Multimedia Content Analysis

Jeripothula Prudviraj

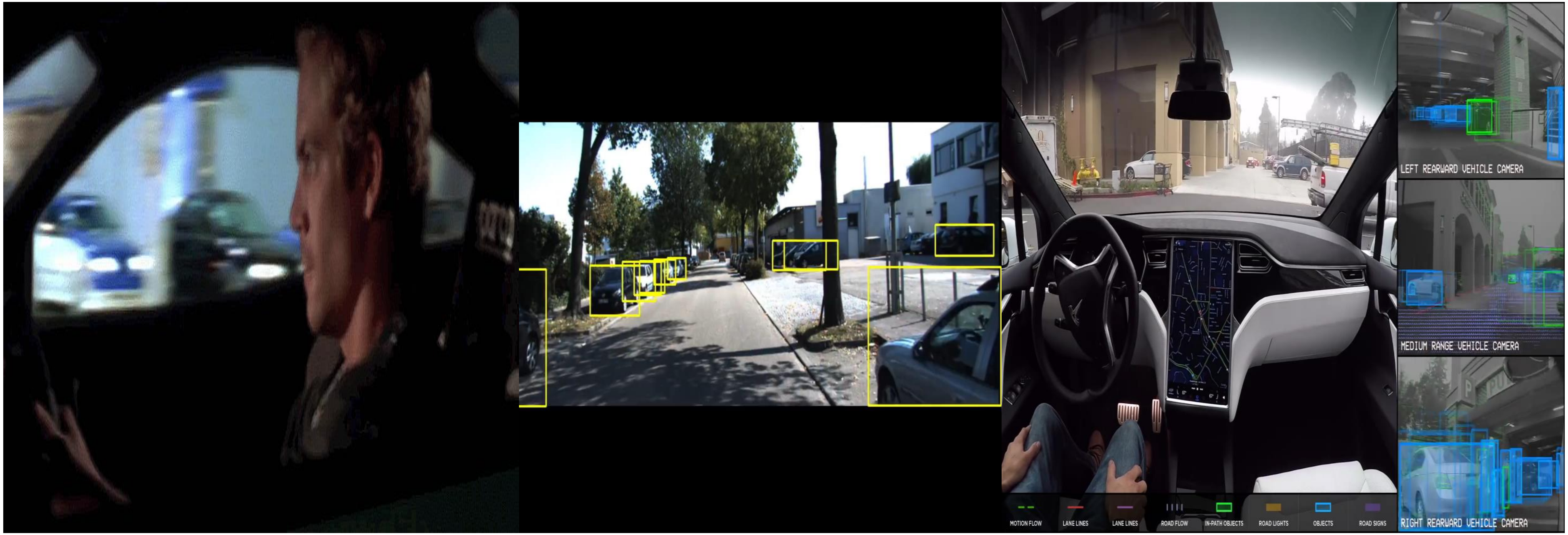
https://sites.google.com/view/theswath/home

Introduction

- Artificial intelligence is applied when a machine mimics "cognitive" functions that humans associate with other human minds

Introduction

- A machine mimics humans “cognition”

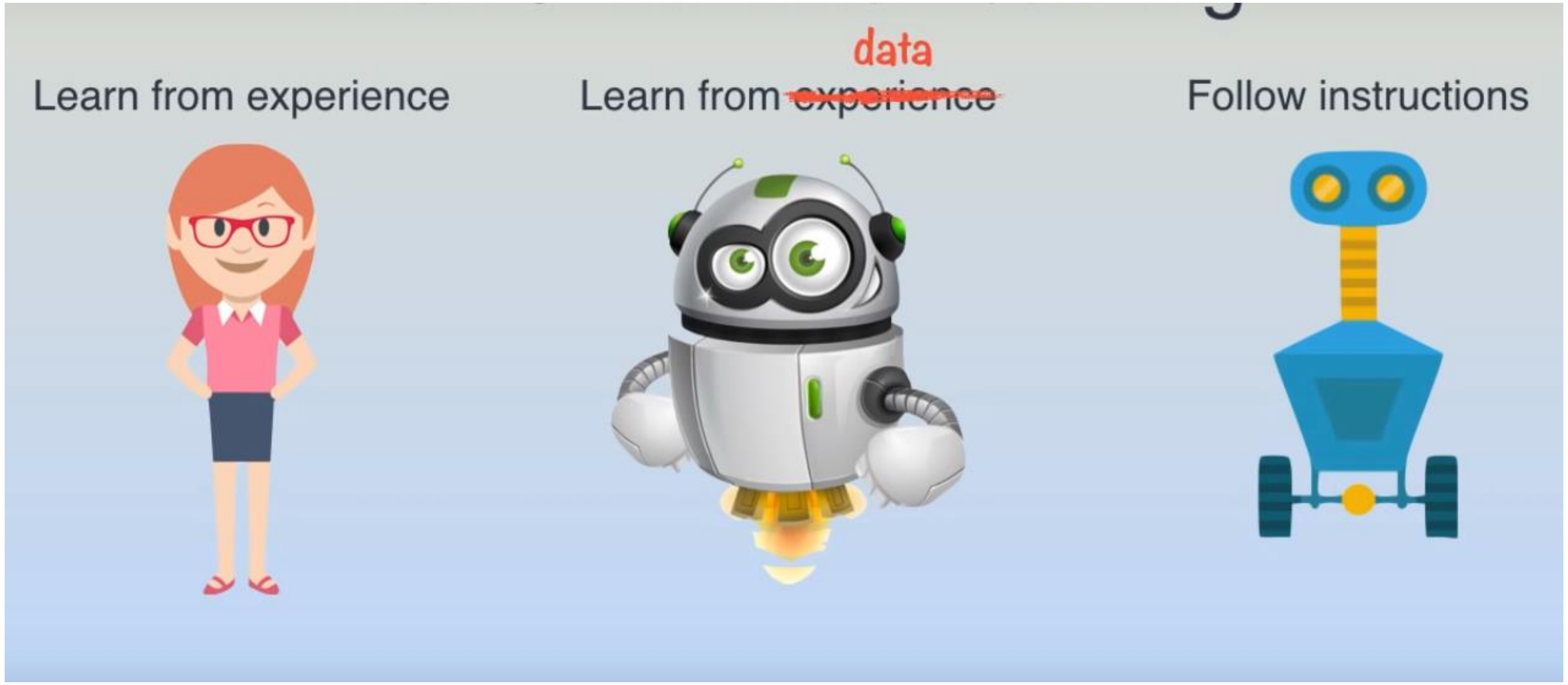


Motivation

- Cognition: Cognition is “the mental action or process of acquiring knowledge and understanding through thought, experience, and the senses”
- To make machine mimic “cognitive functions”, it has to understand human activities or behaviour

