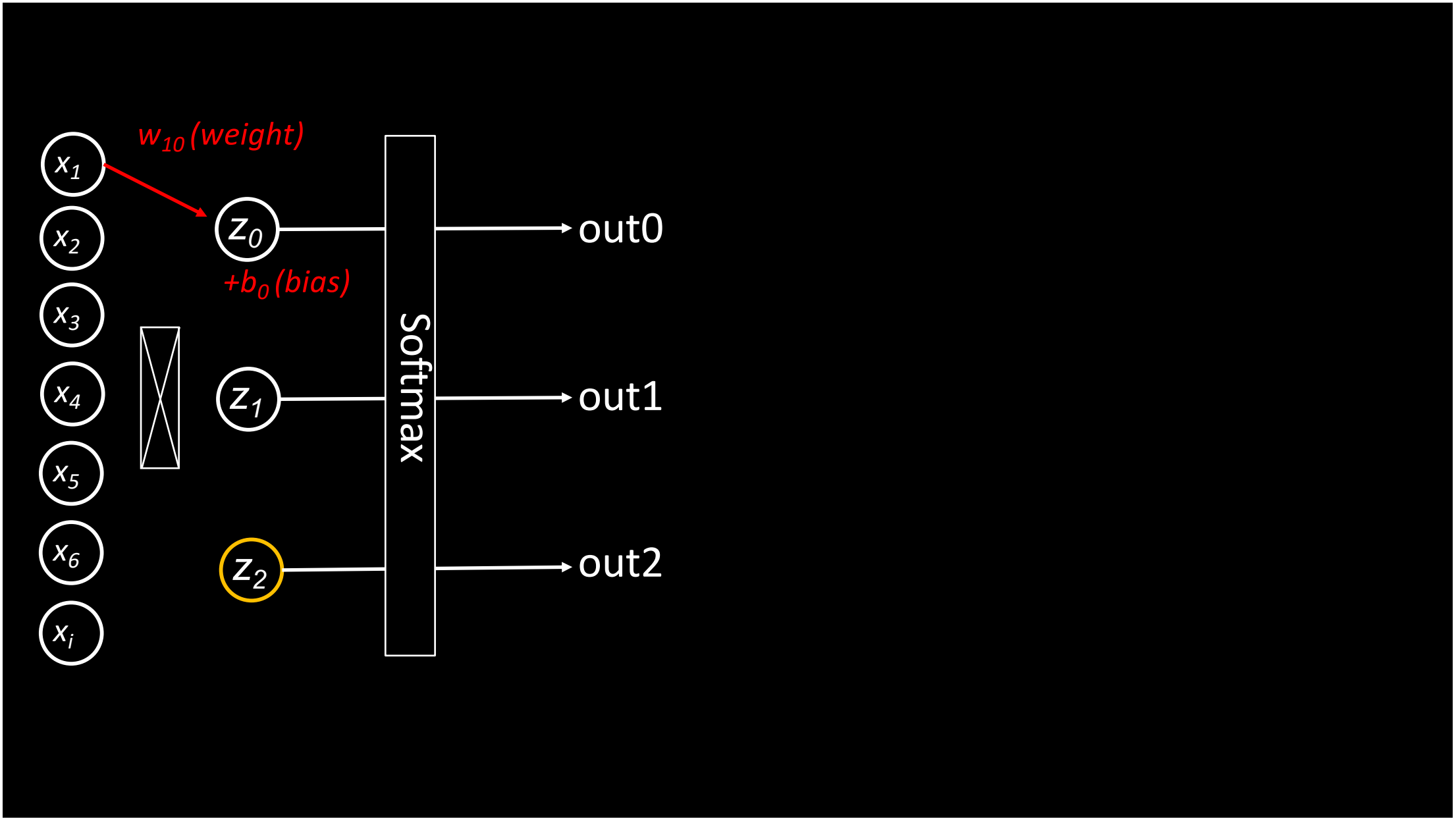
$$p(y = N) = Softmax(W_N X) = \frac{exp-(W_N X)}{\sum_n exp-(W_n X)}$$

