IO_

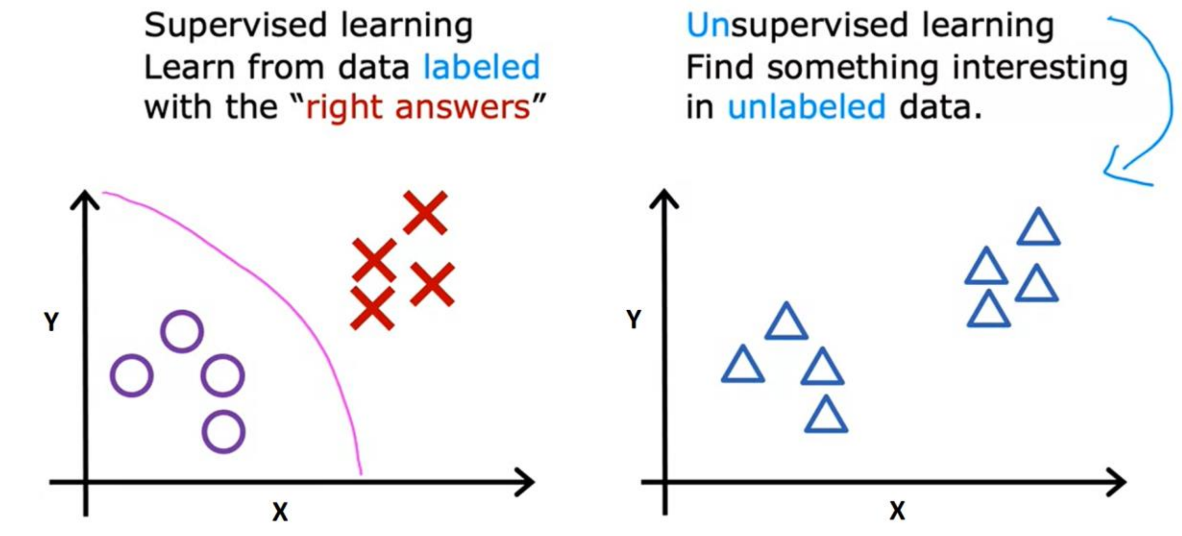
Supervised Learning Tutorials

Ifeanyi Anthony Okpala

Machine learning Algorithms

We have two main types:

- Supervised Learning
- Unsupervised Learning



Key Note:

In supervised learning, we include the expected output Target(y) to the learning algorithm.

Quiz 1

- S Given email labeled as spam/not spam, learn a spam filter.
- US Given a set of news articles found on the web, group them into sets of articles about the same story.
- US Given a database of customer data, automatically discover market segments and group customers into different market segments.
- S Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not

Supervised Learning

We have two types of Supervised Learning Models:

- Regression Model - Predict **numbers** as the output.

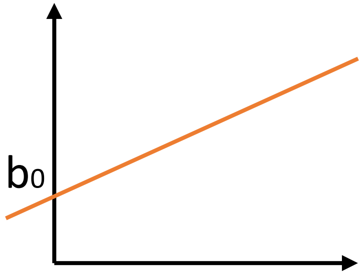
- Linear Regression (Simple Linear , **Multi Linear**)
- Non Linear Regression (Polynomial Regression Model)

- Classification Model - Predict **categories** as the output.

- Decision Tree
- Support Vector Machine
- **Logistic Regression**
- K Nearest Neighbours Algorithm (KNN)