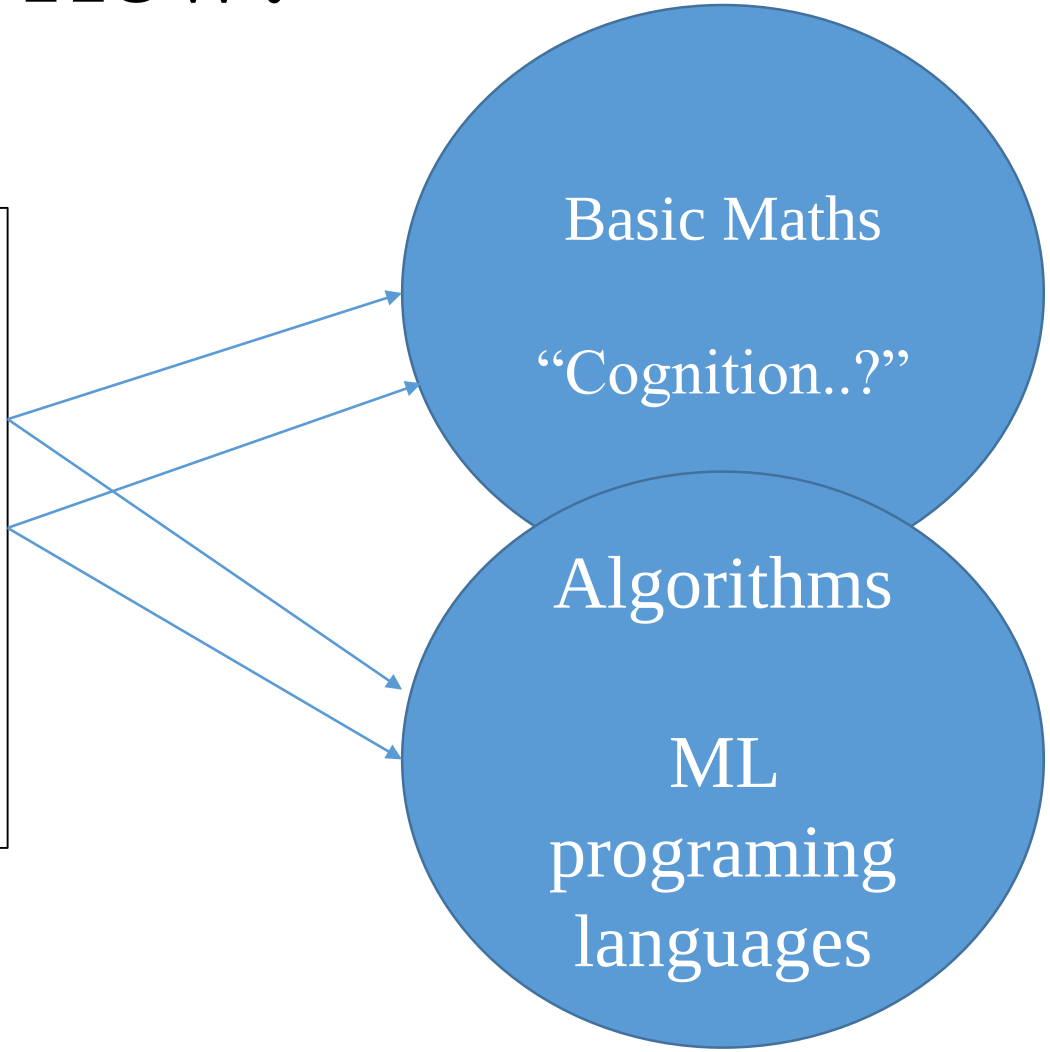
What is ML

- Tom Mitchell (1998) : A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if it's performance on T, as measured by P, improves with experience E.
-



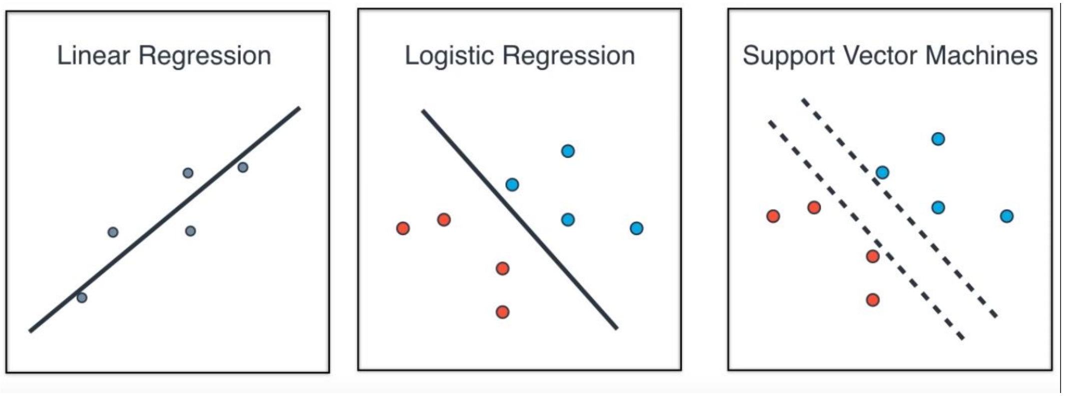
How?

- ✓ Data Collection
- ✓ **Define the problem Intuitively**
- ✓ **ML Algorithms**
- ✓ Optimize and fine-tune