Softmax ==
Sigmoid for multi-class

	Output (Wx)	exp(Wx)	Softmax	argmax
1	3.08	21.76	0.24	1
2	2.87	17.63	0.19	0
3	-0.84	0.43	0.0047	0
4	2.69	14.73	0.16	0
5	1.76	5.81	0.064	0
6	-0.69	0.50	0.0055	0
7	3.03	20.697	0.23	0
8	-4.14	0.016	0.00017	0
9	2.19	8.93	0.098	0
10	-2.93	0.054	0.00005	0

Softmax = “softer” version of argmax


x_1
x_2
x_3
x_4
x_5
x_6
x_i
w_{10}
w_{20}
w_{62}
w_{i2}
z_0
z_1
z_2
$+b_0$
$+b_1$
$+b_2$
Softmax
out0
out1
out2 (out0+out1+out2+.....=1)
Multi-class Logistic Regression ==
Linear regression between probability and Softmax(WX)


w_{10} (weight)
x_1
x_2
x_3
x_4
x_5
x_6
x_i
z_0
$+b_0$ (bias)
z_1
z_2
Softmax
out0
out1
out2

Binary Cross Entropy

$$-\sum_n [\mathrm{y} \cdot \log(p(\hat{y})) + (1-y) \cdot \log(p(1-\hat{y}))])$$

Multi-Class Cross Entropy

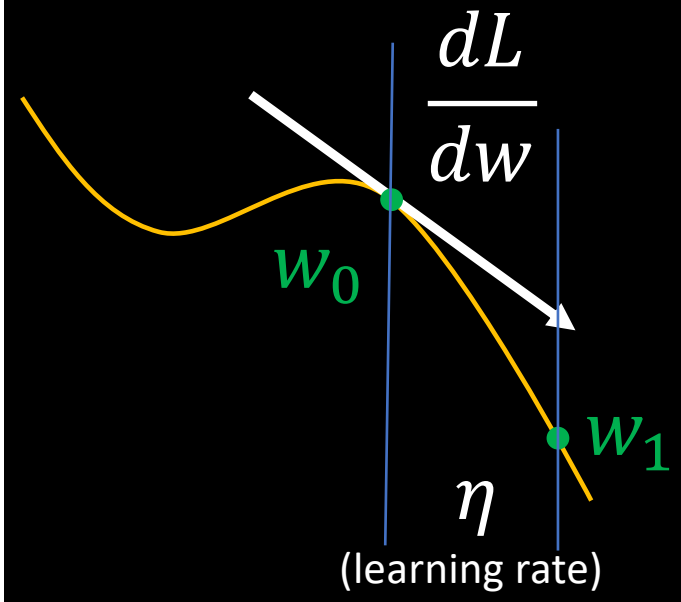
$$-\sum_n \mathrm{y} \cdot \log(p(\hat{y}))$$

y	$\hat{y}$		
0	0.6		
1	0.3	=	-1 * log(0.3)
0	0.1		

Cross Entropy for Classification
MSE for Regression

All because of maximum likelihood

MATH!