Regression Model

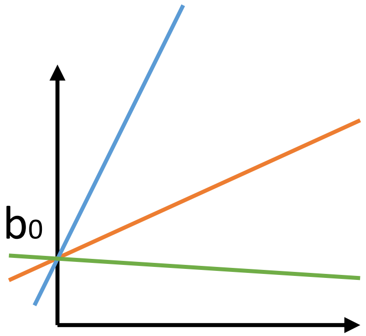
Simple Linear Regression



$$f(x) = w_1x_1 + b_0$$

X1	Y
5	500
1	200
3	150
4	300
7	700

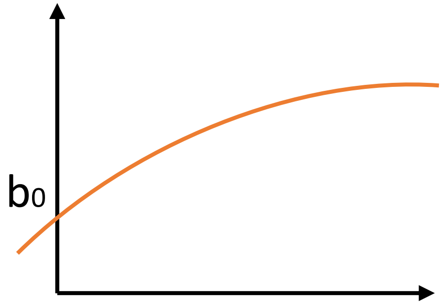
Multi Linear Regression



$$f(x) = w_1x_1 + w_2x_2 + w_3x_3 + b_0$$

X1	X2	X3	Y
5	1.3	20	500
1	1.1	5	200
3	3.2	10	150
4	1.2	15	300
7	2.5	40	700

Polynomial Regression



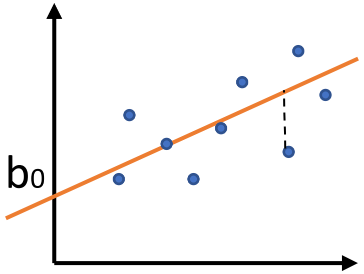
$$f(x) = w_1x_1 + w_2x_1{**}2 + b_0$$

X1	X1**2	Y
5	25	500
1	1	200
3	9	150
4	16	300
7	49	700

$\hat{y}$ = y(Predict) = f(x) dependent variable

w and **x** are vectors which is applicable in multilinear regression. $f(x) = \vec{W} \cdot \vec{X} + b_0$

Errors and Gradient Descent for Linear models

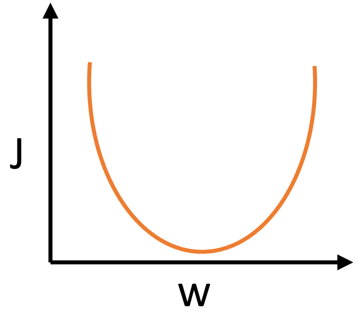


Error $= y - \hat{y}$

Cost function: Squared error cost function

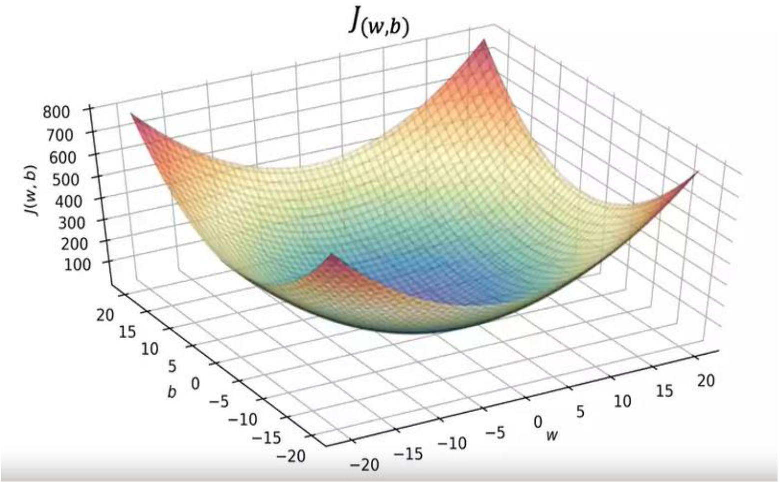
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\Big(\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{error}}\Big)^2$$

m = number of training examples



If b = 0, our cost function can be represented in 2D convex shape.

Our goal is to reduce the cost function by varying **w** and **b**



Errors and Gradient Descent Cont…

- Gradient decent is a systematic way to find the value of **w** and **b** to minimize the cost function J.
- It is used not just for linear regression but also deep learning models.