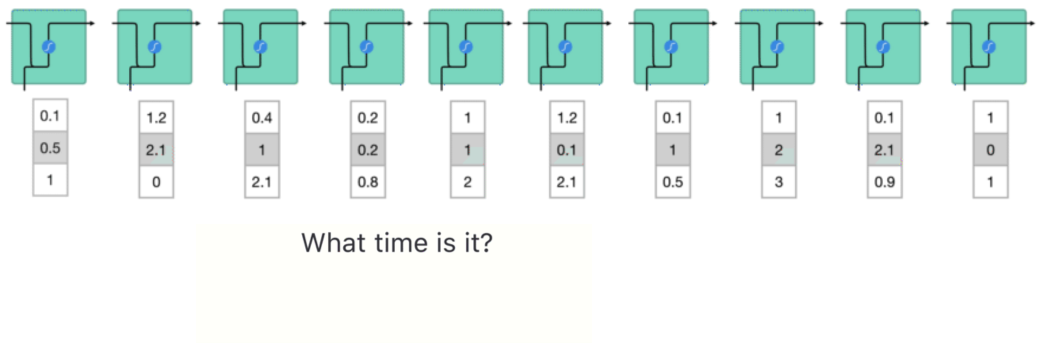


Data

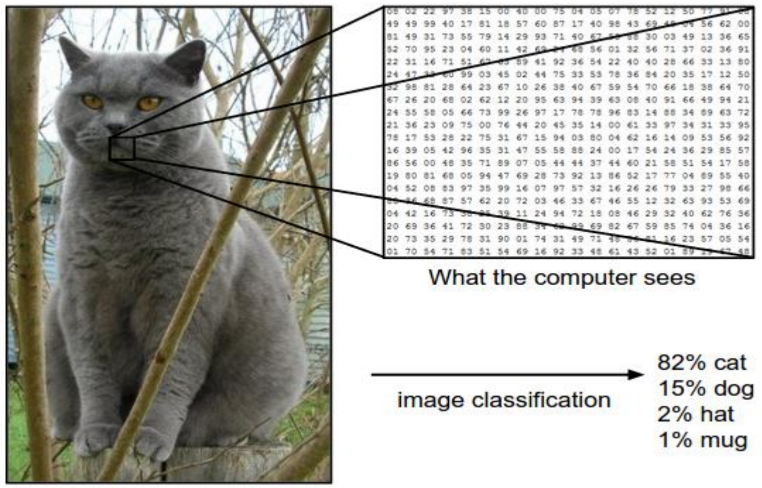
Scalar



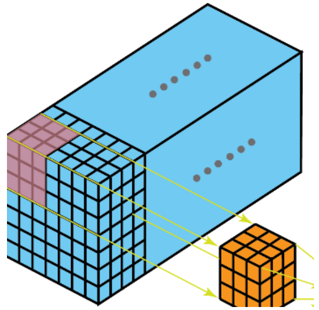
Vector



2D, Matrix

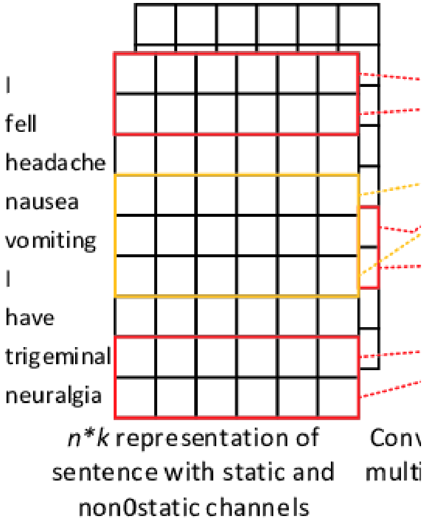


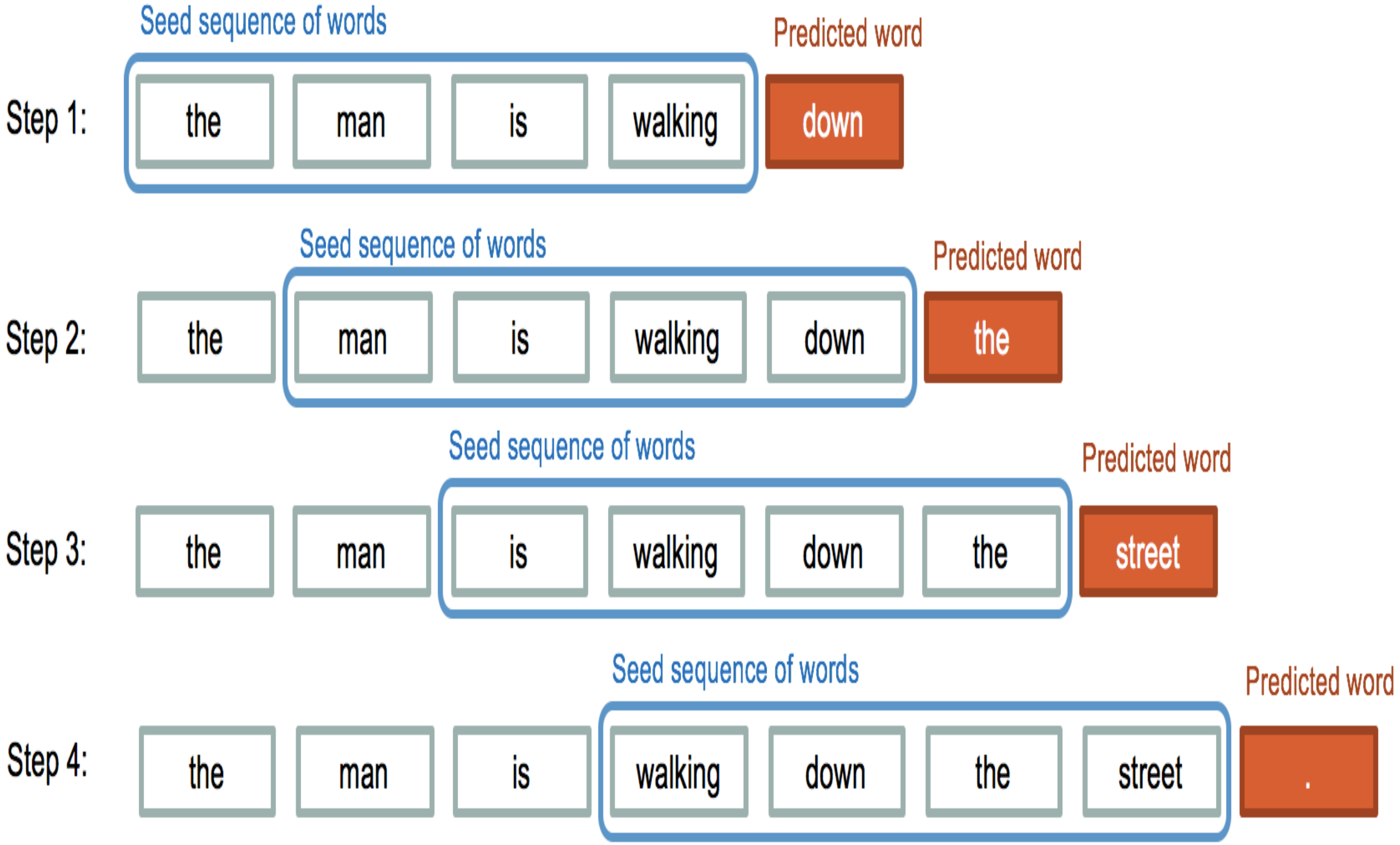
3D, Matrix



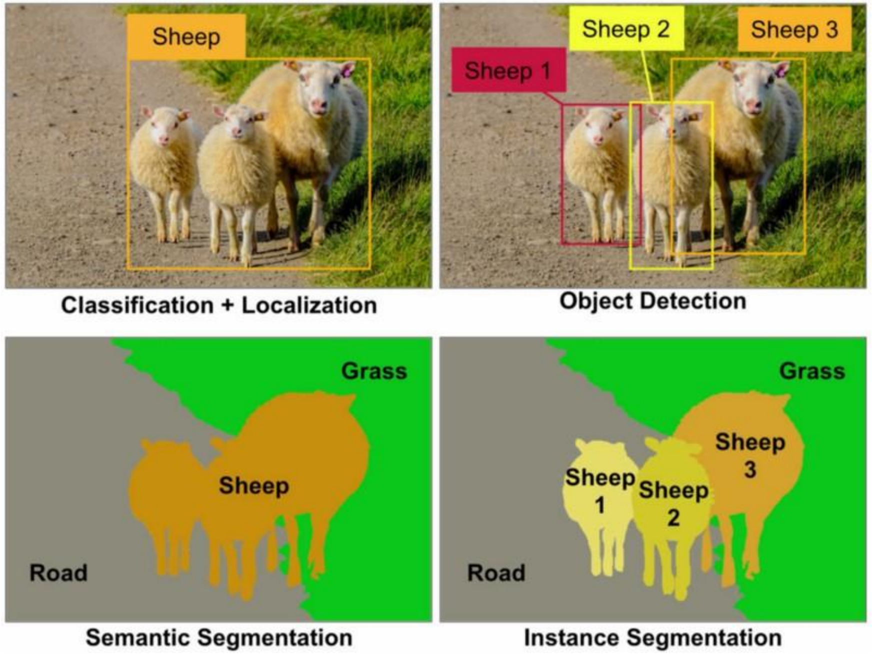
Tasks

	Open	High	Low	Close	Volume
0	0.6277	0.6362	0.6201	0.6201	2575579
1	0.6201	0.6201	0.6122	0.6201	1764749
2	0.6201	0.6201	0.6037	0.6122	2194010
3	0.6122	0.6122	0.5798	0.5957	3255244
4	0.5957	0.5957	0.5716	0.5957	3696430
5	0.5957	0.6037	0.5878	0.5957	2778285
6	0.5957	0.6037	0.5957	0.5957	2337096

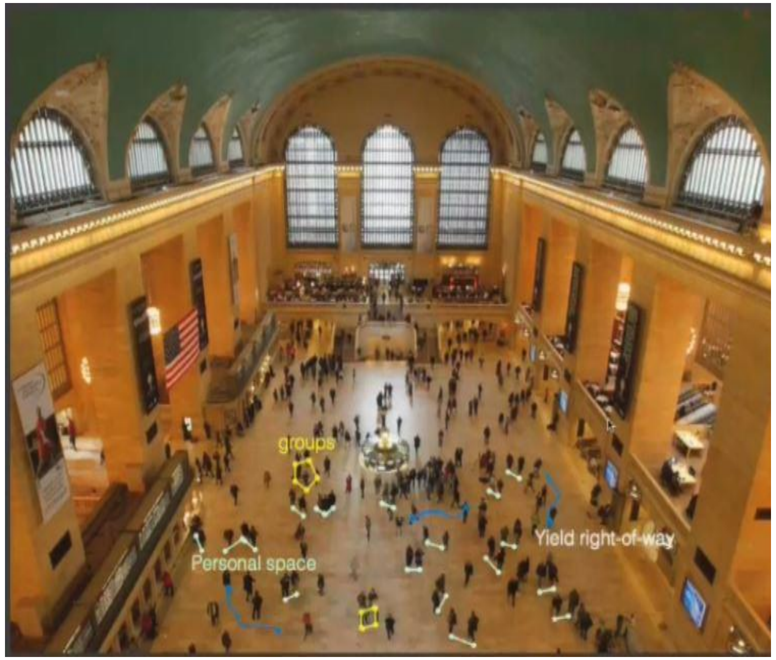




Tasks



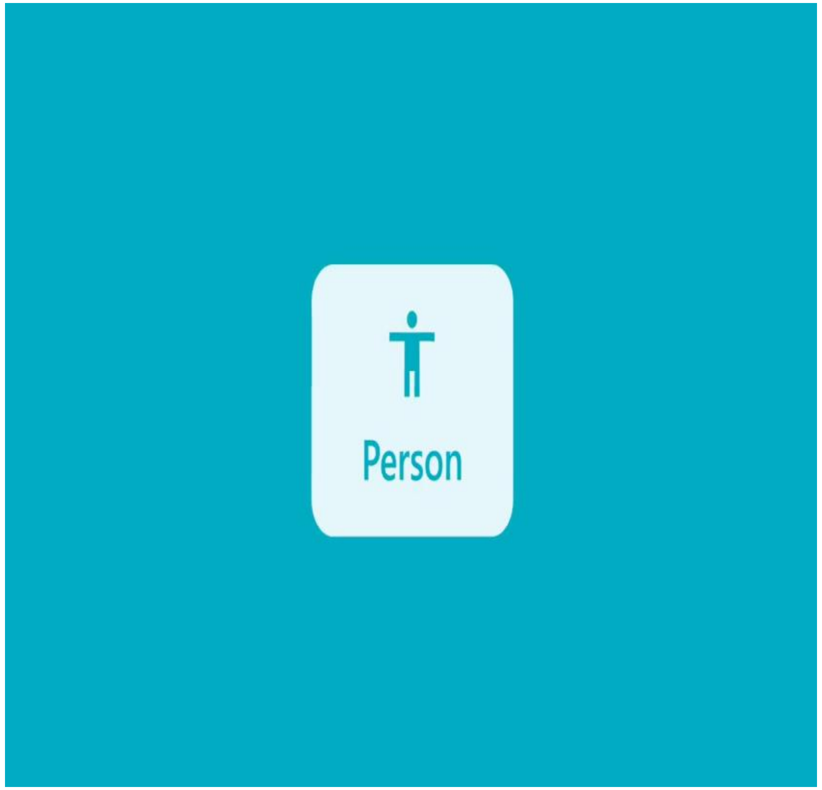
Image/Video Understanding

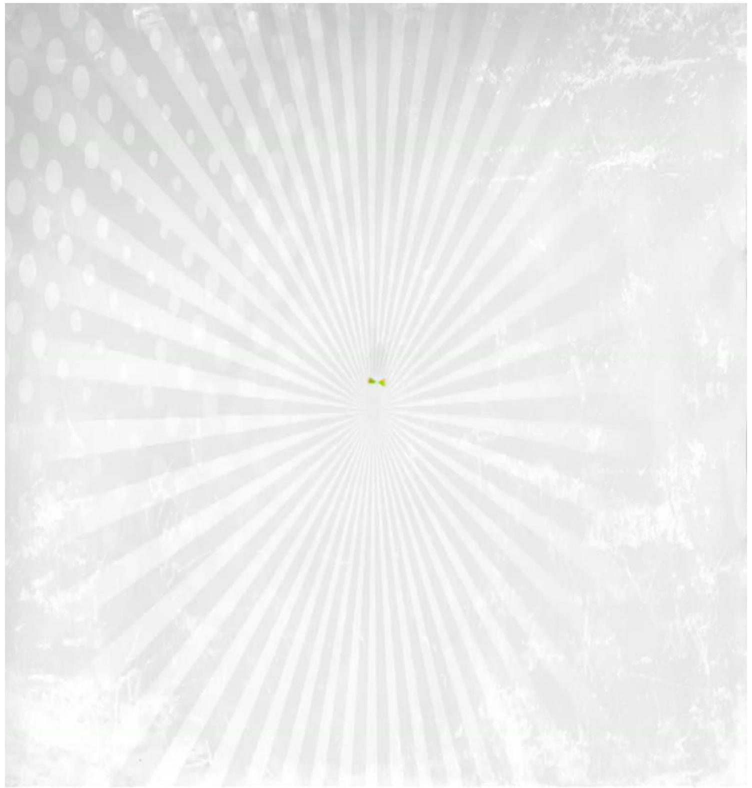


Learning Semantic Behaviour

Image/Video Inferencing

Applications





Intelligent Transportation System

Vision to Language Tasks

Recommendation Systems

Prerequisites

- Linear Algebra by ***Gilbert Strang (MIT)***
- Probability & Calculus from ***3Blue 1 Brown***
- For Deep learning and ML
    - ***Deep learning by Ian Goodfellow and Yoshua Bengio***
    - ***Deeplearning.ai, Stanford (cs231n), MIT, Udacity, edureka, The center for Minds, Brains, and Machines (CBMM), Coursera.***
- PyTorch by ***freeCodeCamp.org, github, patreon.com/patrickloeber, ml-cheatsheet.readthedocs.io***
- Research articles from ***medium.com, CVF open access, iclr.cc, eccv.eu, nature.com, neurips.cc***
- Additional courses: Neuro science, General Psychology, ***(intro to psych by John Gabrieli and Behavioral by Robert sapolsky)***Physics, Maths.