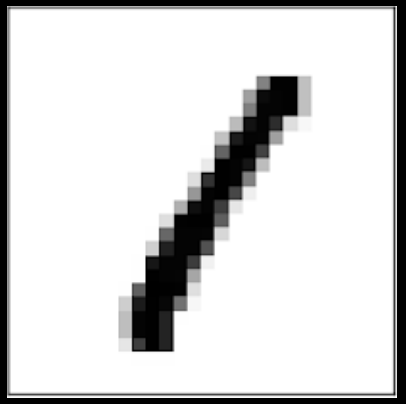
$$w_1 = w_0 - \frac{dL}{dw}\eta$$

$$loss\ function\ (L) = \sum_n (y_n - wx_n - b)^2$$

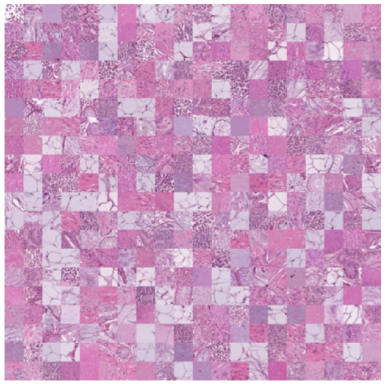
$$\frac{\partial L}{\partial w} = \sum_n 2(y_n - wx_n - b) \cdot (-x_n) = \sum_n 2(wx_n + b - y_n) \cdot (x_n)$$

$$\frac{\partial L}{\partial b} = \sum_n 2(y_n - wx_n - b) \cdot (-1) = \sum_n 2(wx_n + b - y_n)$$