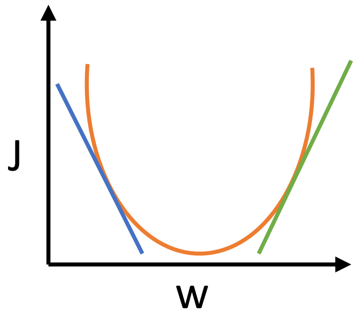
Gradient descent algorithm

repeat until convergence {

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

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

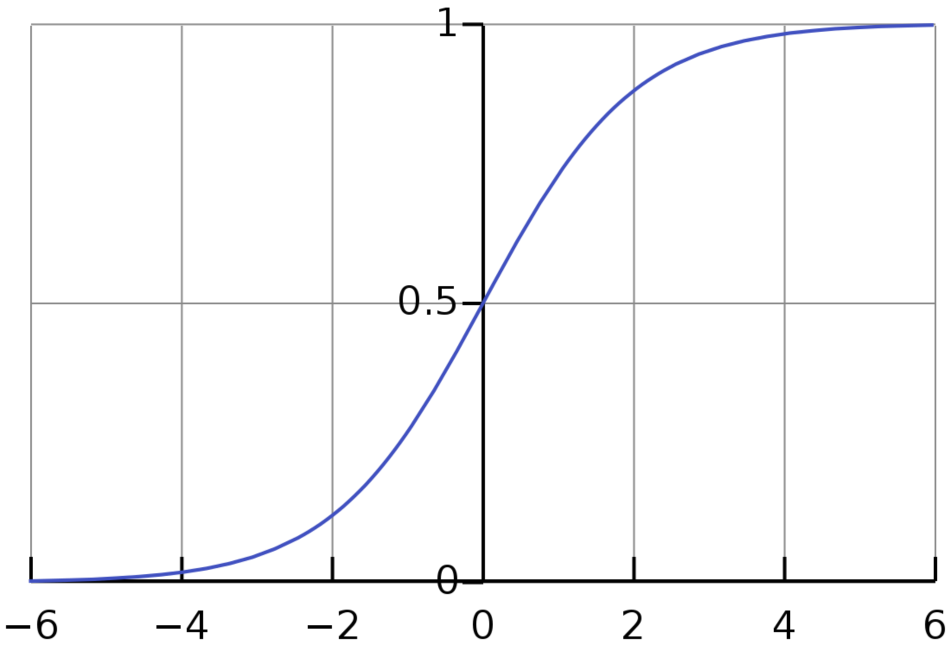
α = **learning rate**



We need **J** the cost function (error) to be at the minimum.

Logistic Regression

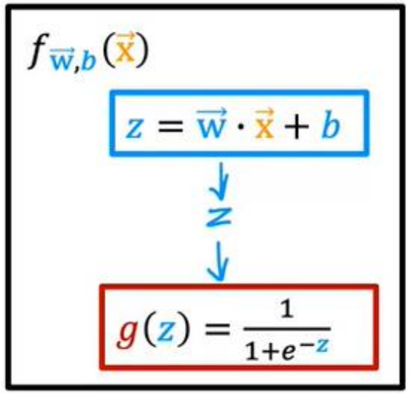
- Logistic regression is used to predict **binary categories** as the output.
- This regression model makes use of a **sigmoid function/ logistic function**.



If z is a very large positive number, the output will be very close to 1.

If z is equal to 0, the output will be 0.5

If z is a very large negative number, the output will be very close to 0.



$$\frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

Errors and Gradient Descent for Logistic Regression

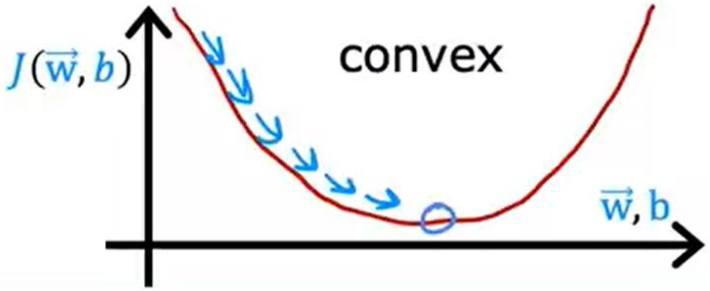
Squared error cost

$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2$$

loss $\boxed{L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)}$

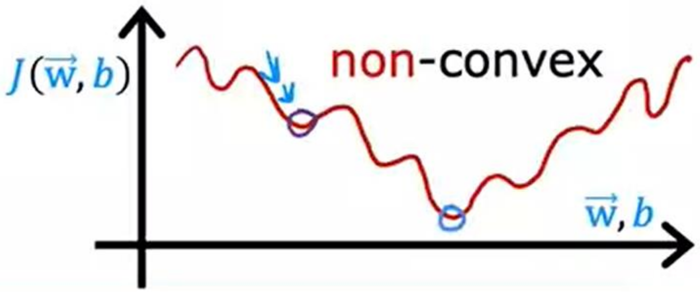
linear regression

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



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$



Errors and Gradient Descent for Logistic Regression Cont…

cost

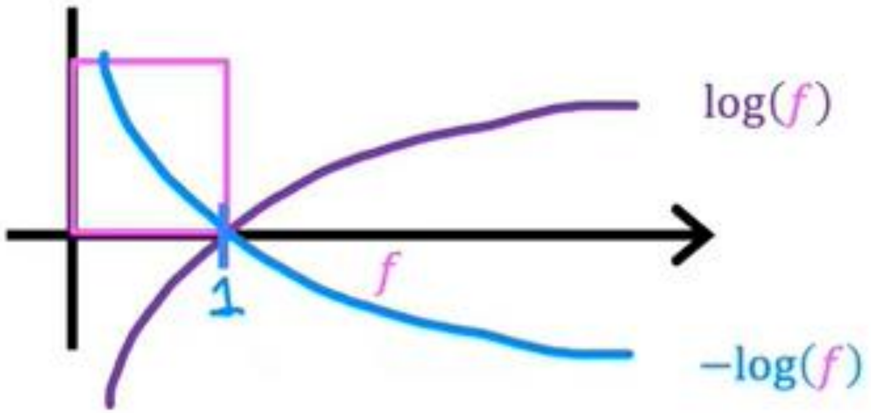
$$J(\vec{w}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)\right]$$