Linear Regression

- Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope.
- It's used to predict values within a continuous range, (e.g. sales, price)
- Simple linear regression uses traditional slope-intercept form,

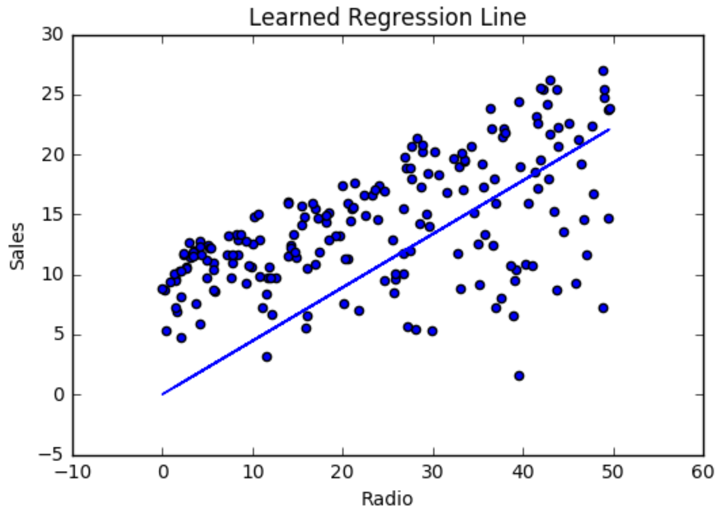
$$y = mx + b$$

where ***'m', 'b'*** are the variables our algorithm will try to ***"learn"*** to produce the most accurate predictions. ***'x'*** represents our input data and ***'y'*** represents our prediction.

Linear Regression

- Let's say we are given with a dataset with the following columns (features):

Company	Radio ($)	Sales
Amazon	37.8	22.1
Google	39.3	10.4
Facebook	45.9	18.3
Apple	41.3	18.5



Our prediction function

$$Sales = Weight \cdot Radio + Bias$$

Linear Regression

Our prediction function

$$Sales = Weight \cdot Radio + Bias$$

Weight:

The coefficient for the Radio independent variable. In machine learning we call coefficients weights.

Radio:

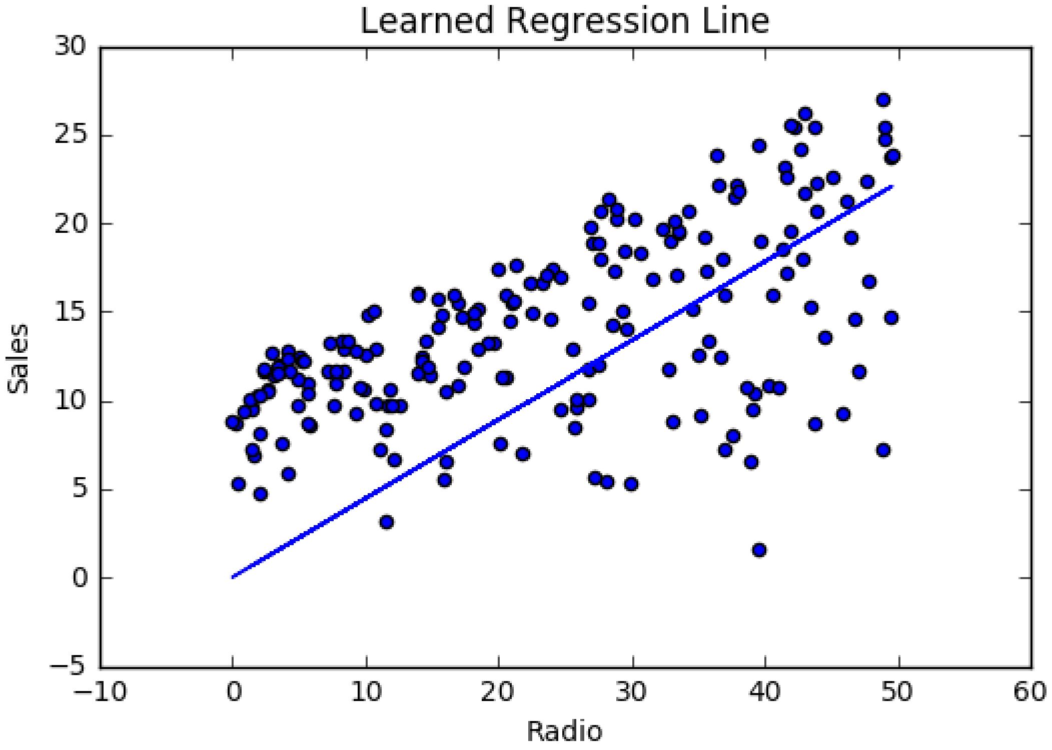
The independent variable. In machine learning we call these variables features.

Bias:

The intercept where our line intercepts the y-axis. In machine learning we can call intercepts bias.

Linear Regression

- Given prediction function, Our algorithm will try to *learn* the correct values for Weight and Bias