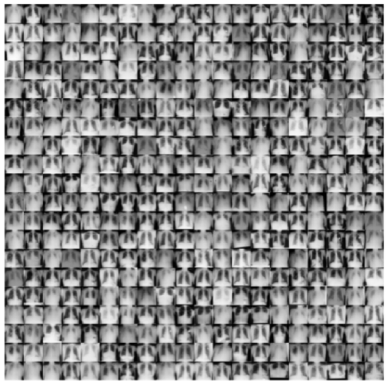
MNIST dataset



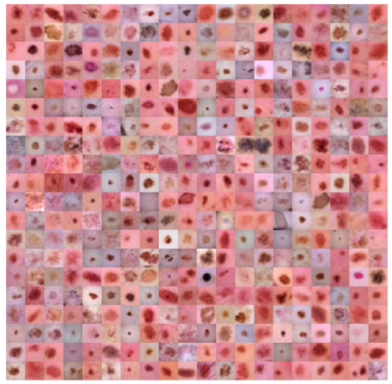
PathMNIST



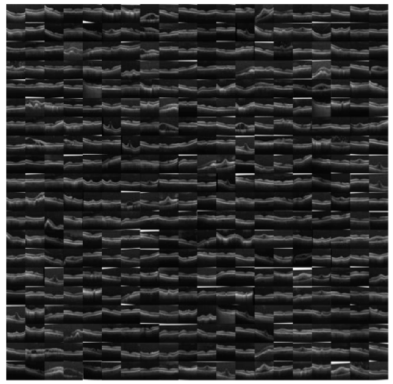
ChestMNIST



DermaMNIST



OCTMNIST

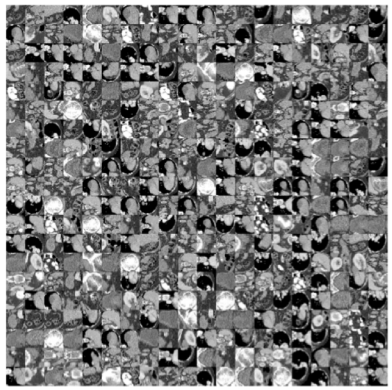


PneumoniaMNIST

RetinaMNIST

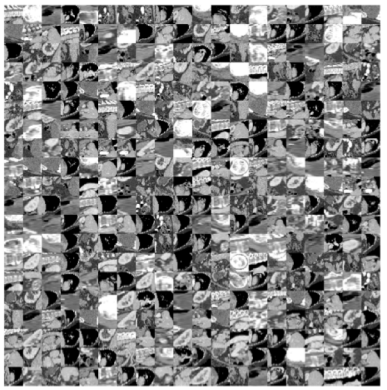
BreastMNIST

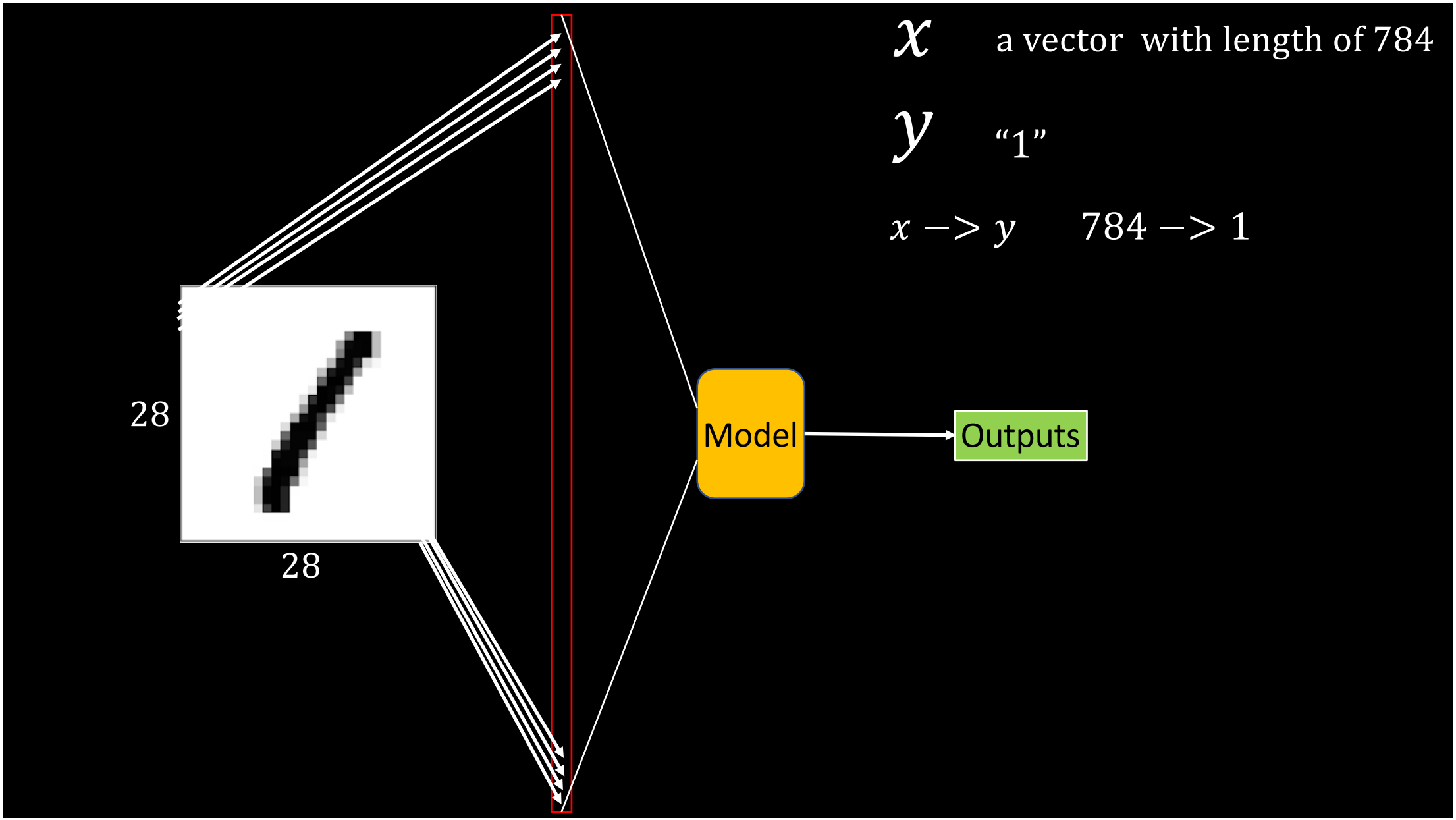
OrganMNIST (axial)



OrganMNIST (coronal)

OrganMNIST (sagittal)




x a vector with length of 784
y "1"
$x -> y$ 784 -> 1
28
28
Model
Outputs