repeat {

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

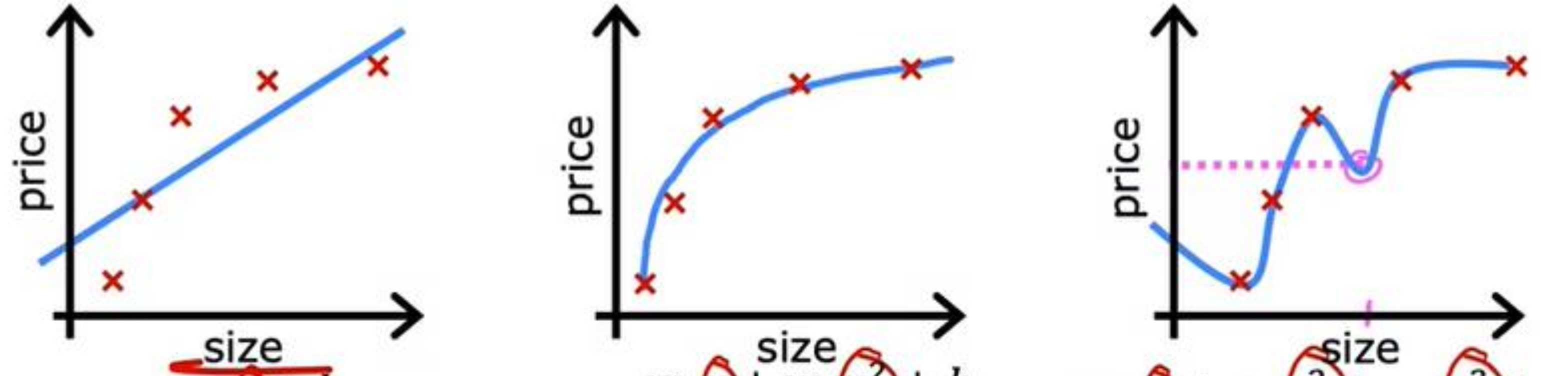
$$b = b - \alpha\frac{\partial}{\partial b}J(\vec{w}, b)$$