Linear Regression

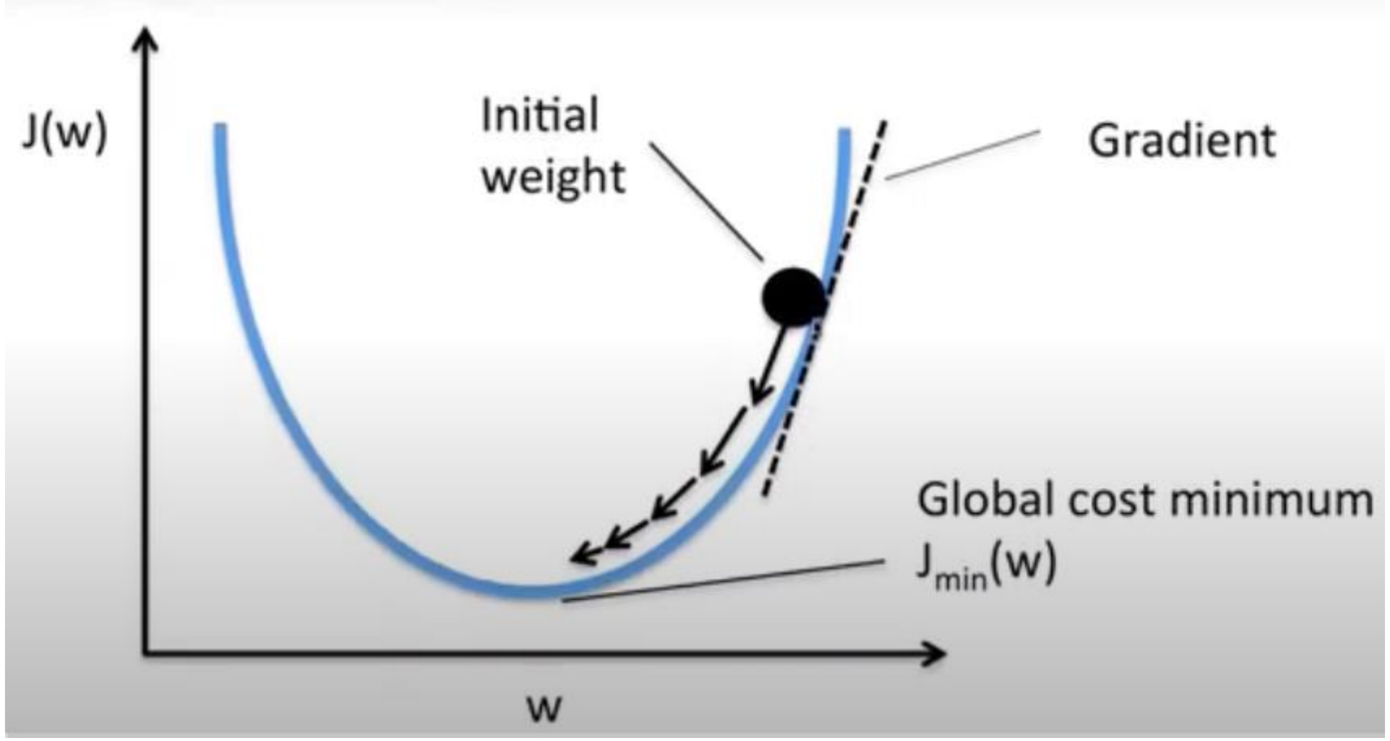
- Approximation $\hat{y} = wx + b$
- Cost Function

$$MSE = J(w, b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

$$J'(m, b) = \begin{bmatrix} \frac{df}{dw} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum -2x_i(y_i - (wx_i + b)) \\ \frac{1}{N}\sum -2(y_i - (wx_i + b)) \end{bmatrix}$$

Linear Regression

- Gradient descent



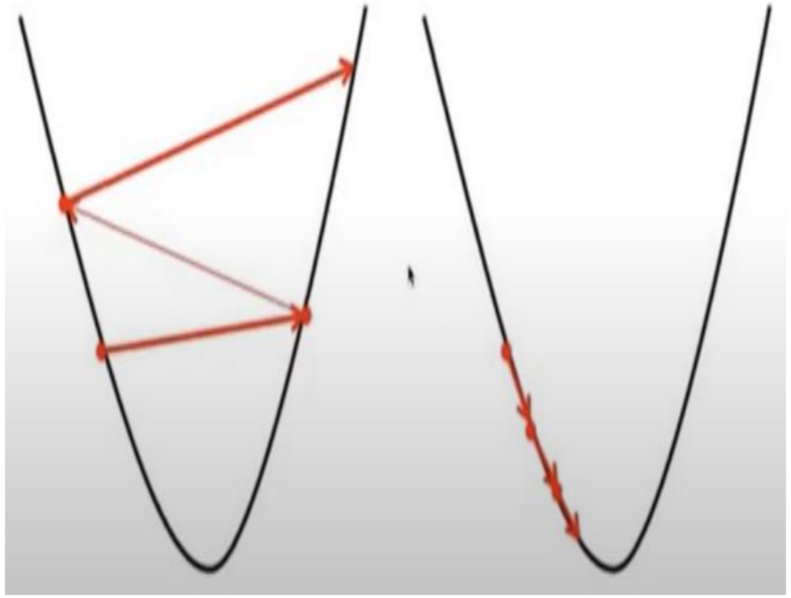
Linear Regression

- Update rules

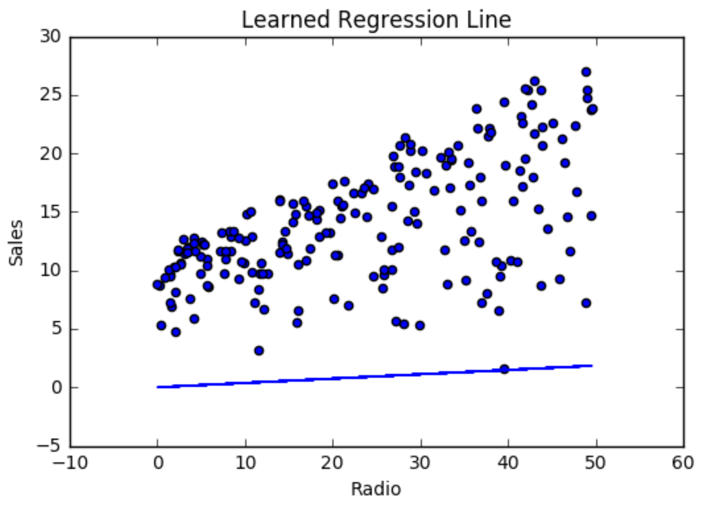
$$w = w - \alpha \cdot dw$$
$$b = b - \alpha \cdot db$$

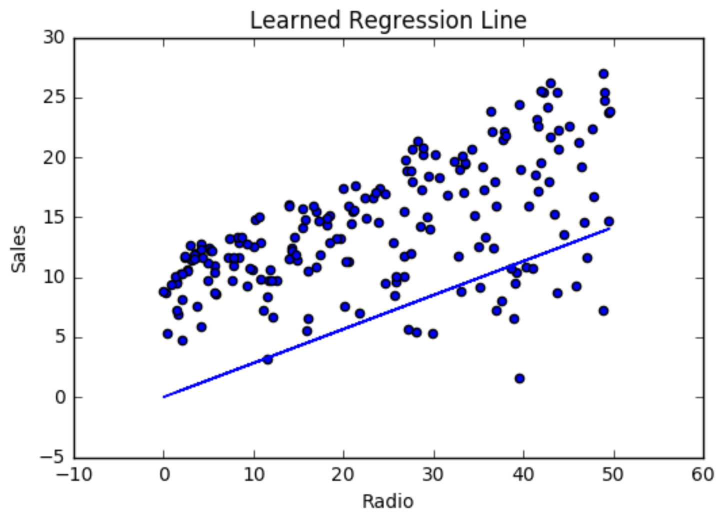
$$\frac{dJ}{dw} = dw = \frac{1}{N}\sum_{i=1}^{n} -2x_i(y_i - (wx_i + b)) = \frac{1}{N}\sum_{i=1}^{n} -2x_i(y_i - \hat{y}) = \frac{1}{N}\sum_{i=1}^{n} 2x_i(\hat{y} - y_i)$$

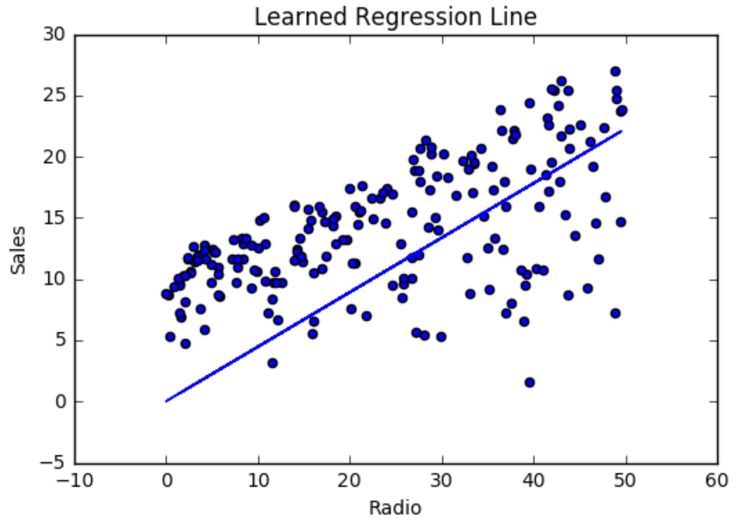
$$\frac{dJ}{db} = db = \frac{1}{N}\sum_{i=1}^{n} -2(y_i - (wx_i + b)) = \frac{1}{N}\sum_{i=1}^{n} -2(y_i - \hat{y}) = \frac{1}{N}\sum_{i=1}^{n} 2(\hat{y} - y_i)$$

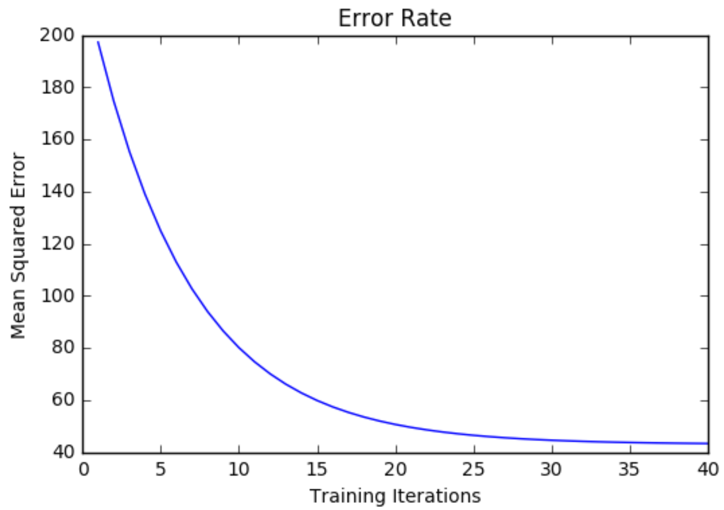


Linear Regression









Linear Regression

```python
class LinearRegression:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
```

```
def fit(self, X, y):
    n_samples, n_features = X.shape

    # init parameters
    self.weights = np.zeros(n_features)
    self.bias = 0

    # gradient descent
    for _ in range(self.n_iters):
        y_predicted = np.dot(X, self.weights) + self.bias
        # compute gradients
        dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
        db = (1 / n_samples) * np.sum(y_predicted - y)

        # update parameters
        self.weights -= self.lr * dw
        self.bias -= self.lr * db
```

$$\frac{dJ}{dw} = dw = \frac{1}{N}\sum_{i=1}^{n} -2x_i(y_i - (wx_i + b)) = \frac{1}{N}\sum_{i=1}^{n} -2x_i(y_i - \hat{y}) = \frac{1}{N}\sum_{i=1}^{n} 2x_i(\hat{y} - y_i)$$

$$\frac{dJ}{db} = db = \frac{1}{N}\sum_{i=1}^{n} -2(y_i - (wx_i + b)) = \frac{1}{N}\sum_{i=1}^{n} -2(y_i - \hat{y}) = \frac{1}{N}\sum_{i=1}^{n} 2(\hat{y} - y_i)$$

$$w = w - \alpha \cdot dw$$
$$b = b - \alpha \cdot db$$

```python
def predict(self, X):
    y_approximated = np.dot(X, self.weights) + self.bias
    return y_approximated
```

```python
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets

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

X, y = datasets.make_regression(
    n_samples=100, n_features=1, noise=20, random_state=4
)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1234
)
```

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1234
)

regressor = LinearRegression(learning_rate=0.01, n_iters=1000)
regressor.fit(X_train, y_train)
predictions = regressor.predict(X_test)

mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)

accu = r2_score(y_test, predictions)
print("Accuracy:", accu)
```

Pytorch

1) Design model (input, output, forward pass with different layers)
2) Construct loss and optimizer
3) Training loop
    - Forward = compute prediction and loss
    - Backward = compute gradients
    - Update weights

```python
import torch
import torch.nn as nn

# Linear regression
# f = w * x

# here : f = 2 * x

# 0) Training samples
X = torch.tensor([1, 2, 3, 4], dtype=torch.float32)
Y = torch.tensor([2, 4, 6, 8], dtype=torch.float32)

# 1) Design Model: Weights to optimize and forward function
w = torch.tensor(0.0, dtype=torch.float32, requires_grad=True)
```

```python
def forward(x):
    return w * x

print(f'Prediction before training: f(5) = {forward(5).item():.3f}')

# 2) Define loss and optimizer
learning_rate = 0.01
n_iters = 100

# callable function
loss = nn.MSELoss()

optimizer = torch.optim.SGD([w], lr=learning_rate)
```

```python
# 3) Training loop
for epoch in range(n_iters):
    # predict = forward pass
    y_predicted = forward(X)