}



Overfitting and Underfitting for Regression

Regression example



$w_1x + b$

underfit

- Does not fit the training set well

high bias

$w_1x + w_2x^2 + b$

just right

- Fits training set pretty well

generalization

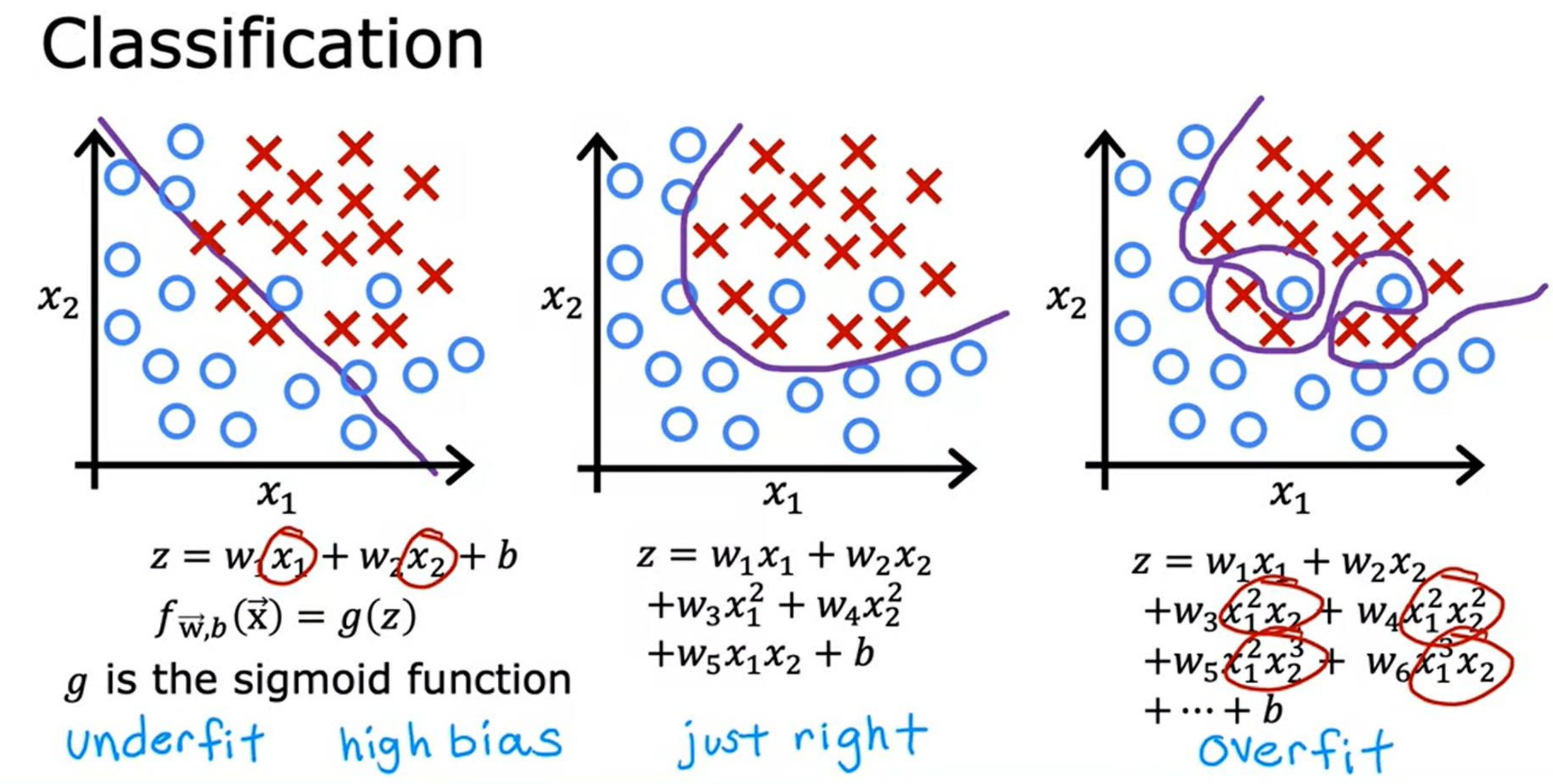
$w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

overfit

- Fits the training set extremely well

high variance

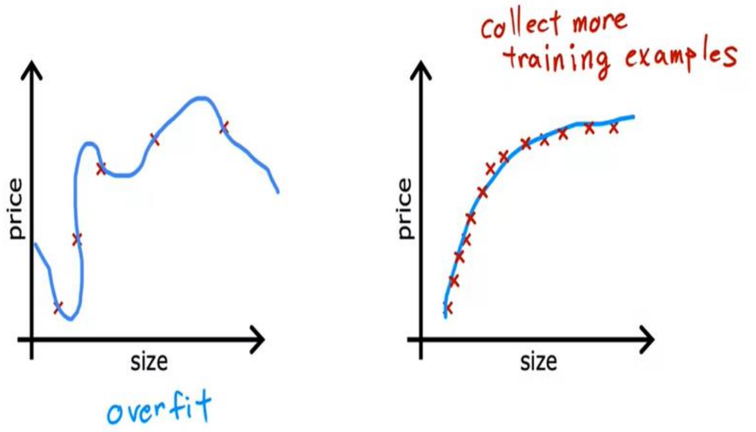
Overfitting and Underfitting for Classification



How to Address Overfitting

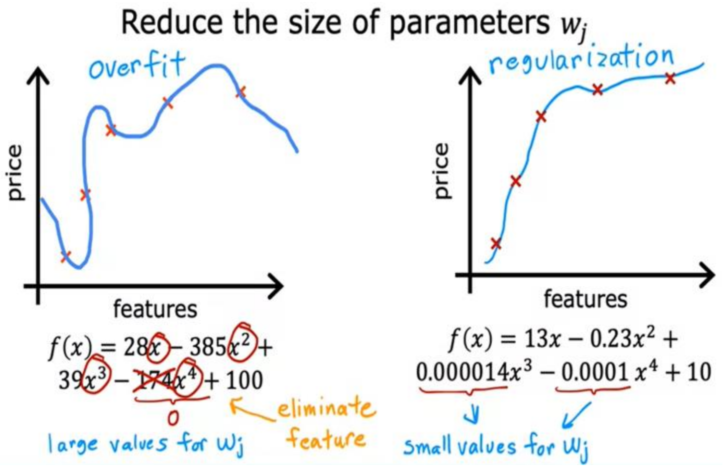
- Collecting more data – If more data are not available, use other methods.
- Feature selection – using a sub-set of the feature **x**.
- Regularization – Reducing the size of the parameters (**w** and **b**).