    # loss
    l = loss(Y, y_predicted)

    # calculate gradients = backward pass
    l.backward()

    # update weights
    optimizer.step()

    # zero the gradients after updating
    optimizer.zero_grad()

    if epoch % 10 == 0:
        print('epoch ', epoch+1, ': w = ', w, ' loss = ', l)

print(f'Prediction after training: f(5) = {forward(5).item():.3f}')
```

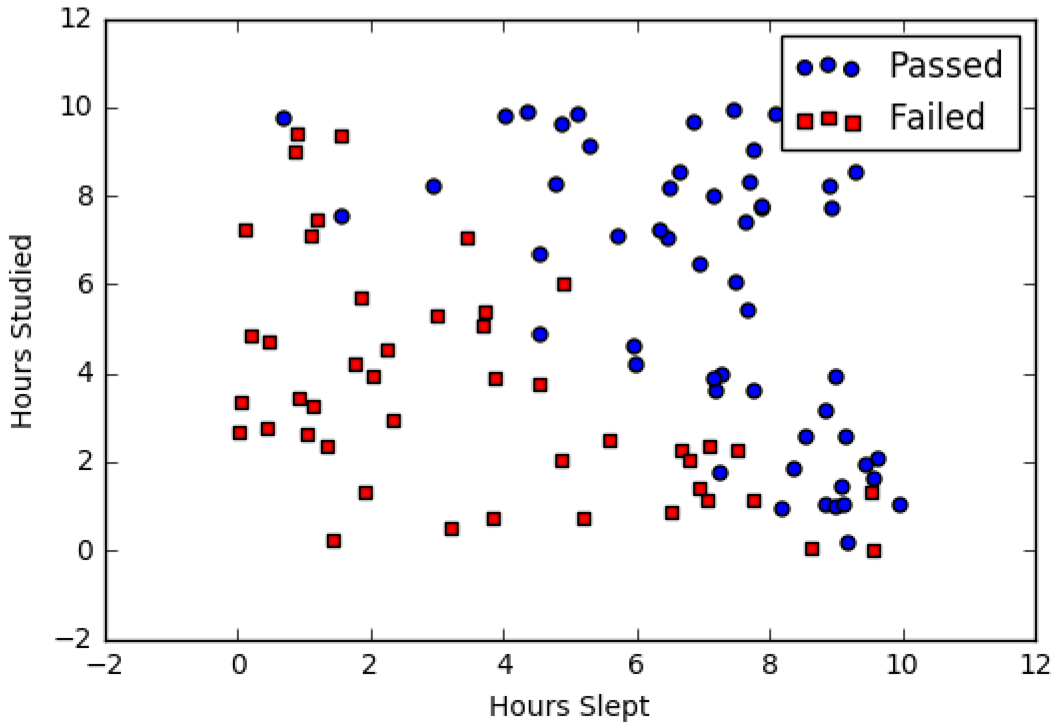
Logistic regression

- Logistic regression is a classification algorithm used to assign observations to a discrete set of classes.
- Unlike *linear regression* which outputs *continuous number values*, *logistic regression* transforms its output using the logistic sigmoid function to return a *probability value* which can then be mapped to two or more discrete classes.
- Linear Regression could help us predict the student's test score on a scale of 0 - 100.
- Logistic Regression could help use predict whether the student passed or failed

Logistic regression

- We're given data on student exam results and our goal is to predict whether a student will pass or fail based on number of hours slept and hours spent studying.

Studied	Slept	Passed
4.85	9.63	1
8.62	3.23	0
5.43	8.23	1
9.21	6.34	0



Logistic regression

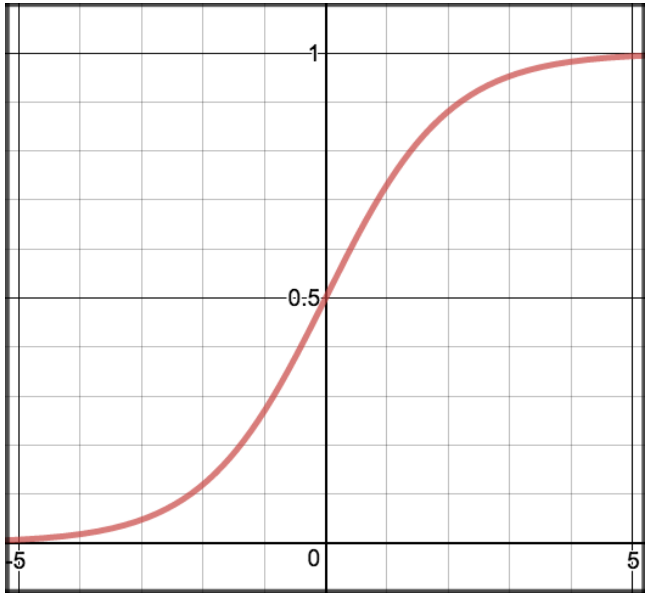
- Approximation

$$f(w,b) = wx + b$$

$$\hat{y} = h_\theta(x) = \frac{1}{1 + e^{-wx+b}}$$

- Sigmoid function

$$s(x) = \frac{1}{1 + e^{-x}}$$



Logistic regression

- Cost function: we use a cost function called Cross-Entropy, also known as Log Loss

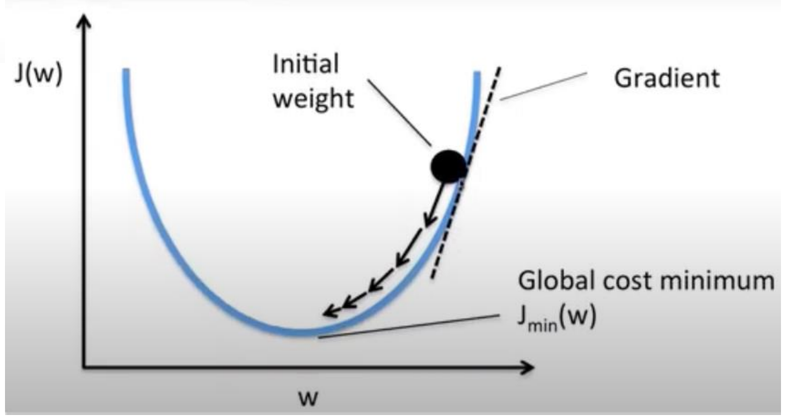
$$\mathrm{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \quad \text{if } y = 1$$
$$\mathrm{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \quad \text{if } y = 0$$

$$J(w, b) = J(\theta) = \frac{1}{N} \sum_{i=1}^{n} [y^i log(h_\theta(x^i)) + (1 - y^i) log(1 - h_\theta(x^i))]$$

- Update rules

$$w = w - \alpha \cdot dw$$
$$b = b - \alpha \cdot db$$

$$J'(\theta) = \begin{bmatrix} \frac{dJ}{dw} \\ \frac{dJ}{db} \end{bmatrix} = [\ldots] = \begin{bmatrix} \frac{1}{N} \sum 2x_i(\hat{y} - y_i) \\ \frac{1}{N} \sum 2(\hat{y} - y_i) \end{bmatrix}$$



https://medium.com/analytics-vidhya/derivative-of-log-loss-function-for-logistic-regression-9b832f025c2d

Logistic regression

```python
# gradient descent
for _ in range(self.n_iters):
    # approximate y with linear combination of weights and x, plus bias
    linear_model = np.dot(X, self.weights) + self.bias
    # apply sigmoid function
    y_predicted = self._sigmoid(linear_model)

    # compute gradients
    dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
    db = (1 / n_samples) * np.sum(y_predicted - y)
    # update parameters
    self.weights -= self.lr * dw
    self.bias -= self.lr * db
```

```python
# 1) Model
# Linear model f = wx + b , sigmoid at the end
class Model(nn.Module):
    def __init__(self, n_input_features):
        super(Model, self).__init__()
        self.linear = nn.Linear(n_input_features, 1)

    def forward(self, x):
        y_pred = torch.sigmoid(self.linear(x))
        return y_pred

model = Model(n_features)
```