Collecting more data



Feature selection

Regularization

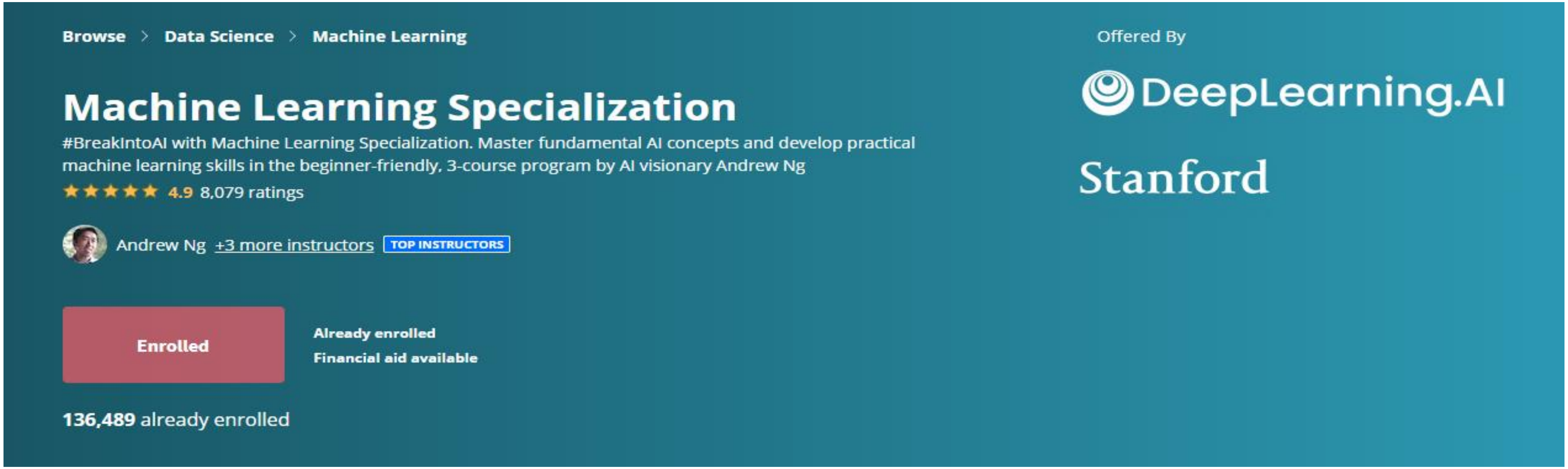


Material Source

https://www.coursera.org/learn/machine-learning

I strongly recommend this course taught by Andrew Ng via Coursera.

If you can’t afford the course, you can apply for the **Financial Aid.**



C1_W3_Lab07_Scikit_Learn_Soln

February 10, 2023

1 Ungraded Lab: Logistic Regression using Scikit-Learn

1.1 Goals

In this lab you will: - Train a logistic regression model using scikit-learn.

1.2 Dataset

Let's start with the same dataset as before.

[1]:
```
import numpy as np

X = np.array([[0.5, 1.5], [1,1], [1.5, 0.5], [3, 0.5], [2, 2], [1, 2.5]])
y = np.array([0, 0, 0, 1, 1, 1])
```

[2]:
```
X.shape
```

[2]:
```
(6, 2)
```

1.3 Fit the model

The code below imports the logistic regression model from scikit-learn. You can fit this model on the training data by calling `fit` function.

[3]:
```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()
lr_model.fit(X, y)
```

[3]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

1.4 Make Predictions

You can see the predictions made by this model by calling the `predict` function.

[4]:
```
y_pred = lr_model.predict(X)

print("Prediction on training set:", y_pred)
```

```
Prediction on training set: [0 0 0 1 1 1]
```

1.5 Calculate accuracy

You can calculate this accuracy of this model by calling the `score` function.

[5]:
```
print("Accuracy on training set:", lr_model.score(X, y))
```

```
Accuracy on training set: 1.0
```